ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΕΡΓΑΣΙΕΣ

ΨΗΦΙΚΗ ΕΠΕΞΕΡΓΑΣΙΑ ΒΙΝΤΕΟ

ΑΣΚΗΣΗ 1

Ονοματεπώνυμο:	Βασιλική Στάμου
Αριθμός Μητρώου:	1059543
Τομέας:	Μεταπτυχιακό ΣΜΗΝ
Έτος φοίτησης:	1ο
Ακ. έτος διεξαγωγής μαθήματος:	2023-2024
Ημ/νία διεξαγωγής Φ.Α.:	03.04.2024

Περιεχόμενα

1 Υλοποιήστε αλγόριθμο οπτικής ροής χρησιμοποιώντας τη μέθοδο Lucas – Kanade. Θα πρέπει να βρείτε τα διανύσματα κίνησης (u,v) για κάθε pixel. 1

2 Επαναχρησιμοποιήστε τον κώδικα για να υλοποιήσετε έναν ανιχνευτή γωνιών λαμβάνοντας υπόψη το μέγεθος των ιδιοτιμών του πίνακα A^TA. 3

3 Χρησιμοποιήστε τα 1,2 ώστε να βρείτε την οπτική ροή μόνο στις γωνίες (αραιή οπτική ροή). 5

4 Σταθεροποίηση βίντεο 6

4.1 Μετασχηματισμός ανά pixel λαμβάνοντας υπόψη την οπτική ροή από frame σε frame. 6

4.2 Μετασχηματισμός με βάση τις γωνίες και με χρήση μοντέλου affine. 7

5 Προαιρετικό 9

6 Κώδικας 9

6.1 Αλγόριθμος Οπτικής Ροής 9

6.2 Αλγόριθμος Ανίχνευσης Ακμών 12

6.3 Αλγόριθμος Αραιής Οπτικής Ροής 13

6.4 Σταθεροποίηση βίντεο με οπτική ροή 15

6.5 Σταθεροποίηση βίντεο με μετασχηματισμό affine 16

6.6 Σταθεροποίηση βίντεο από drone 19

1 Υλοποιήστε αλγόριθμο οπτικής ροής χρησιμοποιώντας τη μέθοδο Lucas – Kanade. Θα πρέπει να βρείτε τα διανύσματα κίνησης (u,v) για κάθε pixel.

Το πρόβλημα της οπτικής ροής αναφέρεται στον υπολογισμό της ταχύτητας και της κατεύθυνσης κίνησης κάθε pixel μιας εικόνας καθώς αυτή μεταβαίνει σε μια διαδοχική εικόνα.

Υλοποιούμε την συνάρτηση: **[u,v] = opticalFlow_implimentation(im1,im2,windowSize)**

- im1, im2: διαδοχικές εικόνες
- windowSize: το μέγεθος του παραθύρου που εξετάζουμε
- u,v: συνιστώσες οπτικής ροής

Βήματα υλοποίησης:

1. **Μετατροπή των εικόνων σε διπλή ακρίβεια:** Για την εξασφάλιση ακριβών υπολογισμών, οι εικόνες μετατρέπονται σε μορφή διπλής ακρίβειας (double).
2. **Υπολογισμός των παραγώγων των εικόνων:** Οι παράγωγοι των εικόνων ως προς τις συντεταγμένες x και y, καθώς και η διαφορά τους (ως προς το χρόνο t), υπολογίζονται μέσω συνελίξεων με κατάλληλους πυρήνες:
   Ix: Παράγωγος κατά μήκος του άξονα x.
   Iy: Παράγωγος κατά μήκος του άξονα y.
   It: Διαφορά των εικόνων, η οποία αντιπροσωπεύει την παράγωγο ως προς το χρόνο.
3. **Δημιουργία Gaussian παραθύρου:** Ένα gaussian φίλτρο χρησιμοποιείται για να δώσει μεγαλύτερο βάρος στα κεντρικά σημεία του παραθύρου και να εξομαλύνει τις τιμές των παραγώγων.
4. **Υπολογισμός των γινομένων των παραγώγων:** Τα γινόμενα των παραγώγων υπολογίζονται για κάθε pixel, τα οποία στη συνέχεια φιλτράρονται με το gaussian παράθυρο για να μειωθεί ο θόρυβος.
5. **Αρχικοποίηση διανυσμάτων ροής:** Τα διανύσματα ροής u και v, που αντιπροσωπεύουν τις οριζόντιες και κάθετες συνιστώσες του οπτικού ρεύματος αντίστοιχα, αρχικοποιούνται με μηδενικές τιμές.
6. **Υπολογισμός της οπτικής ροής:** Για κάθε pixel (ανά παράθυρο), κατασκευάζεται το μητρωό A^TA και το διάνυσμα A^Tb. Γίνεται έλεγχος των ιδιοτιμών του A^TA για να μελετηθεί η αντιστρεψιμότητα του καθώς και το είδος των σημείων:
   a) Αν οι ιδιοτιμές είναι μικρότερες από ένα κατώφλι (λ_1<threshold & λ_2<threshold), το σημείο θεωρείται επίπεδο και η οπτική ροή τίθεται σε μηδέν.
   b) Αν οι ιδιοτιμές έχουν μεγάλη διαφορά ($\lambda_1 >> \lambda_2$), το σημείο θεωρείται ακμή και υπολογίζεται η οπτική ροή.
   c) Αν οι ιδιοτιμές έχουν μεγάλη διαφορά ($\lambda_2 >> \lambda_1$), το σημείο θεωρείται ακμή και υπολογίζεται η οπτική ροή.

d) Αν οι ιδιοτιμές είναι μεγάλες και κοντινές ($\lambda_1 < \lambda_2$ ή $\lambda_2 < \lambda_1$ & $\lambda_1 \sim \lambda_2$), το σημείο θεωρείται γωνία και υπολογίζεται η οπτική ροή.

Σε κάθε άλλη περίπτωση, η οπτική ροή τίθεται σε μηδέν.

Για τον υπολογισμό της οπτικής ροής χρησιμοποιούμε Ελάχιστα Τετράγωνα:

$$A\,u = B \Leftrightarrow A^T A\,u = A^T B \Leftrightarrow \boldsymbol{u = (A^T A)^{-1} A^T B}$$

όπου,

$$A^T A = \begin{bmatrix} \sum_W IxIx & \sum_W IxIy \\ \sum_W IxIy & \sum_W IyIy \end{bmatrix}, \quad u = \begin{matrix} u \\ v \end{matrix}, \quad A^T B = \begin{matrix} -\sum_W IxIt \\ -\sum_W IyIt \end{matrix}$$

με την προϋπόθεση ότι $A^T A$ αντιστρέψιμο.

Σημείωση: Έχουμε υποθέσει ότι για κάθε pixel η οπτική ροή (u,v) είναι σταθερή μέσα σε ένα μικρό παράθυρο W.

```
Time taken for optical flow computation: 1.4452 seconds
Time taken for matlabs optical flow computation: 0.0119 seconds
```

Figure 1. Οπτική ροή σε δύο διαδοχικά frame από τις συναρτήσεις opticalFlow_implimentation και opticalFlowLK .

Παρατηρήσεις: Η υλοποίηση για οπτική ροή δίνει λιγότερα διανύσματα συγκριτικά με την συνάρτηση της Matlab, κατεβάζοντας το κατώφλι που θέτουμε για τις ιδιοτιμές παίρνουμε πολλά διανύσματα στο χώρο του δρόμου. Το τελικό κατώφλι που επιλέχθηκε δηλαδή συγκρίθηκε με αρκετά άλλα και δίνει μεταξύ αυτών τα πιο ικανοποιητικά αποτελέσματα.

Οι χρόνοι των δύο αλγορίθμων έχουν μια διαφορά της τάξης του 10^2.

2 Επαναχρησιμοποιήστε τον κώδικα για να υλοποιήσετε έναν ανιχνευτή γωνιών λαμβάνοντας υπόψη το μέγεθος των ιδιοτιμών του πίνακα A^TA.

Η ανίχνευση γωνιών είναι μια σημαντική διαδικασία στην επεξεργασία εικόνας και στην ανάλυση υπολογιστικής όρασης, καθώς οι γωνίες είναι χαρακτηριστικά σημεία που περιέχουν πλούσια πληροφορία για την δομή της εικόνας. Η υλοποίηση βασίζεται στον ανιχνευτή γωνιών του Harris, ο οποίος χρησιμοποιεί τις ιδιοτιμές του πίνακα A^TA για να εντοπίσει γωνίες σε μια εικόνα. Οι γωνίες ανιχνεύονται βάσει του ότι οι ιδιοτιμές του πίνακα αυτού είναι μεγάλες, κάτι που δείχνει μεγάλη αλλαγή στις τιμές των παραγώγων σε όλες τις κατευθύνσεις.

Μπορούμε εύκολα να αναγνωρίσουμε τέτοια σημεία κοιτάζοντας μέσα από ένα μικρό παράθυρο. Η μετατόπιση του παραθύρου προς οποιαδήποτε κατεύθυνση θα πρέπει δίνει μεγάλη αλλαγή στην φωτεινότητα.

Αναμένουμε: $A^TA = \begin{bmatrix} \sum_W IxIx & \sum_W IxIy \\ \sum_W IxIy & \sum_W IyIy \end{bmatrix} = \begin{bmatrix} \sum_W IxIx & 0 \\ 0 & \sum_W IyIy \end{bmatrix} = \begin{bmatrix} \lambda 1 & 0 \\ 0 & \lambda 2 \end{bmatrix}$

Εάν είτε το a είτε το b είναι κοντά στο 0, τότε δεν έχουμε γωνία, οπότε αναζητούμε σημεία όπου και οι δύο τιμές είναι μεγάλες.

Μια καλή μετρική για τον εντοπισμό γωνιών με βάση τις ιδιοτιμές του A^TA είναι η εξής:

$R = \det(A^TA) - \alpha \, \text{trace}(A^TA)^2 = \lambda_1\lambda_2 - \alpha(\lambda_1+\lambda_2)^2$, όπου λ_1,λ2 ιδιοτιμές του A^TA και $\alpha = 0.05$.

Σύμφωνα με αυτή την μετρική, εάν, R>0 έχουμε γωνία.

Υλοποιούμε την συνάρτηση: **[i,j] = cornerDetection(im, windowSize, thres)**

- im: εικόνα
- windowSize: το μέγεθος του παραθύρου που εξετάζουμε
- thres: το κατώφλι πάνω από το οποίο πρέπει να βρίσκεται η ιδιοτιμή
- i,j: διανύσματα που περιέχουν τις συντεταγμένες των γωνιών

Βήματα υλοποίησης:

1. **Μετατροπή των εικόνων σε διπλή ακρίβεια:** Για την εξασφάλιση ακριβών υπολογισμών, οι εικόνες μετατρέπονται σε μορφή διπλής ακρίβειας (double).
2. **Υπολογισμός των παραγώγων των εικόνων:** Οι παράγωγοι των εικόνων ως προς τις συντεταγμένες x και y υπολογίζονται μέσω συνελίξεων με κατάλληλους πυρήνες:
3. **Δημιουργία Gaussian παραθύρου:** Ένα gaussian φίλτρο χρησιμοποιείται για να δώσει μεγαλύτερο βάρος στα κεντρικά σημεία του παραθύρου και να εξομαλύνει τις τιμές των παραγώγων.

4. **Υπολογισμός των γινομένων των παραγώγων:** Τα γινόμενα των παραγώγων υπολογίζονται για κάθε pixel, τα οποία στη συνέχεια φιλτράρονται με το gaussian παράθυρο για να μειωθεί ο θόρυβος.
5. **Υπολογισμός του R(i,j):**
   Για κάθε σημείο της εικόνας, υπολογίζεται ο πίνακας A^TA χρησιμοποιώντας τα αθροίσματα των φιλτραρισμένων γινομένων των παραγώγων. Οι ιδιοτιμές του πίνακα υπολογίζονται και χρησιμοποιούνται για τον υπολογισμό του R.
6. **Κατωφλίωση και Ανίχνευση Τοπικών Μέγιστων:**
   Ο πίνακας R κατωφλιώνεται με βάση το κατώφλι.
   Ανιχνεύονται τα τοπικά μέγιστα του κατωφλιωμένου πίνακα χρησιμοποιώντας τη συνάρτηση imregionalmax.
7. **Επιστροφή των Συντεταγμένων των Γωνιών:** Οι συντεταγμένες των ανιχνευμένων γωνιών επιστρέφονται ως διανύσματα i και j.

```
Time taken for corner detection computation: 0.6648 seconds
Time taken for matlabs corner detection computation: 0.0251 seconds
```

Figure 2. Εντοπισμός γωνιών στην ίδια εικόνα από τις συναρτήσεις cornerDetection και detectHarrisFeatures.

Παρατηρήσεις: Η υλοποίηση για ανίχνευση γωνιών δίνει αποτελέσματα σχεδόν ταυτόσημα με την συνάρτηση του Matlab, οι χρόνοι βέβαια έχουν μια διαφορά της τάξης του 10.

3 Χρησιμοποιήστε τα 1,2 ώστε να βρείτε την οπτική ροή μόνο στις γωνίες (αραιή οπτική ροή).

Το πρόβλημα που εξετάζουμε είναι ο εντοπισμός των γωνιών σε μια εικόνα και η εκτίμηση της οπτικής ροής μόνο στις θέσεις των γωνιών αυτών.

Βήματα υλοποίησης:

1. **Επιλογή δύο frame:** Επιλέγουμε δύο διαδοχικά frame από το video1.avi.
2. **Μετατροπή σε grayscale:** Μετατρέπουμε τις εικόνες σε grayscale για να απλοποιήσουμε και να επιταχύνουμε τους υπολογισμούς.
3. **Ανίχνευση Γωνιών:** Εντοπίζουμε τις γωνίες στην πρώτη εικόνα καλώντας την συνάρτηση cornerDetection.
4. **Υπολογισός Οπτικής Ροής:** Υπολογίζουμε την οπτική ροή μόνο στις γωνίες που έχουν εντοπιστεί από την συνάρτηση cornerDetection.
5. **Οπτικοποίηση:** Προβάλλουμε τις γωνίες και τα διανύσματα οπτικής ροής στην πρώτη εικόνα.

```
Time taken for sparce optical flow computation: 0.0292 seconds
```

Figure 3. Εντοπισμός οπτικής ροής μόνο σε γωνίες.

Παρατηρήσεις: Με την χρήση της αραιής οπτικής ροής έχουμε 479 σημεία (γωνίες) ενώ με την χρήση της πυκνής οπτικής ροής είχαμε σημεία 7942 (γωνίες και ακμές) . Επιπλέον ο χρόνος εκτέλεσης της αραιής οπτικής ροής είναι 0.0292s ενώ της πυκνής οπτικής ροής 1.4452s, διαφορά της τάξης του 10^2.

4 Σταθεροποίηση βίντεο

Στόχος είναι η χρήση της οπτικής ροής που υπολογίζουμε για να εξαλείψουμε τις ανεπιθύμητες κινήσεις. Χρησιμοποιούμε βίντεο χωρίς κινούμενο φόντο στο αρχείο video 1, οποιαδήποτε κίνηση μεταξύ διαδοχικών frames (μπορεί να μοντελοποιηθεί ως μετασχηματισμός affine της θέσης) θεωρείται αστάθεια και πρέπει να εξαλειφθεί.

Υλοποιούμε την συνάρτηση: **ret = videoStabilization1(invideo, outvideo)**

- invideo: βίντεο εισόδου
- outvideo: το σταθεροποιημένο βίντεο
- ret: 0 ή 1 ανάλογα με το αν διαβάστηκε/ γράφτηκε το βίντεο σωστά.

4.1 Μετασχηματισμός ανά pixel λαμβάνοντας υπόψη την οπτική ροή από frame σε frame.

Η διαδικασία σταθεροποίησης του βίντεο εκμεταλλεύεται την οπτική ροή μεταξύ διαδοχικών frame για να εκτιμήσει και να διορθώσει τις ανεπιθύμητες κινήσεις. Η οπτική ροή υπολογίζεται και στη συνέχεια τα frame προσαρμόζονται ανάλογα για να σταθεροποιηθεί το βίντεο. Ο αλγόριθμος μπορεί να διαχωριστεί στα εξής βήματα:

1. **Φόρτωση του βίντεο και αρχικοποίηση μεταβλητών:** Διαβάζουμε το εισερχόμενο βίντεο, προετοιμάζουμε το αρχείο εξόδου και αρχικοποιούμε τις απαραίτητες μεταβλητές.
2. **Επεξεργασία των frame:**
    - Μετατροπή των frame σε γκρι κλίμακα για απλοποίηση των υπολογισμών.
    - Αρχικοποίηση της πρώτης εικόνας και των διανυσμάτων οπτικής ροής.
3. **Υπολογισμός Οπτικής Ροής:**
    - Χρησιμοποιούμε την συνάρτηση opticalFlow_implimentation για υπολογισμό της οπτικής ροής μεταξύ δύο διαδοχικών frame.
    - Υπολογισμός των μεταβολών στις οριζόντιες (u) και κατακόρυφες (v) συνιστώσες της ροής.
4. **Εξομάλυνση της Οπτικής Ροής:** Χρησιμοποιούμε φίλτρο box για να εξομαλύνουμε τα διανύσματα ροής ώστε να αποφύγουμε αιχμηρές μεταβολές.
5. **Παραμόρφωση των frame:**
    - Δημιουργία πλέγματος (grid) για την παραμόρφωση της τρέχουσας εικόνας με βάση τις μεταβολές των διανυσμάτων ροής.
    - Χρήση της συνάρτησης interp2 για την παραμόρφωση της τρέχουσας εικόνας με βάση τα διανύσματα ροής.
6. **Αποθήκευση του σταθεροποιημένου frame:** Αποθήκευση του προσαρμοσμένου καρέ στο αρχείο εξόδου.

Παρατηρήσεις: Η παραπάνω υλοποίηση συγκρίθηκε με παρόμοια όπου για τον υπολογισμό της οπτικής ροής έγινε χρήση της συνάρτησης opticalFlowLK() του Matlab αντί της συνάρτησης opticalFlow_impimentation() και παρατηρήθηκε πως στην περίπτωση αυτή οι παραμορφώσεις των αντικειμένων της σκηνής είναι αρκετά αισθητές συγκριτικά με την αρχική υλοποίηση. Επιπλέον δοκιμάστηκε και η χρήση της αραιής οπτικής ροής, καθώς τα frame προχωρούσαν, το βίντεο χαλούσε αισθητά.

4.2 Μετασχηματισμός με βάση τις γωνίες και με χρήση μοντέλου affine.

1. **Εισαγωγή και προετοιμασία βίντεο:**
    - Γίνεται ανάγνωση του αρχείου εισόδου (invideo) και δημιουργείται ένα νέο αρχείο βίντεο εξόδου (outvideo).
    - Δημιουργείται ένα VideoPlayer για την προβολή των πλαισίων.
2. **Επεξεργασία πλαισίων:**
    - Το πρώτο frame μετατρέπεται σε γκρι κλίμακα και αποθηκεύεται ως movMean και imgB.
    - Ορίζονται παράμετροι όπως το windowSize για τον εντοπισμό γωνιών, το thres για το κατώφλι γωνιών και το maxCorners για τον μέγιστο αριθμό γωνιών.
3. **Βρόχος επεξεργασίας πλαισίων:**
    - Για κάθε νέο frame (imgB), γίνεται σύγκριση με το προηγούμενο frame (imgA).
    - Εντοπίζονται οι γωνίες στα δύο frame χρησιμοποιώντας τη συνάρτηση **cornerDetection**.
    - Οι γωνίες μετατρέπονται σε σημεία (cornerPoints).
    - Γίνονται εξαγωγή χαρακτηριστικών και ταίριασμα των σημείων μεταξύ των δύο frame.
    - Υπολογίζεται ο affine μετασχηματισμός (tformAffine) με χρήση της συνάρτησης **estimateAffine**.
    - Ενημερώνεται ο σωρευτικός μετασχηματισμός (cumulativeTform) και εφαρμόζεται στο τρέχον frame (imgBp).
    - Γίνεται προβολή και αποθήκευση του διορθωμένου frame.
4. **Τέλος επεξεργασίας:**
    - Κανονικοποιούνται οι μέσοι όροι των frame.
    - Ολοκληρώνεται η αποθήκευση του βίντεο εξόδου και η συνάρτηση επιστρέφει 1 για επιτυχία ή 0 για αποτυχία.

Συνάρτηση **tformAffine = estimateAffine(pointsA, pointsB)**

- pointsA, pointsB: σύνολο σημείων μεταξύ των οποίων θα υπολογιστεί ο affine μετασχηματισμός
- tformAffine: affine μετασχηματισμός που αντιστοιχίζει τα σημεία του συνόλου pointsB στα pointsA.

Βήματα υλοποίησης:

1. **Μετατροπή των σημείων στη σωστή μορφή:** Ελέγχεται αν τα σημεία είναι της κλάσης cornerPoints και αν ναι, μετατρέπονται σε έναν πίνακα συντεταγμένων με τη χρήση της ιδιότητας Location.
2. **Αριθμός σημείων:** Υπολογίζεται ο αριθμός των σημείων που περιέχονται στους πίνακες pointsA και pointsB.
3. **Κατασκευή του πίνακα A και του διανύσματος b:**
    - Ο πίνακας A κατασκευάζεται έτσι ώστε κάθε σημείο στο pointsB να συνεισφέρει δύο γραμμές: μία για την οριζόντια συνιστώσα και μία για την κατακόρυφη.
    - Το διάνυσμα b περιέχει τις συντεταγμένες των σημείων στο pointsA.
4. **Επίλυση του προβλήματος ελαχίστων τετραγώνων:** Χρησιμοποιείται η μέθοδος των ελαχίστων τετραγώνων για την επίλυση του συστήματος $A \cdot p = b$, όπου p είναι οι παράμετροι του affine μετασχηματισμού.
5. **Κατασκευή του affine μετασχηματισμού:** Οι παράμετροι του μετασχηματισμού τοποθετούνται σε έναν πίνακα 3x3 για να δημιουργηθεί ο affine μετασχηματισμός.

Παρατηρήσεις: Η παραπάνω υλοποίηση συγκρίθηκε με παρόμοια όπου για τον εντοπισμό των γωνιών έγινε χρήση της συνάρτησης detectFASTFeatures() του Matlab αντί της συνάρτησης cornerDetection() και παρατηρήθηκε πως μέχρι το frame 100 περίπου δίνουν ακριβώς τα ίδια αποτελέσματα, έπειτα υπάρχουν μικρές αποκλίσεις που γίνονται όλο και πιο αισθητές καθώς το βίντεο προχωράει.

5 Προαιρετικό

Για καλύτερη απόσβεση των μικροκινήσεων της κάμερας εφαρμόζουμε βαθυπερατό φιλτράρισμα δηλαδή βγάζουμε ένα κινητό μέσο όρο από N συνεχόμενα πλαίσια και στη συνέχεια να κάνουμε τον μετασχηματισμό. Εφαρμόζουμε το αποτέλεσμα σε βίντεο από drone video2.mp4. Επειδή υπάρχουν κινούμενα αντικείμενα εφαρμόζουμε RANSAC.

Τροποποιήσεις στον κώδικα:

1. **Αντικατάσταση του αρχείου βίντεο με το νέο βίντεο από drone.**
2. **Εισαγωγή βαθυπερατού φιλτραρίσματος χρησιμοποιώντας κινητό μέσο όρο:**
    - Προσθήκη buffer (transformBuffer) για αποθήκευση των τελευταίων N μετασχηματισμών.
    - Υπολογισμός του κινητού μέσου όρου των μετασχηματισμών για τη δημιουργία ενός ομαλοποιημένου μετασχηματισμού.
3. **Χρήση του αλγορίθμου RANSAC για τον υπολογισμό του affine μετασχηματισμού:** Χρήση της συνάρτησης estimateGeometricTransform με την επιλογή ransac για τον υπολογισμό του affine μετασχηματισμού που είναι ανθεκτικός στα κινούμενα αντικείμενα.

Με αυτές τις αλλαγές, το σύστημα σταθεροποίησης βίντεο γίνεται πιο ανθεκτικό στις μικροκινήσεις και τα κινούμενα αντικείμενα, παρέχοντας ένα πιο σταθερό και ομαλό αποτέλεσμα.

6 Κώδικας

6.1 Αλγόριθμος Οπτικής Ροής

```
function [u, v] = opticalFlow_implimentation(im1, im2, windowSize)
    % Convert images to double precision
    im1 = double(im1);
    im2 = double(im2);

    % Compute image gradients
    Ix = conv2(im1, [-1 1; -1 1], 'same'); % gradient in x
    Iy = conv2(im1, [-1 -1; 1 1], 'same'); % gradient in y
    It = im2 - im1;
    %It = conv2(im2, ones(2), 'same') - conv2(im1, ones(2), 'same'); % gradient in t

    % Gaussian weighting window
    gaussianWindow = fspecial('gaussian', [windowSize windowSize], 0.1);

    % Compute products of derivatives at every pixel
    Ix2 = Ix.^2;
    Iy2 = Iy.^2;
    Ixy = Ix.*Iy;
    Ixt = Ix.*It;
```

```
    Iyt = Iy.*It;

    % Apply Gaussian filter to these products
    Ix2 = conv2(Ix2, gaussianWindow, 'same');
    Iy2 = conv2(Iy2, gaussianWindow, 'same');
    Ixy = conv2(Ixy, gaussianWindow, 'same');
    Ixt = conv2(Ixt, gaussianWindow, 'same');
    Iyt = conv2(Iyt, gaussianWindow, 'same');

    % Initialize flow vectors
    u = zeros(size(im1));
    v = zeros(size(im1));

    % Threshold for eigenvalues
    threshold = 35*10^3;
    % Compute optical flow
    for i = 1:size(im1, 1)
        for j = 1:size(im1, 2)

           % Ensure window stays within image boundaries
            minX = max(1, i - floor(windowSize / 2));
            maxX = min(size(im1, 1), i + floor(windowSize / 2));
            minY = max(1, j - floor(windowSize / 2));
            maxY = min(size(im1, 2), j + floor(windowSize / 2));

            %Construct matrix A
            Ix2_block = Ix2(minX:maxX, minY:maxY);
            Iy2_block = Iy2(minX:maxX, minY:maxY);
            Ixy_block = Ixy(minX:maxX, minY:maxY);
            Ixt_block = Ixt(minX:maxX, minY:maxY);
            Iyt_block = Iyt(minX:maxX, minY:maxY);

            ATA = [sum(Ix2_block(:)), sum(Ixy_block(:)); sum(Ixy_block(:)),
sum(Iy2_block(:))];
            ATb = -[sum(Ixt_block(:)); sum(Iyt_block(:))];

            % Compute eigenvalues of ATA
            eigenvalues = eig(ATA);
            lambda1 = eigenvalues(1);
            lambda2 = eigenvalues(2);

            % Check the four cases based on eigenvalues and threshold
            if lambda1 < threshold || lambda2 < threshold %new
                % Case 1: Both eigenvalues are small (flat region)
                u(i, j) = 0;
                v(i, j) = 0;
            elseif lambda1 > 10*lambda2
                % Case 2: lambda1 >> lambda2 (edge)
                flow = ATA\ATb;
                u(i, j) = flow(1);
                v(i, j) = flow(2);
            elseif lambda2 > 10*lambda1
                % Case 3: lambda2 >> lambda1 (edge)
                flow = ATA\ATb;
                u(i, j) = flow(1);
                v(i, j) = flow(2);
            elseif (lambda1 > lambda2 || lambda2 > lambda1) && abs(lambda1 -
lambda2)<10^5
                % Case 4: Both eigenvalues are large and well-conditioned
                % matrix (corner)
                flow = ATA\ATb;
                u(i, j) = flow(1);
                v(i, j) = flow(2);
            else
```

```
                u(i, j) = 0;
                v(i, j) = 0;
            end
        end
    end
end
```

Εκτέλεση

```
clear;clc;close all;
%% Extract Frames
% Read the video file
video = VideoReader('video1.avi');
% Choose two frames
frame1Idx = 10; % frame index 1
frame2Idx = 11; % frame index 2
% Read the two frames
video.CurrentTime = (frame1Idx-1) / video.FrameRate;
frame1 = readFrame(video);
video.CurrentTime = (frame2Idx-1) / video.FrameRate;
frame2 = readFrame(video);
% Convert frames to grayscale
frame1Gray = rgb2gray(frame1);
frame2Gray = rgb2gray(frame2);
%% For Optical Flow
windowSize =7;
% Compute optical flow
tic;
[u, v] = opticalFlow_implimentation(frame1Gray, frame2Gray, windowSize);
elapsedTime = toc;
fprintf('Time taken for optical flow computation: %.4f seconds\n', elapsedTime);
%% Using Matlab's built in
tic;
% Create optical flow object using the built-in function
opticFlow = opticalFlowLK('NoiseThreshold', 0.01);
% Estimate the flow for the first frame
estimateFlow(opticFlow, frame1Gray);
% Estimate the flow for the second frame
flow = estimateFlow(opticFlow, frame2Gray);
% Extract horizontal and vertical flow components
U = flow.Vx;
V = flow.Vy;
elapsedTime = toc;
fprintf('Time taken for matlabs optical flow computation: %.4f seconds\n',
elapsedTime);
%% Plots
% Display the first frame
figure;
subplot(1, 2, 1);
imshow(frame1Gray);
hold on;
% Display the optical flow vectors
quiver(u, v, 'r', 'LineWidth', 2, 'MaxHeadSize', 2);
hold off;
title('Optical Flow (Custom Implementation)');
subplot(1, 2, 2);
imshow(frame1Gray);
hold on;
quiver(U, V, 'r');
hold off;
title('Optical Flow (MATLAB Built-In)');
```

6.2 Αλγόριθμος Ανίχνευσης Ακμών

```
function [i,j] = cornerDetection(im,windowSize,thres)
    % Convert image to double precision
    im = double(im);

    % Compute image gradients
    Ix = conv2(im, [-1 1; -1 1], 'same'); % gradient in x
    Iy = conv2(im, [-1 -1; 1 1], 'same'); % gradient in y

    % Gaussian weighting window
    gaussianWindow = fspecial('gaussian', [windowSize windowSize], 0.1);

    % Compute products of derivatives at every pixel
    Ix2 = Ix.^2;
    Iy2 = Iy.^2;
    Ixy = Ix.*Iy;

    % Apply Gaussian filter to these products
    Ix2 = conv2(Ix2, gaussianWindow, 'same');
    Iy2 = conv2(Iy2, gaussianWindow, 'same');
    Ixy = conv2(Ixy, gaussianWindow, 'same');

    [rows, cols] = size(im);
    R = zeros(rows, cols);
    % Compute optical flow
    for i = 1:rows
        for j = 1:cols

           % Ensure window stays within image boundaries
            minX = max(1, i - floor(windowSize / 2));
            maxX = min(size(im, 1), i + floor(windowSize / 2));
            minY = max(1, j - floor(windowSize / 2));
            maxY = min(size(im, 2), j + floor(windowSize / 2));

            %Construct matrix ATA
            Ix2_block = Ix2(minX:maxX, minY:maxY);
            Iy2_block = Iy2(minX:maxX, minY:maxY);
            Ixy_block = Ixy(minX:maxX, minY:maxY);

            ATA = [sum(Ix2_block(:)), sum(Ixy_block(:)); sum(Ixy_block(:)),
sum(Iy2_block(:))];

            % Compute eigenvalues of ATA
            eigenvalues = eig(ATA);
            lambda1 = eigenvalues(1);
            lambda2 = eigenvalues(2);
            R(i,j) = lambda1 * lambda2 - 0.05 * (lambda1 + lambda2)^2;
        end
    end
    R(R < thres) = 0;
    corners = imregionalmax(R);
    % Find the coordinates of the corners
    [i, j] = find(corners);
end
```

Εκτέλεση

```
clear;clc;close all;
%% Extract Frames
% Read the video file
video = VideoReader('video1.avi');
% Choose two frames
```

```
frame1Idx = 10;
% Read the two frames
video.CurrentTime = (frame1Idx-1) / video.FrameRate;
frame1 = readFrame(video);
% Convert frame to grayscale
frame1Gray = rgb2gray(frame1);
%% For Corner
windowSize = 7;
thres = 9*10^8;
tic;
[i, j] = cornerDetection(frame1Gray,windowSize,thres);
elapsedTime = toc;
fprintf('Time taken for corner detection computation: %.4f seconds\n', elapsedTime);
%% Using Matlab's built in
tic;
corners = detectHarrisFeatures(frame1Gray, 'MinQuality', 0.0008, 'FilterSize',
windowSize);
elapsedTime = toc;
fprintf('Time taken for matlabs corner detection computation: %.4f seconds\n',
elapsedTime);
%% Plots
% Plot the original image
figure;subplot(1, 2, 1);imshow(frame1Gray, []);hold on;
% Plot the detected corners
plot(j, i, 'r+', 'MarkerSize', 5, 'LineWidth', 1);title('Detected Corners (Custom
Implementation)');hold off;
% Plot the original image
subplot(1, 2, 2); imshow(frame1Gray, []);hold on;
% Highlight the detected corners
plot(corners.Location(:,1), corners.Location(:,2), 'r+', 'MarkerSize', 5, 'LineWidth',
1);
title('Detected Corners (MATLAB Built-In)');hold off;
```

6.3 Αλγόριθμος Αραιής Οπτικής Ροής

```
clear;clc;close all;
%% Extract Frames
% Read the video file
video = VideoReader('video1.avi');
% Choose two frames
frame1Idx = 10; % frame index 1
frame2Idx = 11; % frame index 2
% Read the two frames
video.CurrentTime = (frame1Idx-1) / video.FrameRate;
frame1 = readFrame(video);
video.CurrentTime = (frame2Idx-1) / video.FrameRate;
frame2 = readFrame(video);
% Convert frames to grayscale
frame1Gray = rgb2gray(frame1);
frame2Gray = rgb2gray(frame2);
%% Corner Detection
windowSize = 7;
thres = 9*10^8;
[i, j] = cornerDetection(frame1Gray,windowSize,thres);
%% Optical Flow at returnd corners
% Initialize the optical flow vectors
tic;
u = zeros(size(frame1Gray));
v = zeros(size(frame1Gray));

% Convert images to double precision
```

```
im1 = double(frame1Gray);
im2 = double(frame2Gray);

% Compute image gradients
Ix = conv2(im1, [-1 1; -1 1], 'same'); % gradient in x
Iy = conv2(im1, [-1 -1; 1 1], 'same'); % gradient in y
It = im2 - im1;

% Gaussian weighting window
gaussianWindow = fspecial('gaussian', [windowSize windowSize], 0.1);

% Compute products of derivatives at every pixel
Ix2 = Ix.^2;
Iy2 = Iy.^2;
Ixy = Ix.*Iy;
Ixt = Ix.*It;
Iyt = Iy.*It;

% Apply Gaussian filter to these products
Ix2 = conv2(Ix2, gaussianWindow, 'same');
Iy2 = conv2(Iy2, gaussianWindow, 'same');
Ixy = conv2(Ixy, gaussianWindow, 'same');
Ixt = conv2(Ixt, gaussianWindow, 'same');
Iyt = conv2(Iyt, gaussianWindow, 'same');

% Threshold for eigenvalues
threshold = 35 * 10^3;

% Compute optical flow only at the detected corners
for k = 1:length(i)
    x = i(k);
    y = j(k);

    % Ensure window stays within image boundaries
    minX = max(1, x - floor(windowSize / 2));
    maxX = min(size(im1, 1), x + floor(windowSize / 2));
    minY = max(1, y - floor(windowSize / 2));
    maxY = min(size(im1, 2), y + floor(windowSize / 2));

    % Construct matrix A
    Ix2_block = Ix2(minX:maxX, minY:maxY);
    Iy2_block = Iy2(minX:maxX, minY:maxY);
    Ixy_block = Ixy(minX:maxX, minY:maxY);
    Ixt_block = Ixt(minX:maxX, minY:maxY);
    Iyt_block = Iyt(minX:maxX, minY:maxY);

    ATA = [sum(Ix2_block(:)), sum(Ixy_block(:)); sum(Ixy_block(:)),
sum(Iy2_block(:))];
    ATb = -[sum(Ixt_block(:)); sum(Iyt_block(:))];

    flow = ATA \ ATb;
    u(x, y) = flow(1);
    v(x, y) = flow(2);
end
elapsedTime = toc;
fprintf('Time taken for sparce optical flow computation: %.4f seconds\n',
elapsedTime);
%% Plot
figure;imshow(frame1Gray);hold on;
% Display the corners
plot(j, i, 'ro', 'MarkerSize', 3, 'LineWidth', 0.5);
% Display the optical flow vectors
quiver(j, i, u(i + (j - 1) * size(im1, 1)), v(i + (j - 1) * size(im1, 1)), 'r',
'LineWidth', 0.7);
```

```
hold off;title('Optical Flow at Detected Corners');
```

6.4 Σταθεροποίηση βίντεο με οπτική ροή

```
function ret = videoStabilization1(invideo, outvideo)
    try
        % Load video file
        InputVid = vision.VideoFileReader(invideo);
        output_object = VideoWriter(outvideo);
        open(output_object);

        % Initialize first frame
        fprintf('Start processing frames \n')
        I1 = rgb2gray(step(InputVid));
        u = zeros(size(I1));
        v = zeros(size(I1));
        writeVideo(output_object, I1);

        % Parameters
        windowSize = 7;
        numOfFrames = 200;

        for k = 2:numOfFrames
            % Protection against too short video
            if isDone(InputVid)
                break
            end
            I2 = rgb2gray(step(InputVid));

            % Compute flow
            [du, dv] = opticalFlow_implimentation(I1, I2, windowSize);
            %{
            opticFlow = opticalFlowLK('NoiseThreshold', 0.01);
            estimateFlow(opticFlow, I1);
            flow = estimateFlow(opticFlow, I2);
            du = flow.Vx;
            dv = flow.Vy;
            %}
            u = u + du;
            v = v + dv;

            % Perform averaging to smooth the result
            u = imboxfilt(u, 5);
            v = imboxfilt(v, 5);

            % Generate grid
            [X, Y] = meshgrid(1:size(I2, 2), 1:size(I2, 1));

            % Warp the current frame (distort the current frame)
            I2_warp = interp2(X, Y, I2, X + u, Y + v);
            I2_warp(isnan(I2_warp)) = I2(isnan(I2_warp));

            % Save to file
            writeVideo(output_object, I2_warp);
            I1 = I2;

            fprintf('finished frame number = %d/%d\n', k, numOfFrames);
        end

        % Close the output video object
        close(output_object);
        release(InputVid);
```

```
        % Return 1 indicating success
        ret = 1;
    catch
        % Return 0 indicating failure
        ret = 0;
    end
end
```

Εκτέλεση

```
clear; clc; close all;
%% videoStabilization1
ret = videoStabilization1('video1.avi', 'StabilizedVideo_1a_Custom.avi');
%%
% Load the videos
video1 = VideoReader('video1.avi');
video2 = VideoReader('StabilizedVideo_1a_Matlab.avi');
video3 = VideoReader('StabilizedVideo_1a_Custom.avi');

% Create a new figure for video playback
figure;

while hasFrame(video1) && hasFrame(video2) && hasFrame(video3)
    % Read the next frame from each video
    frame1 = readFrame(video1);
    frame2 = readFrame(video2);
    frame3 = readFrame(video3);

    % Display the first video
    subplot(1, 3, 1);
    imshow(frame1);
    title('Original Video');

    % Display the second video
    subplot(1, 3, 2);
    imshow(frame2);
    title('Stabilized Video (Matlabs optical flow)');

    % Display the third video
    subplot(1, 3, 3);
    imshow(frame3);
    title('Stabilized Video (Custom)');

    % Pause to allow for playback at the correct frame rate
    pause(1/video1.FrameRate);
end
```

6.5 Σταθεροποίηση βίντεο με μετασχηματισμό affine

```
function ret = videoStabilization2(invideo, outvideo)
    try
        % Read the input video
        hVideoSrc = VideoReader(invideo);
        outputVideo = VideoWriter(outvideo);
        open(outputVideo);

        % Initialize the video player
        hVPlayer = vision.VideoPlayer;

        % Process all frames in the video
```

```
movMean = im2gray(im2single(readFrame(hVideoSrc)));
imgB = movMean;
imgBp = imgB;
correctedMean = imgBp;
ii = 2;
cumulativeTform = eye(3);
windowSize = 7;  % Window size for custom corner detection
thres = 9 * 10^8;  % Threshold for corner detection
maxCorners = 10^4;  % Maximum number of corners to detect
%%
while hasFrame(hVideoSrc)
    % Read in new frame
    imgA = imgB;
    imgAp = imgBp;
    imgB = im2gray(im2single(readFrame(hVideoSrc)));
    movMean = movMean + imgB;

    % Generate prospective points
    %pointsA = detectFASTFeatures(imgA, 'MinContrast', 0.1);
    %pointsB = detectFASTFeatures(imgB, 'MinContrast', 0.1);

    % Custom corner detection
    [iA, jA] = cornerDetection(imgA, windowSize, thres);
    [iB, jB] = cornerDetection(imgB, windowSize, thres);

    % Convert corner coordinates to point objects
    pointsA = cornerPoints([jA, iA]);
    pointsB = cornerPoints([jB, iB]);

    % Randomly select maxCorners points
    if length(pointsA) > maxCorners
        indicesA = randperm(length(pointsA), maxCorners);
        pointsA = pointsA(indicesA);
    end

    if length(pointsB) > maxCorners
        indicesB = randperm(length(pointsB), maxCorners);
        pointsB = pointsB(indicesB);
    end

    % Extract features for the corners
    [featuresA, pointsA] = extractFeatures(imgA, pointsA);
    [featuresB, pointsB] = extractFeatures(imgB, pointsB);

    % Match features which were computed from the current and the previous
images
    indexPairs = matchFeatures(featuresA, featuresB);
    pointsA = pointsA(indexPairs(:, 1), :);
    pointsB = pointsB(indexPairs(:, 2), :);

    % Estimate affine transformation manually
    tformAffine = estimateAffine(pointsA.Location, pointsB.Location)';

    % Convert a 3x3 affine transform to a scale-rotation-translation transform
    % Extract rotation and translation submatrices
    R = tformAffine(1:2, 1:2);
    % Translation remains the same:
    translation = tformAffine(3, 1:2);
    % Compute theta from mean of stable arctangents
    theta = mean([atan2(R(2), R(1)) atan2(-R(3), R(4))]);
    % Compute scale from mean of two stable mean calculations
    scale = mean(R([1 4])/cos(theta));
    % Reconstitute new s-R-T (scale-rotation-translation) transform
    R = [cos(theta) -sin(theta); sin(theta) cos(theta)];
```

```
            sRtTform = [[scale*R; translation], [0 0 1]'];

            % Update cumulative transformation
            cumulativeTform = cumulativeTform * sRtTform;
            % Create a structure for imwarp
            tformStruct = maketform('affine', cumulativeTform);
            % Warp the current frame
            imgBp = imtransform(imgB, tformStruct, 'XData', [1 size(imgB, 2)],
'YData', [1 size(imgB, 1)]);

            % Write the frame to the output video
            writeVideo(outputVideo, imgBp);

            % Display as color composite with last corrected frame
            step(hVPlayer, imfuse(imgAp, imgBp, 'ColorChannels', 'red-cyan'));
            correctedMean = correctedMean + imgBp;

            disp(['Processing frame ', num2str(ii)]);

            ii = ii + 1;
        end
        %%
        % Normalize the means
        correctedMean = correctedMean / (ii - 2);
        movMean = movMean / (ii - 2);

        % Release the video player and close the output video file
        release(hVPlayer);
        close(outputVideo);

        % Return 1 indicating success
        ret = 1;
    catch
        % Return 0 indicating failure
        ret = 0;
    end
end
```

Συνάρτηση estimateAffine

```
function tformAffine = estimateAffine(pointsA, pointsB)
    % Ensure points are in correct format
    if isa(pointsA, 'cornerPoints')
        pointsA = pointsA.Location;
    end
    if isa(pointsB, 'cornerPoints')
        pointsB = pointsB.Location;
    end

    % Number of points
    n = size(pointsA, 1);

    % Construct the matrix A and vector b
    A = zeros(2*n, 6);
    b = zeros(2*n, 1);

    for i = 1:n
        A(2*i-1, :) = [pointsB(i, 1), pointsB(i, 2), 1, 0, 0, 0];
        A(2*i, :) = [0, 0, 0, pointsB(i, 1), pointsB(i, 2), 1];
        b(2*i-1) = pointsA(i, 1);
        b(2*i) = pointsA(i, 2);
    end
```

```
    % Solve the least-squares problem
    p = A \ b;

    % Construct the affine transformation matrix
    tformAffine = [
        p(1), p(2), p(3);
        p(4), p(5), p(6);
        0,    0,    1
    ];
end
```

Εκτέλεση

```
clear; clc; close all;
%% videoStabilization2
ret = videoStabilization2('video1.avi', 'StabilizedVideo_1b_Custom.avi');
%%
% Load the videos
video1 = VideoReader('video1.avi');
video2 = VideoReader('StabilizedVideo_1b_Matlab.avi');
video3 = VideoReader('StabilizedVideo_1b_Custom.avi');

% Create a new figure for video playback
figure;

while hasFrame(video1) && hasFrame(video2) && hasFrame(video3)
    % Read the next frame from each video
    frame1 = readFrame(video1);
    frame2 = readFrame(video2);
    frame3 = readFrame(video3);

    % Display the first video
    subplot(1, 3, 1);
    imshow(frame1);
    title('Original Video');

    % Display the second video
    subplot(1, 3, 2);
    imshow(frame2);
    title('Stabilized Video (Matlabs corner detection)');

    % Display the third video
    subplot(1, 3, 3);
    imshow(frame3);
    title('Stabilized Video (Custom)');

    % Pause to allow for playback at the correct frame rate
    pause(1/video1.FrameRate);
end
```

6.6 Σταθεροποίηση βίντεο από drone

```
clear; clc; close all;
%%
filename = 'video2.mp4';
hVideoSrc = VideoReader(filename);
outputVideo = VideoWriter('StabilizedVideo_Drone_Custom.avi');
open(outputVideo);
hVPlayer = vision.VideoPlayer;

% Process all frames in the video
```

```
movMean = im2gray(im2single(readFrame(hVideoSrc)));
imgB = movMean;
imgBp = imgB;
correctedMean = imgBp;
ii = 2;
cumulativeTform = eye(3);
windowSize = 7;  % Window size for custom corner detection
thres = 9 * 10^8;  % Threshold for corner detection
maxCorners = 3.5 * 10^4;  % Maximum number of corners to detect

% Parameters for moving average
N = 5; % Number of frames for moving average
transformBuffer = repmat(eye(3), 1, 1, N);
bufferIndex = 1;

%%
while hasFrame(hVideoSrc)  %&& ii < 700
    % Read in new frame
    imgA = imgB;
    imgAp = imgBp;
    imgB = im2gray(im2single(readFrame(hVideoSrc)));
    movMean = movMean + imgB;

    %Corner detection
    pointsA = detectFASTFeatures(imgA, 'MinContrast', 0.1);
    pointsB = detectFASTFeatures(imgB, 'MinContrast', 0.1);
    %{
    % Custom corner detection
    [iA, jA] = cornerDetection(imgA, windowSize, thres);
    [iB, jB] = cornerDetection(imgB, windowSize, thres);

    % Convert corner coordinates to point objects
    pointsA = cornerPoints([jA, iA]);
    pointsB = cornerPoints([jB, iB]);

    % Randomly select maxCorners points
    if length(pointsA) > maxCorners
        indicesA = randperm(length(pointsA), maxCorners);
        pointsA = pointsA(indicesA);
    end

    if length(pointsB) > maxCorners
        indicesB = randperm(length(pointsB), maxCorners);
        pointsB = pointsB(indicesB);
    end
    %}
    % Extract features for the corners
    [featuresA, pointsA] = extractFeatures(imgA, pointsA);
    [featuresB, pointsB] = extractFeatures(imgB, pointsB);

    % Match features which were computed from the current and the previous images
    indexPairs = matchFeatures(featuresA, featuresB);
    pointsA = pointsA(indexPairs(:, 1), :);
    pointsB = pointsB(indexPairs(:, 2), :);

    % Estimate affine transformation using RANSAC
    [tform, inlierPointsA, inlierPointsB] = estimateGeometricTransform(pointsB,
pointsA, 'affine', 'MaxDistance', 4, 'Confidence', 99.99, 'MaxNumTrials', 2000);
    tformAffine = tform.T;

    % Convert a 3x3 affine transform to a scale-rotation-translation transform
    % Extract rotation and translation submatrices
    R = tformAffine(1:2, 1:2);
    translation = tformAffine(3, 1:2);
```

```
    theta = mean([atan2(R(2), R(1)) atan2(-R(3), R(4))]);
    scale = mean(R([1 4]) / cos(theta));
    R = [cos(theta) -sin(theta); sin(theta) cos(theta)];
    sRtTform = [[scale*R; translation], [0 0 1]'];

    % Update the transformation buffer with the new transformation
    transformBuffer(:, :, bufferIndex) = sRtTform;
    bufferIndex = mod(bufferIndex, N) + 1;

    % Calculate the moving average of the transformations
    cumulativeTform = mean(transformBuffer, 3);

    % Create a structure for imwarp
    tformStruct = maketform('affine', cumulativeTform);
    % Warp the current frame
    imgBp = imtransform(imgB, tformStruct, 'XData', [1 size(imgB, 2)], 'YData', [1
size(imgB, 1)]);

    % Write the frame to the output video
    writeVideo(outputVideo, imgBp);

    % Display as color composite with last corrected frame
    step(hVPlayer, imfuse(imgAp, imgBp, 'ColorChannels', 'red-cyan'));
    correctedMean = correctedMean + imgBp;

    ii = ii + 1;
end
%%
correctedMean = correctedMean / (ii - 2);
movMean = movMean / (ii - 2);
release(hVPlayer);
close(outputVideo);
```

Εκτέλεση

```
clear; clc; close all;
%% videoStabilization2
%ret = videoStabilization2('video1.avi', 'StabilizedVideo_1b_Custom.avi');
%%
% Load the videos
video1 = VideoReader('video2.mp4');
video2 = VideoReader('StabilizedVideo_Drone_Custom.avi');

% Create a new figure for video playback
figure;

while hasFrame(video1) && hasFrame(video2)
    % Read the next frame from each video
    frame1 = readFrame(video1);
    frame2 = readFrame(video2);

    % Display the first video
    subplot(1, 2, 1);
    imshow(frame1);
    title('Original Video');

    % Display the second video
    subplot(1, 2, 2);
    imshow(frame2);
    title('Stabilized Video');

    % Pause to allow for playback at the correct frame rate
    pause(1/video1.FrameRate);
end
```