



**ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ**  

---

**UNIVERSITY OF PIRAEUS**

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ  
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ  
ΕΠΙΚΟΙΝΩΝΙΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

**ΤΕΛΙΚΗ ΑΠΑΛΛΑΚΤΙΚΗ ΕΡΓΑΣΙΑ ΜΑΘΗΜΑΤΟΣ**  
**ΣΥΣΤΗΜΑΤΑ ΠΟΛΥΜΕΣΩΝ**

ΓΚΟΛΕΜΙ ΚΡΙΣΤΙΑΝ, Π18029

ΑΛΕΞΑΝΔΡΟΣ ΓΚΙΝΕΤΣΙ, Π18028

**Θέμα 1 (4.5 βαθμοί):** Έστω video της επιλογής σας διάρκειας 5 s – 15 s. Υποθέστε ότι το Frame 1 είναι πάντα I frame και ότι τα επόμενα πλαίσια είναι P frames.

i) Κάθε πλαίσιο P προβλέπεται χωρίς αντιστάθμιση κίνησης από το προηγούμενο πλαίσιο. Υπολογίστε και απεικονίστε την ακολουθία εικόνων σφάλματος και κωδικοποιήστε την χωρίς απώλειες. Υλοποιήστε τον κωδικοποιητή/αποκωδικοποιητή.

ii) Υλοποιήστε την τεχνική αντιστάθμισης κίνησης για την συμπίεση της ακολουθίας πλαισίων χρησιμοποιώντας αντιστάθμιση κίνησης σε macroblocks 16x16, ακτίνα αναζήτησης  $k=16$  και τεχνική σύγκρισης macroblocks της επιλογής σας. Αν θέλετε, μπορείτε να επιταχύνετε τη διαδικασία υλοποιώντας λογαριθμική ή ιεραρχική αναζήτηση. Υπολογίστε και απεικονίστε την ακολουθία εικόνων πρόβλεψης και εικόνων σφαλμάτων. Υλοποιήστε τον κωδικοποιητή/ αποκωδικοποιητή.

Για την επίλυση του θέματος 1 έχουν δημιουργηθεί τα εξής source code αρχεία σε python: frames.py, huffman.py, codecs\_huffman.py, motion\_compensation.py, codecs\_motionCompensation.py ενώ για το 1<sup>ο</sup> ερώτημα συγκεκριμένα, χρησιμοποιούνται τα αρχεία frames.py, huffman.py, codecs\_huffman.py τα οποία θα αναλύσουμε το κάθε ένα ξεχωριστά παρακάτω.

### **frames.py**

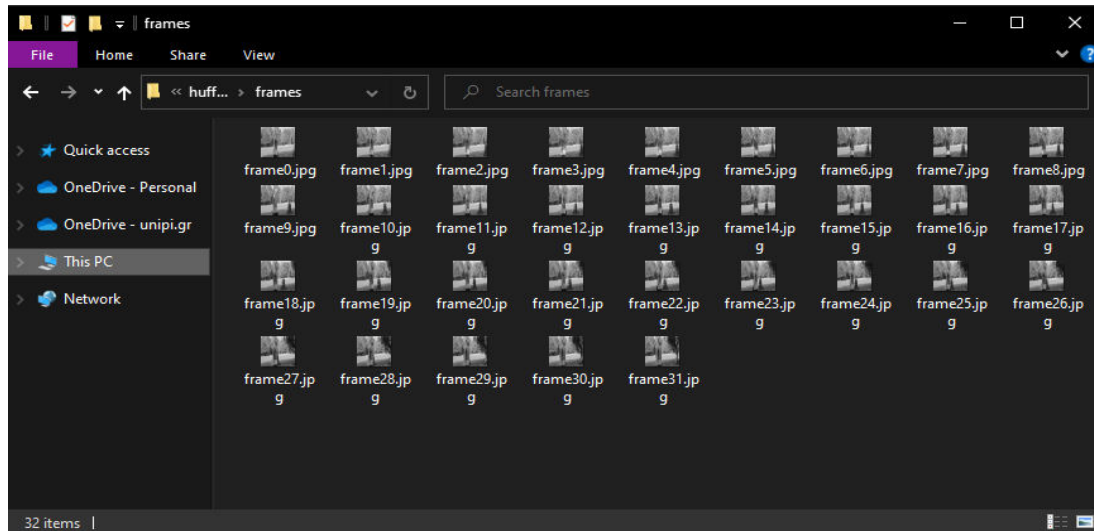
Για την επίλυση του ερωτήματος έχει χρησιμοποιηθεί βίντεο διάρκειας 15s το οποίο απεικονίζει έναν περαστικό σε ένα πάρκο, του οποίου το 1<sup>ο</sup> frame φαίνεται στην παρακάτω εικόνα.



Μια από τις λειτουργίες που υλοποιεί το αρχείο frames.py, λοιπόν, είναι να “κόβει” το δοθέν βίντεο σε frames που ορίζουμε εμείς. Αυτό, γίνεται μέσω της συνάρτησης cut\_frames(), η οποία χρησιμοποιώντας την έτοιμη συνάρτηση VideoCapture() της βιβλιοθήκης OpenCV, διαβάζει το βίντεο που της δίνουμε ως όρισμα και αποθηκεύει διαδοχικά τα frames στο path huffman\_encoding/frames που βρίσκεται στον ίδιο φάκελο με τα αρχεία πηγαίου κώδικα. Για τη δημιουργία των παραπάνω φακέλων γίνεται έλεγχος μέσω του κώδικα και σε περίπτωση που δεν υπάρχουν ήδη, δημιουργούνται αυτόματα. Για μεγαλύτερη ευκολία κατά την υλοποίηση του προγράμματος και μείωση των αρχείων που αποθηκεύονται έχουμε επιλέξει να αποθηκεύουμε 1 ανά 10 frames του βίντεο. Έτσι λοιπόν, τα συνολικά frames που δημιουργούνται είναι 32. Κατά τη δημιουργία και αποθήκευση του κάθε frame, εμφανίζεται μήνυμα στο terminal όπως φαίνεται στην παρακάτω εικόνα ώστε να υπάρχει ενημέρωση προς τον χρήστη για την πορεία της διαδικασίας.

```
PS C:\Users\golem\Desktop\test 1> python -u "c:\Users\golem\Desktop\test 1\frames.py"
Creating...huffman_encoding/frames/frame0.jpg
Creating...huffman_encoding/frames/frame1.jpg
Creating...huffman_encoding/frames/frame2.jpg
Creating...huffman_encoding/frames/frame3.jpg
Creating...huffman_encoding/frames/frame4.jpg
Creating...huffman_encoding/frames/frame5.jpg
Creating...huffman_encoding/frames/frame6.jpg
Creating...huffman_encoding/frames/frame7.jpg
Creating...huffman_encoding/frames/frame8.jpg
Creating...huffman_encoding/frames/frame9.jpg
Creating...huffman_encoding/frames/frame10.jpg
Creating...huffman_encoding/frames/frame11.jpg
Creating...huffman_encoding/frames/frame12.jpg
Creating...huffman_encoding/frames/frame13.jpg
Creating...huffman_encoding/frames/frame14.jpg
Creating...huffman_encoding/frames/frame15.jpg
Creating...huffman_encoding/frames/frame16.jpg
Creating...huffman_encoding/frames/frame17.jpg
Creating...huffman_encoding/frames/frame18.jpg
Creating...huffman_encoding/frames/frame19.jpg
Creating...huffman_encoding/frames/frame20.jpg
Creating...huffman_encoding/frames/frame21.jpg
Creating...huffman_encoding/frames/frame22.jpg
Creating...huffman_encoding/frames/frame23.jpg
Creating...huffman_encoding/frames/frame24.jpg
Creating...huffman_encoding/frames/frame25.jpg
Creating...huffman_encoding/frames/frame26.jpg
Creating...huffman_encoding/frames/frame27.jpg
Creating...huffman_encoding/frames/frame28.jpg
Creating...huffman_encoding/frames/frame29.jpg
Creating...huffman_encoding/frames/frame30.jpg
Creating...huffman_encoding/frames/frame31.jpg
```

Εκτελώντας, λοιπόν, τη συνάρτηση `cut_frames()` του αρχείου `frames.py`, δημιουργούνται οι φάκελοι `huffman_encoding/frames` που περιέχουν τα frames του βίντεο, όπως φαίνεται από την παρακάτω εικόνα.



Αφού χωρίσαμε το βίντεο σε frames, σκοπός μας είναι να υπολογίσουμε και να απεικονίσουμε την ακολουθία εικόνων σφάλματος και αργότερα να την κωδικοποιήσουμε χωρίς απώλειες. Για τον υπολογισμό των εικόνων σφάλματος θα πρέπει αρχικά να φτιάξουμε την ακολουθία με τα προβλεπόμενα frames. Για την επίλυση του ερωτήματος η πρόβλεψη που κάνουμε είναι η απλούστερη, δηλαδή προβλέπουμε συνέχεια πώς το επόμενο frame θα είναι το ίδιο με το προηγούμενο, γεγονός από το οποίο προκύπτει το σφάλμα πρόβλεψης.

Για τη δημιουργία της ακολουθίας των προβλεπόμενων frames έχουμε υλοποιήσει τη συνάρτηση `predict_frames()`, η οποία με βάση τα frames του βίντεο που χωρίσαμε και αποθηκεύσαμε προηγουμένως, δημιουργεί το path `huffman_encoding/predicted_frames` στον οποίο αποθηκεύει την ακολουθία των προβλεπόμενων frames. Το πρώτο προβλεπόμενο frame είναι προφανώς το ίδιο με το 1<sup>ο</sup> πραγματικό frame καθώς δεν υπάρχει προηγούμενο για να γίνει πρόβλεψη, ενώ τα επόμενα προκύπτουν από το παρακάτω κομμάτι κώδικα το οποίο ορίζει όπως αναφέραμε παραπάνω ότι  $\text{predicted\_frame}_{n+1} = \text{actual\_frame}_n$ :

```
# every predicted frame is the same as the previous actual frame  
for i in range(1, len(frames)):  
    predicted_frames.insert(i, frames[i-1])
```

Τελικά, εκτελώντας τη συνάρτηση `predict_frames()`, δημιουργείται ο υποφάκελος `predicted_frames` του φακέλου `huffman_encoding`, στον οποίο έχουν αποθηκευτεί τα προβλεπόμενα frames για λόγους οπτικοποίησης. Κατά τη δημιουργία και αποθήκευση του κάθε προβλεπόμενου frame, εμφανίζεται ανάλογο μήνυμα με την περίπτωση που “κόβαμε” το βίντεο σε frames, για την ενημέρωση του χρήστη, απόσπασμα του οποίου φαίνεται στην παρακάτω εικόνα:

```
Creating...huffman_encoding/predicted_frames/p_frame0.jpg  
Creating...huffman_encoding/predicted_frames/p_frame1.jpg  
Creating...huffman_encoding/predicted_frames/p_frame2.jpg  
Creating...huffman_encoding/predicted_frames/p_frame3.jpg  
Creating...huffman_encoding/predicted_frames/p_frame4.jpg  
Creating...huffman_encoding/predicted_frames/p_frame5.jpg  
Creating...huffman_encoding/predicted_frames/p_frame6.jpg  
Creating...huffman_encoding/predicted_frames/p_frame7.jpg  
Creating...huffman_encoding/predicted_frames/p_frame8.jpg  
Creating...huffman_encoding/predicted_frames/p_frame9.jpg  
Creating...huffman_encoding/predicted_frames/p_frame10.jpg
```

Αφού δημιουργήθηκε και η ακολουθία με τα προβλεπόμενα frames, είναι ώρα να υπολογιστούν τα error frames έτσι ώστε να επιλυθεί το πρόβλημα του ερωτήματος. Για τον σκοπό αυτό έχει υλοποιηθεί η συνάρτηση `prediction_error()`, η οποία παίρνει ως όρισμα ένα actual και ένα predicted frame κάθε φορά μέσω των frames που αποθηκεύσαμε προηγουμένως. Η συγκεκριμένη συνάρτηση, υπολογίζει το σφάλμα πρόβλεψης βρίσκοντας τη διαφορά `actual_frame - predicted_frame` την οποία αποθηκεύει σε μια λίστα `error_frames` τα οποία αργότερα μέσω της έτοιμης συνάρτησης `cv2.imwrite()` αποθηκεύονται στον φάκελο `error_frames`.

Υπολογισμός του σφάλματος πρόβλεψης και αποθήκευσε αρχικά του error frame σε μορφή μήτρας στη λίστα error\_frames[].

```
for i in range(len(frames)):

    # error = actual_frame(n) - predicted_frame(n)
    prediction_error = np.subtract(frames[i], predicted_frames[i])
    error_frames.insert(i, prediction_error)
```

Σε κάθε μια από τις παραπάνω συναρτήσεις, τα frames είναι αποθηκευμένα σε μορφή μήτρας η οποία περιέχει τα pixel values της κάθε εικόνας. Τα pixel values κυμαίνονται από 0-255 καθώς τα frames είναι σε grayscale μορφή. Η αποθήκευση των frames σε μορφή μήτρας επιτυγχάνεται μέσω της συνάρτησης cv2.imread(), ενώ για λόγους οπτικοποίησης του αποτελέσματος, μέσω της cv2.imwrite(), τα frames αποθηκεύονται σε αντίστοιχους φακέλους. Η παραπάνω διαδικασία φαίνεται ενδεικτικά στις παρακάτω δύο εικόνες που είναι κομμάτια του κώδικα:

```
# array to store the frames
frames = []
for frame in filenames:
    temp = cv2.imread(frame, cv2.IMREAD_GRAYSCALE)    # read the frames as grayscale
    frames.append(temp)
```

```
# save the predicted frames in a separate folder
currentFrame = 0
while (currentFrame < len(predicted_frames)):

    name = 'huffman_encoding/predicted_frames/p_frame' + str(currentFrame) + '.jpg'
    predicted = predicted_frames[currentFrame]

    print('Creating...' + name)
    cv2.imwrite(name, predicted)
    currentFrame += 1
```

Μετά την εκτέλεση της `prediction_error()`, λοιπόν δημιουργείται ο φάκελος `error_frames` ο οποίος περιέχει τα `frames` που απεικονίζουν το σφάλμα πρόβλεψης. Ενδεικτικά, παρακάτω φαίνεται το `error_frame1.jpg` που αναπαριστά το σφάλμα πρόβλεψης μεταξύ στο `actual_frame1` και `predicted_frame1`. Τα κομμάτια χρωματισμένα με λευκό χρώμα είναι κομμάτια τα οποία δεν έχουν αποτυπωθεί από την εικόνα πρόβλεψης, ενώ τα μαύρα το αντίθετο. Είναι εμφανές λοιπόν, πώς το γεγονός ότι επιλέξαμε να αποθηκεύσουμε ένα από τα 10 `frames`, οδηγεί σε μεγάλο σφάλμα πρόβλεψης.



Μετά και από τη δημιουργία και την απεικόνιση των `frames` που αναπαριστούν τις εικόνες σφάλματος, μπορούμε να προχωρήσουμε στην κωδικοποίηση των εικόνων αυτών χωρίς απώλειες. Για τον σκοπό αυτό χρησιμοποιούμε τον αλγόριθμο Huffman. Για την κωδικοποίηση Huffman έχουν δημιουργηθεί τα αρχεία `huffman.py` και `codecs_huffman.py`. Το πρώτο περιέχει την κλάση `HuffmanCoding` στην οποία ορίζονται όσα χρειάζονται για να υλοποιηθεί η κωδικοποίηση, ενώ το δεύτερο αρχείο περιέχει συναρτήσεις οι οποίες αναπαριστούν τον κωδικοποιητή και τον αποκωδικοποιητή και χρησιμοποιώντας τις συναρτήσεις της κλάσης



HuffmanCoding, υλοποιούν την κωδικοποίηση των εικόνων σφαλμάτων. Στη συνέχεια αναλύουμε κάθε αρχείο και τις λειτουργίες του ξεχωριστά.

## huffman.py

Πριν την ανάλυση του συγκεκριμένου αρχείου είναι σημαντικό να αναφέρουμε τη λειτουργία της συνάρτησης `jpg_to_txt()` που βρίσκεται στο αρχείο `codecs_huffman.py`. Η συγκεκριμένη συνάρτηση για κάθε εικόνα σφάλματος που προηγουμένως αποθηκεύσαμε, δημιουργεί ένα `.txt` αρχείο, το οποίο περιέχει τα `pixel values` της μήτρας που αναπαριστά την grayscale εικόνα. Ετσι, για παράδειγμα για την 2<sup>η</sup> εικόνα σφάλματος που φαίνεται στην προηγούμενη σελίδα, δημιουργείται ένα αρχείο `.txt` που περιέχει τα `pixel values` της όπως φαίνεται παρακάτω σε ένα πολύ μικρό απόσπασμα του αρχείου.

[illegible]

Η κλάση HuffmanCoding, περιέχει την υποκλάση HeapNode, η οποία αναπαριστά τους κόμβους του δένδρου Huffman που θα “χτίσουμε” για να γίνει η κωδικοποίηση. Κάθε κόμβος που βρίσκεται στα κλαδιά του δένδρου αποτελείται από έναν χαρακτήρα και μια τιμή η οποία αναπαριστά τη συχνότητα του στο .txt αρχείο, ενώ οι υπόλοιποι κόμβοι του δένδρου αποτελούνται μόνο από μια τιμή και όχι χαρακτήρα.



Τα βήματα που χρησιμοποιούνται για να υλοποιηθεί η κωδικοποίηση Huffman είναι τα εξής:

- 1) Δημιουργία python dictionary που θα αποθηκεύει τη συχνότητα του κάθε χαρακτήρα μέσα στο .txt αρχείο.
- 2) Δημιουργία priority queue (MinHeap) έτσι ώστε όταν αφαιρούμε ένα στοιχείο από τη στοίβα, αυτό θα είναι το ελάχιστο.
- 3) “Χτίσιμο” δένδρου Huffman: επέλεξε τους δύο μικρότερους κόμβους από τη στοίβα και ένωσε τους μέχρι να υπάρχει μόνο ένας κόμβος στη στοίβα.
- 4) Ανάθεση codewords σε κάθε ξεχωριστό χαρακτήρα: διατρέχουμε το δέντρο από τη ρίζα – αν ακολουθήσουμε αριστερό μονοπάτι προσθέτουμε 0 στον κωδικό, διαφορετικά 1.
- 5) Κωδικοποίηση του αρχείου: έχοντας dictionary αντιστοιχίας codeword – character, αντικατάσταση κάθε χαρακτήρα με τον κωδικό που του αντιστοιχεί.
- 6) Αν το μέγεθος του τελικού κωδικοποιημένου αρχείου δεν είναι πολλαπλάσιο του 8, προσθήκη ψηφίων μέχρι να γίνει καθώς το αρχείο θα πρέπει να αποθηκευτεί σε binary μορφή.
- 7) Αποθήκευση πληροφορίας σχετικά με τα bits που προστέθηκαν στην αρχή του κωδικοποιημένου αρχείου ώστε να διαβαστεί αργότερα από τον αποκωδικοποιητή.

Η υλοποίηση των δύο πρώτων βημάτων γίνεται με τη βοήθεια των συναρτήσεων `create_freq_dict()` και `create_heap()` αντίστοιχα. Η πρώτη, διατρέχει όλο το αρχείο .txt και φτιάχνει ένα dictionary στο οποίο αποθηκεύει τη συχνότητα εμφάνισης του κάθε χαρακτήρα στο αρχείο, ενώ η δεύτερη για κάθε στοιχείο που βρίσκεται στο dictionary φτιάχνει ένα αντικείμενο της κλάσης `HearNode` και το προσθέτει στη στοίβα με βάση τη συχνότητα εμφάνισης του.

Αφού δημιουργηθεί το frequency dictionary και η στοίβα μπορούμε πλέον να κάνουμε merge τους κόμβους που βρίσκονται πλέον μέσα στη στοίβα δημιουργώντας με αυτόν τον τρόπο το δένδρο Huffman. Η συνάρτηση `merge_nodes()`, εκτελεί αυτήν τη λειτουργία παίρνοντας κάθε φορά τα δύο ελάχιστα στοιχεία της στοίβας και δημιουργώντας έναν νέο κόμβο του οποίου η τιμή είναι ίση με το άθροισμα των δύο κόμβων γονέων του και στη συνέχεια προσθέτοντας τον νέο κόμβο ξανά στη στοίβα ώστε να γίνει merge αργότερα με κάποιον άλλον κόμβο.

Στη συνέχεια, μέσω των συναρτήσεων `create_codenames_help()` και `create_codenames()` ανατρέχουμε το δένδρο που δημιουργήθηκε πλασματικά προηγουμένως και αναθέτουμε έναν κωδικό σε κάθε χαρακτήρα όπως περιγράφηκε και στο βήμα (4). Τα codenames για τον κάθε χαρακτήρα αποθηκεύονται στα `dictionaries codes{}` και `reverse_codes{}` που αντιστοιχούν τον κάθε χαρακτήρα με τον κωδικό του και αντίστροφα, αντίστοιχα.

Αφού εκτελεστούν οι παραπάνω διαδικασίες, μπορούμε να προχωρήσουμε στην κωδικοποίηση του .txt αρχείου, αντικαθιστώντας τον κάθε χαρακτήρα με το codename του. Για την υλοποίηση της διαδικασίας αυτής έχει δημιουργηθεί η συνάρτηση `get_encoded_text()`, η οποία παίρνει σαν όρισμα το αρχικό αρχείο και επιστρέφει ένα νέο κωδικοποιημένο πλέον αρχείο. Για να ολοκληρωθεί το κωδικοποιημένο αρχείο και να αποθηκευτεί σε binary μορφή πρέπει να εκτελεστούν τα βήματα 6 και 7 που αναλύσαμε παραπάνω, πράγμα το οποίο γίνεται μέσω των συναρτήσεων `pad_encoded_text()` και `get_byte_array()`.

Τελικά, λοιπόν, αφού έχουν οριστεί όλες οι παραπάνω συναρτήσεις, μπορούμε να τις εκτελέσουμε διαδοχικά προσομοιώνοντας τη λειτουργία του κωδικοποιητή. Η συνάρτηση `encode()`, παίρνοντας ένα αρχείο .txt κάθε φορά εκτελεί διαδοχικά τις παραπάνω συναρτήσεις και τα βήματα της κωδικοποίησης Huffman που αναλύσαμε παραπάνω, δημιουργώντας ένα κωδικοποιημένο πλέον αρχείο και επιστρέφοντας το path στο οποίο έχει αποθηκευτεί για ενημέρωση του χρήστη.

Μετά την ολοκλήρωση της διαδικασίας κωδικοποίησης των αρχείων .txt που αναπαριστούν τις εικόνες σφάλματος, ορίζονται οι συναρτήσεις που χρησιμοποιούνται για την αποκωδικοποίηση των κωδικοποιημένων αρχείων. Οι συναρτήσεις αυτές έχουν δημιουργηθεί ώστε να εκτελούν τα παρακάτω βήματα, τα οποία θα οδηγήσουν στην αποκωδικοποίηση των αρχείων.

1) Ανάγνωση του κωδικοποιημένου αρχείου.

2) Αφαίρεση των πρόσθετων bits που προστέθηκαν πιθανών για να έχουν τα αρχεία μέγεθος πολλαπλάσιο του 8 ώστε να αποθηκευτούν ως binary αρχεία.

3) Δημιουργία και αποθήκευση αποκωδικοποιημένου αρχείου: ανάγνωση του κωδικοποιημένου αρχείου και κάθε φορά που συναντάται κωδικός που αντιστοιχεί σε codename που έχουμε αποθηκευμένο στα dictionaries που δημιουργήσαμε προηγουμένως, αντικατάσταση του από τον χαρακτήρα που του αντιστοιχεί.

Για την εκτέλεση του 1<sup>ου</sup> βήματος έχουμε δημιουργήσει τη συνάρτηση `remove_padding()` η οποία αφαιρεί τα πρόσθετα bits διαβάζοντας από την αρχή του κειμένου αν και πόσα bits προστέθηκαν και τέλος επιστρέφει το κωδικοποιημένο αρχείο χωρίς να έχει πλέον τα πρόσθετα bits.

Στη συνέχεια, για την αποκωδικοποίηση εκτελείται η συνάρτηση `decode_text()`, η οποία παίρνει ως όρισμα ένα αρχείο κωδικοποιημένο και αφού κάνει την αντικατάσταση που αναλύσαμε στο βήμα (3), επιστρέφει το αποκωδικοποιημένο αρχείο, το οποίο θα πρέπει να είναι πανομοιότυπο με το αρχικό που κωδικοποιήσαμε και ίδιου μεγέθους.

Τέλος, προσομοιώνοντας τον αποκωδικοποιητή, έχει δημιουργηθεί η συνάρτηση `decode()` που καλεί τις δύο προηγούμενες συναρτήσεις, οδηγώντας στην τελική αποκωδικοποίηση του αρχείου και επιστρέφοντας το path στο οποίο έχει αποθηκευτεί για ενημέρωση του χρήστη.

## codecs-huffman.py

Το συγκεκριμένο αρχείο χρησιμοποιώντας τις συναρτήσεις που ορίστηκαν στο huffman.py, αλλά και εκτελώντας τη συνάρτηση jpg\_to\_txt() που αναλύσαμε παραπάνω, κωδικοποιεί και αποκωδικοποιεί διαδοχικά όλα τα .txt αρχεία που έχουμε αποθηκεύσει και αναπαριστούν τις εικόνες σφάλματος.

```
def encode_decode(filename):
    path = "C:\\Users\\golem\\Desktop\\TASK 1\\huffman_encoding\\error_frames\\pixel_values\\" + filename
    H = HuffmanCoding(path)
    output_path = H.encode()
    H.decode(output_path)

    print("Finished with: " + filename)




if __name__ == "__main__":
    txt_names = jpg_to_txt()

    print(txt_names)

    for i in range(len(txt_names)):
        j = txt_names[i]
        encode_decode(j)
```

Όπως φαίνεται και από το παραπάνω κομμάτι κώδικα εκτελούνται διαδοχικά οι συναρτήσεις κωδικοποίησης και αποκωδικοποίησης για κάθε αρχείο. Το μονοπάτι path θα πρέπει να οριστεί ανάλογα με την τοποθεσία που έχουν αποθηκευτεί οι εικόνες και το directory του κώδικα στον εκάστοτε υπολογιστή. Με την κωδικοποίηση και αποκωδικοποίηση του κάθε αρχείου εμφανίζεται και ένα μήνυμα στον χρήστη καθώς είναι μια διαδικασία που παίρνει λίγα λεπτά. Αφού ολοκληρωθεί η διαδικασία, έχει δημιουργηθεί ένας φάκελος ο οποίος περιέχει κάθε αρχικό αρχείο καθώς και τα αντίστοιχα κωδικοποιημένα και αποκωδικοποιημένα αρχεία. Για παράδειγμα όπως φαίνεται από την παρακάτω εικόνα για το error\_frame1 υπάρχει το αρχικό του αρχείο .txt που περιέχει τα pixel values του (error\_frame1.txt), το κωδικοποιημένο αρχείο

(error\_frame1.bin) αποθηκευμένο σε binary μορφή και το αποκωδικοποιημένο αρχείο (error\_frame1\_decoded.txt).

 error_frame1.bin	6/23/2021 2:59 PM	BIN File	1,002 KB
 error_frame1.txt	6/23/2021 2:58 PM	Text Document	2,767 KB
 error_frame1_decoded.txt	6/23/2021 2:59 PM	Text Document	2,767 KB

Είναι εμφανές ότι το μέγεθος του κωδικοποιημένου αρχείου (1002 kb) είναι πολύ μικρότερο από το αρχικό (2767 kb), ενώ το αποκωδικοποιημένο αρχείο έχει ακριβώς το ίδιο μέγεθος με το αρχικό, και μπορούμε να διαπιστώσουμε πώς είναι ακριβώς το ίδιο αρχείο. Μπορούμε να συμπεράνουμε, λοιπόν, πώς η ζητούμενη διαδικασία encoding – decoding ολοκληρώθηκε με επιτυχία.

### **Εκτέλεση αρχείων πηγαίου κώδικα για επίλυση του ερωτήματος**

Όσον αφορά την εκτέλεση των αρχείων πηγαίου κώδικα για το 1<sup>ο</sup> ερώτημα, θα πρέπει στο ίδιο φάκελο να υπάρχουν τα αρχεία frames.py, huffman.py, codecs-huffman.py μαζί με το βίντεο του περαστικού από το πάρκο με όνομα grayscale.mp4. Στη συνέχεια πρέπει να εκτελεστεί αρχικά το αρχείο frames.py δημιουργώντας έτσι κάποιους φακέλους που αναλύσαμε παραπάνω και έπειτα να εκτελεστεί το αρχείο codecs-huffman.py που θα δημιουργήσει τα κωδικοποιημένα αρχεία στον φάκελο error\_frames/pixel\_values του ίδιου directory. Είναι σημαντικό στη γραμμή 52 του αρχείου codecs\_huffman.py να αντικατασταθεί με το αντίστοιχο path που έχουν αποθηκευτεί τα αρχεία στον εκάστοτε υπολογιστή.

## **Ερώτημα (ii) – motion\_compensation.py**

Για την υλοποίηση του ερωτήματος (ii) χρησιμοποιούνται τα αρχεία `frames.py`, `motion_compensation.py`, `codecs_motionCompensation.py`, τα οποία θα αναλύσουμε στη συνέχεια. Στο συγκεκριμένο ερώτημα σκοπός μας είναι να υλοποιήσουμε την τεχνική αντιστάθμισης κίνησης για το βίντεο με τον περαστικό που χρησιμοποιήσαμε και στο προηγούμενο ερώτημα, να υπολογίσουμε και να απεικονίσουμε την ακολουθία εικόνων πρόβλεψης και σφάλματος και τελικά να υλοποιηθεί ο κωδικοποιητής και ο αποκωδικοποιητής.

Ξεκινώντας τη διαδικασία, είναι απαραίτητο να έχουμε τα `frames` του βίντεο, πράγμα που δεν χρειάζεται να επαναλάβουμε καθώς έχει ήδη γίνει για το προηγούμενο ερώτημα χρησιμοποιώντας το αρχείο `frames.py`. Τα `frames` αυτά είναι αποθηκευμένα στον φάκελο `huffman_coding/frames` και για να έχουμε πρόσβαση σε αυτά, φτιάχνουμε μια λίστα που περιέχει το κάθε `frame` σε μορφή μήτρας μέσω της `cv2.imread()`.

Το κύριο αρχείο που χρησιμοποιείται και ορίζει τις συναρτήσεις για την υλοποίηση της τεχνικής αντιστάθμισης κίνησης είναι το `motion_compensation.py`, το οποίο θα αναλύσουμε στη συνέχεια.

Ξεκινώντας τον αλγόριθμο, θα πρέπει το κάθε `frame` να χωριστεί σε `macroblocks`, χρησιμοποιώντας τη συνάρτηση `divide_frame()` που έχουμε δημιουργήσει η οποία παίρνει ως όρισμα ένα `frame` και επιστρέφει τον αριθμό των κάθετων αλλά και οριζόντιων `macroblocks` που αυτή έχει, ανάλογα με το ύψος και το πλάτος της κάθε εικόνας. Στην περίπτωση μας οι εικόνες είναι  $720 * 1280$  επομένως αφού από την εκφώνηση έχει οριστεί πώς το μέγεθος του κάθε `block` είναι  $16 * 16$ , τα οριζόντια μπλοκ θα είναι  $720 / 16 = 45$ , ενώ τα κάθετα θα είναι  $1280 / 16 = 80$ .

Έπειτα ορίζεται η συνάρτηση `find_center()` η οποία ανάλογα με τα `coordinates` που της δοθούν ως όρισμα, υπολογίζει και επιστρέφει τα `coordinates` του κέντρου για το συγκεκριμένο μπλοκ. Για παράδειγμα αν

δοθεί ως όρισμα (0, 0) θα δοθεί ως αποτέλεσμα πώς οι συντεταγμένες του κέντρου είναι (8, 8) αφού το μπλοκ είναι μεγέθους  $16 * 16$ .

Χρησιμοποιώντας τις δύο παραπάνω συναρτήσεις, ορίζουμε την `find_search_area()`, η οποία θα υπολογίζει τις συντεταγμένες της αριστερής γωνίας του συνόλου των μπλοκ που θα απαρτίζουν την περιοχή αναζήτησης και θα επιστρέφει την ίδια την περιοχή αναζήτησης κάνοντας slice την περιοχή του numpy array που μας ενδιαφέρει. Για παράδειγμα αν θελήσουμε να βρούμε την περιοχή αναζήτησης για το pixel στη θέση (24, 24), θα βρούμε ότι η περιοχή αναζήτησης στον οριζόντιο άξονα ξεκινάει από (0, 0) -> (48, 0), ενώ στον κατακόρυφο άξονα από (0, 0) -> (48, 0) καθώς η ακτίνα αναζήτησης έχει οριστεί πώς πρέπει να είναι  $16 * 16$ .

Στη συνέχεια ορίζεται η `get_block_zone()` που λειτουργεί σε συνδυασμό με την `find_search_area()` και επιστρέφει κάθε φορά ένα από τα macroblocks της επιφάνειας αναζήτησης για να συγκριθεί με το τρέχον μπλοκ, ανάλογα με τις συντεταγμένες της επιφάνειας αναζήτησης που της δοθούν ως όρισμα.

Αφού έχουμε βρεί την περιοχή στην οποία πρέπει να αναζητήσουμε τα μπλοκ που προσπαθούμε να προβλέψουμε και μπορούμε να παρούμε κάθε ένα από αυτά μέσω της `get_block_zone()`, ορίζουμε τη συνάρτηση `find_mad()`, η οποία χρησιμοποιείται για τη σύγκριση των μπλοκ και επιστρέφει την τιμή Mean Absolute Difference για κάθε ζεύγος που δοθεί ως όρισμα.

Για την εύρεση του προβλεφθέντος μπλοκ ώστε να δημιουργηθεί η εικόνα πρόβλεψης στη συνέχεια χρησιμοποιώντας τα μπλοκ του προηγούμενου frame, ορίζουμε τη συνάρτηση `find_match()`, η οποία παίρνοντας ως όρισμα το τρέχον μπλοκ και την περιοχή αναζήτησης που βρίσκουμε μέσω της `find_search_area()`, επιστρέφει το μπλοκ της περιοχής αναζήτησης με την μικρότερη τιμή Mean Absolute Difference. Για την αναζήτηση των μπλοκ στην περιοχή αναζήτησης έχει υλοποιηθεί ο αλγόριθμος Three Step Search τα βήματα του οποίου είναι τα εξής:

- 1) Start with search location at center



- 2) Set step size  $S = 4$  and search parameter  $p = 7$
- 3) Search 8 locations  $\pm S$  pixels around location (0,0) and the location (0,0)
- 4) Pick among the 9 locations searched, the one with minimum cost function
- 5) Set the new search origin to the above picked location
- 6) Set the new step size as  $S = S/2$
- 7) Repeat the search procedure until  $S = 1$

Πηγή: [https://en.wikipedia.org/wiki/Block-matching\\_algorithm](https://en.wikipedia.org/wiki/Block-matching_algorithm)

Αφού ορίσουμε όλες τις παραπάνω συναρτήσεις οι οποίες μας δίνουν τη δυνατότητα για κάθε μπλοκ του `frame_n+1` να βρούμε ένα μπλοκ στο `frame_n` που να είναι σχεδόν πανομοιότυπο, μπορούμε πλέον να προχωρήσουμε στη δημιουργία του προβλεφθέντος frame. Αυτό γίνεται μέσω της συνάρτησης `create_predicted()`, η οποία παίρνει σαν όρισμα δύο διαδοχικά frames και αφού τρέξει τη `find_match()` για κάθε μπλοκ του `target_frame`, δημιουργεί το predicted frame, προσθέτοντας τα values των matching blocks σε ένα numpy array που αρχικά περιείχε μόνο τιμές 1 με διαστάσεις 1280\*720.

Τέλος, ορίζονται οι συναρτήσεις `find_residual()` και `reconstruct_target()`. Η `find_residual()` είναι η τελική συνάρτηση που χρειάζεται για να ολοκληρωθεί η λειτουργία του κωδικοποιητή καθώς αφαιρώντας το πραγματικό frame από το προβλεφθέν, βρίσκει την εικόνα σφάλματος η οποία είναι απαραίτητη για τον αποκωδικοποιητή ο οποίος για να ανακατασκευάσει την αρχική εικόνα προσθέτει το προβλεφθέν και το frame σφάλματος μέσω της συνάρτησης `reconstruct_target()`.

## **codecs\_motionCompensation.py**

Έχοντας ορίσει τις συναρτήσεις και τις μεταβλητές που χρειάζονται για να εκτελέσουμε την τεχνική αντιστάθμισης κίνησης, δημιουργούμε το αρχείο `codecs_motionCompensation.py`, το οποίο περιέχει συναρτήσεις οι οποίες προσομοιώνουν τη λειτουργία του κωδικοποιητή και του αποκωδικοποιητή.

Ξεκινώντας, φτιάχνεται μια λίστα που περιέχει τα frames του βίντεο που είχαν δημιουργηθεί προηγουμένως. Τα frames αυτά βρίσκονται στη λίστα σε μορφή μητρώων που έχουν ως στοιχεία τα pixel values της κάθε εικόνας, με τη βοήθεια της `cv2.imread()`.

Εν συνεχεία, ορίζουμε τη συνάρτηση `encoder()`, η οποία παίρνει ως όρισμα δύο διαδοχικά frames και επιστρέφει το προβλεφθέν frame αλλά και το residual frame που αναπαριστά το σφάλμα πρόβλεψης. Ο υπολογισμός των παραπάνω frames, γίνεται με τη βοήθεια των συναρτήσεων `create_predicted()`, `find_residual()` που ορίσαμε στο αρχείο `motionCompensation.py`. Η συγκεκριμένη συνάρτηση λοιπόν εκτελεί την κωδικοποίηση μιας εικόνας χρησιμοποιώντας την τεχνική αντιστάθμισης κίνησης.

Επόμενη συνάρτηση που ορίζεται στο συγκεκριμένο αρχείο είναι η `decoder()`, η οποία παίρνει ως όρισμα ένα residual frame και ένα predicted frame και επιστρέφει το ανακατασκευασμένο frame, προσομοιώνοντας έτσι τη λειτουργία ενός αποκωδικοποιητή. Η `decoder()`, καλώντας τη συνάρτηση `reconstruct_target()` του αρχείου `motion_compensation.py`, προσθέτει την εικόνα διαφορών με την προβλεφθείσα εικόνα φτιάχνοντας έτσι μια νέα εικόνα που είναι πανομοιότυπη με την αρχική πριν γίνει κωδικοποίησή της.

Τέλος, η `main()` σε συνδυασμό με τη συνάρτηση `helper()`, καλούν διαδοχικά τις συναρτήσεις κωδικοποίησης και αποκωδικοποίησης για κάθε

frame του βίντεο φτιάχνοντας παράλληλα φακέλους στους οποίους αποθηκεύονται τα frames ανά κατηγορία για λόγους οπτικοποίησης του αποτελέσματος. Όλα τα frames αποθηκεύονται σε φακέλους οι οποίοι είναι υποφάκελοι του motion\_compensation/ στο directory που βρίσκονται τα αρχεία πηγαίου κώδικα. Συγκεκριμένα, τα frames που θέλουμε να προβλέψουμε αποθηκεύονται στον φάκελο target\_frames, οι εικόνες πρόβλεψης στον φάκελο pframes, οι εικόνες διαφορών στον φάκελο residual\_frames και τέλος, οι ανακατασκευασμένες εικόνες στον φάκελο reconstructed\_frames.

Παρακάτω ακολουθούν κάποιες από τις εικόνες που προκύπτουν από την κωδικοποίηση ανά κατηγορία:



Στην παραπάνω εικόνα φαίνεται το προβλεφθέν frame χρησιμοποιώντας τα macroblocks της προηγούμενης εικόνας. Ξανά το γεγονός ότι επιλέξαμε να κρατήσουμε 1/10 frames έχει επίπτωση στην ανάλυση των εικόνων πρόβλεψης. Όσο προχωράνε οι εικόνες πρόβλεψης, η ανάλυση χειροτερεύει καθώς κάθε φορά χρησιμοποιείται το προηγούμενο P-frame, χωρίς να έχουμε ενδιάμεσα I-frames. Στην παρακάτω εικόνα φαίνεται το πόσο χειροτερεύει η ανάλυση σε ένα ενδιάμεσο P-frame.



**predicted - frame**

**target - frame**



**residual - frame**



Η παραπάνω εικόνα διαφορών απεικονίζει το σφάλμα πρόβλεψης ανάμεσα στο ζεύγος predicted, target frame, δείχνοντας πώς παρότι η ανάλυση είναι κακή, το μεγαλύτερο ποσοστό πληροφορίας έχει αποτυπωθεί στην εικόνα πρόβλεψης. Χρησιμοποιώντας τα παραπάνω

predicted, residual frames, ο αποκωδικοποιητής που υλοποιήσαμε μπορεί να ανακατασκευάσει την εικόνα στόχου ξανά η οποία φαίνεται παρακάτω και είναι πανομοιότυπη με την αρχική εικόνα.

**reconstructed - frame**



### **Εκτέλεση αρχείων πηγαίου κώδικα για επίλυση του ερωτήματος**

Για τη σωστή εκτέλεση των αρχείων θα πρέπει να υπάρχουν στο ίδιο directory το βίντεο με όνομα grayscale.mp4 καθώς και τα αρχεία motion\_compensation.py, codecs\_motionCompensation.py, frames.py. Αρχικά θα πρέπει και για το συγκεκριμένο ερώτημα να εκτελεστεί το frames.py όπως και για το ερώτημα (i) ώστε να πάρουμε τα frames του βίντεο. Στη συνέχεια εκτελείται το codecs\_motionCompensation.py, από το οποίο θα προκύψουν οι φάκελοι που θα περιέχουν τις απαραίτητες κωδικοποιημένες, αποκωδικοποιημένες, προβλεφθείσες και εικόνες σφαλμάτων όπως αναλύσαμε παραπάνω.

## **Θέμα 2 (4 βαθμοί):**

*Σε βίντεο της επιλογής σας, διάρκειας 5 s – 15 s, στο οποίο υπάρχει ήπια κίνηση αντικειμένου και κάμερας, επιλέξτε ένα αντικείμενο και εξαφανίστε το. Δημιουργήστε και αποθηκεύστε δηλαδή ένα νέο βίντεο στο οποίο δεν θα υπάρχει το αντικείμενο που επιλέξατε. Για τον σκοπό αυτόν, αξιοποιήστε την τεχνική αντιστάθμισης κίνησης.*

Για την επίλυση του συγκεκριμένου θέματος έχουμε δημιουργήσει τα αρχεία `main.py`, `remove_car.py` τα οποία και θα αναλύσουμε παρακάτω. Για τον σκοπό του ερωτήματος έχουμε χρησιμοποιήσει ένα βίντεο που απεικονίζει ένα αυτοκίνητο να κινείται τραβηγμένο από drone, του οποίου το 1<sup>ο</sup> frame φαίνεται στην παρακάτω εικόνα.



Στόχος μας είναι χρησιμοποιώντας την τεχνική αντιστάθμισης κίνησης, να δημιουργήσουμε ένα νέο βίντεο στο οποίο δεν θα υπάρχει το αυτοκίνητο, δηλαδή να αφαιρέσουμε το αυτοκίνητο.

Η λειτουργία του αρχείου `main.py` είναι παρόμοια με το αρχείο `frames.py` του προηγούμενου ερωτήματος καθώς χρησιμοποιείται ώστε να κόψει το βίντεο σε frames ώστε να μπορέσουμε να εκτελέσουμε τον αλγόριθμο. Στο θέμα 2 όπως και στο 1, επιλέγουμε να κρατάμε 1 ανά 10 frames για μείωση του αποθηκευτικού χώρου και μεγαλύτερη ευκολία.

### **remove-car.py**

Στο συγκεκριμένο αρχείο, υπάρχουν οι περισσότερες συναρτήσεις που φτιάξαμε στο αρχείο `motion_compensation.py` για το προηγούμενο ερώτημα πανομοιότυπες ώστε να μας βοηθήσουν να εκτελέσουμε τον αλγόριθμο αντιστάθμισης κίνησης ενώ παράλληλα έχουν δημιουργηθεί νέες ώστε να μπορέσουμε τελικά να αφαιρέσουμε το αυτοκίνητο από το βίντεο.

Ξεκινώντας, έχουμε επιλέξει να εστιάσουμε στο συγκεκριμένο σημείο των frames στο οποίο βρίσκεται το όχημα ενώ κινείται κατά τη διάρκεια του βίντεο. Μέσω της συνάρτησης `get_car()`, λοιπόν, φτιάχνουμε έναν φάκελο ονόματι `frames/specific_area`, ο οποίος περιέχει για κάθε frame του



βίντεο, μόνο το κομμάτι που βρίσκεται το όχημα κάθε φορά. Για παράδειγμα για το 1<sup>ο</sup> frame του βίντεο έχουμε τις δύο παρακάτω εικόνες:



Φτιάχνοντας τα συγκεκριμένα frames, λοιπόν, και εστιάζοντας αρχικά στο κομμάτι των εικόνων που βρίσκεται μόνο το αυτοκίνητο μπορούμε να συμπεριφερθούμε σαν να έχουμε ένα βίντεο που δεν κινείται η κάμερα καθώς το background με την άσφαλτο είναι σχεδόν το ίδιο σε κάθε frame.

Ο αλγόριθμος που θα ακολουθήσουμε για την αφαίρεση του βίντεο είναι ο παρακάτω:

- 1) Εκτέλεση του αλγορίθμου αντιστάθμισης κίνησης όπως κάναμε στο προηγούμενο θέμα.

2) Έχοντας τελικά βρεί τα matching blocks μέσω της find\_match() και των υπόλοιπων συναρτήσεων που αναλύσαμε στο προηγούμενο θέμα για την αντιστάθμιση κίνησης, εύρεση των motion vectors.

3) Αν η τιμή των motion vectors είναι κάτω από ένα όριο που ορίζουμε, θεωρούμε ότι το συγκεκριμένο block είναι κομμάτι ασφάλτου και δεν κινείται. Τα υπόλοιπα blocks, εφόσον η τιμή των motion vectors τους δεν είναι μηδενική, καταλαβαίνουμε πώς είναι κομμάτι του αυτοκινήτου και τα αφαιρούμε βάζοντας στη θέση τους το κομμάτι του προηγούμενου frame στο οποίο το αυτοκίνητο δεν υπήρχε.

4) Έχοντας αφαιρέσει το αυτοκίνητο από όλες τις μικρότερες εικόνες που είναι κομμάτι του συνολικού frame, προσθήκη του κομματιού αυτού στην αρχική εικόνα, αφαιρώντας έτσι το αυτοκίνητο και από το αρχικό, μεγαλύτερο frame.

5) Ένωση όλων των frames που δεν περιέχουν το βίντεο ώστε να δημιουργήσουμε πλέον το βίντεο το οποίο δεν θα περιλαμβάνει το αυτοκίνητο.

Αφού βρούμε τα matching blocks όπως αναφέρουμε στο βήμα (1), καλείται η συνάρτηση find\_motion() για την εύρεση των motion vectors. Η find\_motion() επιστρέφει τη μήτρα pixels\_moved η οποία για κάθε μπλοκ περιέχει την τιμή που κινήθηκε από το frame\_n στο frame\_n+1. Η τιμή των motion vectors βρίσκεται από τον παρακάτω τύπο:

$$distance = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Έτσι εκτελώντας τη find\_motion(), έχουμε μια μήτρα που το κάθε στοιχείο της μας υποδηλώνει πόσο κινήθηκε το κάθε macroblock.

Στη συνέχεια, γνωρίζοντας τα motion vectors και ακολουθώντας τα βήματα του αλγορίθμου που αναφέραμε παραπάνω, μέσω των `remove_car_helper()` και `remove_car()`, αφαιρούμε το αυτοκίνητο από κάθε ένα από τα επιμέρους frames. Αυτό γίνεται με τον εξής τρόπο: Αρχικά μηδενίζουμε κάθε motion vector που η τιμή του είναι μικρότερη από 100 θεωρώντας αυτή την κίνηση αμελητέα και καταλαβαίνοντας ότι πρόκειται για κομμάτι ασφάλτου. Στη συνέχεια, κάθε μπλοκ που έχει μη μηδενικό πλέον motion vector, το αφαιρούμε και το αντικαθιστούμε με το αντίστοιχο μπλοκ του προηγούμενου frame στο οποίο δεν υπήρχε το αυτοκίνητο, φτιάχνοντας έτσι μια επιμέρους εικόνα του βίντεο στην οποία δεν υπάρχει πλέον το όχημα. Οι επιμέρους εικόνες αυτές, για λόγους οπτικοποίησης του αποτελέσματος αποθηκεύονται σε ξεχωριστούς φακέλους μέσω των συναρτήσεων.

Εν συνεχεία έχοντας πλέον τις επιμέρους εικόνες που δεν περιλαμβάνουν το αυτοκίνητο, αφαιρούμε μέσω της `final_removal()`, το κομμάτι του αυτοκινήτου και προσθέτουμε το αντίστοιχο επιμέρους frame στο οποίο αφαιρέσαμε το αυτοκίνητο. Έτσι, καταλήγουμε να έχουμε τα τελικά frames που δεν περιέχουν πια το αντικείμενο που θέλουμε να αφαιρέσουμε.

Τελικά, εκτελώντας την `frames_to_video()`, ενώνουμε όλα τα frames από τα οποία αφαιρέσαμε το αυτοκίνητο, δημιουργώντας έτσι το τελικό ζητούμενο βίντεο για την επίλυση του 2<sup>ου</sup> θέματος.

Όλες οι παραπάνω διαδικασίες που αναφέραμε εκτελούνται διαδοχικά μέσω της `main()`. Αρχικά, καλούμε την `get_car()` ώστε να πάρουμε τα επιμέρους κομμάτια των frames του βίντεο που περιλαμβάνουν μόνο το αμάξι και στη συνέχεια αφαιρούμε το αμάξι από αυτά καλώντας τη `remove_car()`. Τελικά, καλώντας τις `final_removal()` και `frames_to_video()`, αφαιρούμε το κομμάτι που υπήρχε το αμάξι, προσθέτουμε το κομμάτι του frame από το οποίο έχει αφαιρεθεί το αμάξι και εν τέλει ενώνουμε τα frames δημιουργώντας το ζητούμενο βίντεο.

## Εκτέλεση αρχείων πηγαίου κώδικα για επίλυση του ερωτήματος

Για την επίλυση του συγκεκριμένου θέματος θα πρέπει τα αρχεία main.py, remove\_car.py να βρίσκονται στο ίδιο directory μαζί με το βίντεο car\_moving.mp4. Αρχικά, εκτελούμε το αρχείο main.py ώστε να πάρουμε τα frames του βίντεο. Στο ίδιο directory θα δημιουργηθεί ο φάκελος frames ο οποίος θα περιέχει τα frames του βίντεο, στον οποίον θα πρέπει να προστεθεί η εικόνα background.jpg, η οποία αποτελεί frame χωρίς το αμάξι και δεν αποτυπώνεται στη διάρκεια του βίντεο που έχουμε επιλέξει. Στη συνέχεια, θα πρέπει να εκτελεστούν οι συναρτήσεις του remove\_car.py όπως έχουν γραφτεί στο παρακάτω κομμάτι κώδικα και όχι ταυτόχρονα αλλά η μια μετά την άλλη ώστε να φτιαχτούν οι εικόνες που αναλύσαμε παραπάνω αλλά και τελικά το βίντεο που δεν θα περιλαμβάνει πια το αμάξι να κινείται:

```
if __name__ == "__main__":

    # get_car()

    # remove_car()

    # try:
    #     if not os.path.exists('frames/final/'):
    #         os.makedirs('frames/final')
    # except OSError:
    #     print("Error creating folder!")

    # for i in range(len(frames)):
    #     final = final_removal(no_car[i], frames[i+1])
    #     cv2.imwrite("frames/final/final" + str(i) + ".jpg", final)

    frames_to_video()
```