ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
Τμήμα Πληροφορικής

Εργασία Μαθήματος «Συστήματα Πολυμέσων»

Εργασία Μαθήματος	Τελική εργασία μαθήματος
Εκπαιδευτής	Άγγελος Πικράκης
Όνομα φοιτητή – Αρ. Μητρώου	Αργυροπούλου Μαρία : Π18011
	Μάλλιος Αναστάσιος : Π19216
Ημερομηνία παράδοσης	20-09-2022

Περιεχόμενα

Εισαγωγή

Για την ανάπτυξή των προγραμμάτων κάναμε την επιλογή να χρησιμοποιήσουμε το λογισμικό PyCharm και για γλώσσα προγραμματισμού τη Python (3.9), λόγο της εξοικείωσής μας με την γλώσσα. Έγινε επίσης χρήση των βιβλιοθηκών Numpy και Cv2.

Θέμα 1ο

Εκφώνηση

Θέμα 1 (1.5 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 32x32, ακτίνα αναζήτησης k=16 και τεχνική σύγκρισης macroblocks της επιλογής σας. Να επιταχυνθεί η διαδικασία υλοποιώντας ιεραρχική αναζήτηση. Υπολογίστε, αποθηκεύστε και απεικονίστε την ακολουθία εικόνων πρόβλεψης και εικόνων σφαλμάτων. Υλοποιήστε τον κωδικοποιητή/αποκωδικοποιητή.

Υλοποίηση i)

<u>Εισαγωγή/Ορισμός του προβλήματος</u>

Κύρια ζητήματα στην συγκεκριμένη εργασία είναι:

- Κάθε πλαίσιο P να προκύπτει από το προηγούμενο πλαίσιο.
- Ο υπολογισμός της ακολουθίας εικόνων σφάλματος.
- Η κωδικοποίηση της χωρίς απώλειες.
- Η αποκωδικοποίηση της ακολουθίας.

<u>Παραδοχές/Υποθέσεις</u>

- Γίνεται η υπόθεση πως δεν υπάρχει πρόβλημα να μετατραπεί κάθε πλαίσιο σε grayscale.
- Γίνεται η παραδοχή της χρήσης κωδικοποιητή Huffman καθώς είναι τεχνική χωρίς απώλειες και επίσης παρέχει συμπίεση.
- Γίνεται η παραδοχή της χρήσης αρχείου κειμένου (.txt) για την αποθήκευση των bits που προκύπτουν από την κωδικοποίηση Huffman καθώς

και την αποθήκευση του αντίστοιχου λεξικού που χρειάζεται για την αποκωδικοποίηση.

Προεπεξεργασία

Για την υλοποίηση του θέματος αρχικά, έγιναν δοκιμές για την επίλυση ενός μικρότερου προβλήματος, παίρνοντας τα 3 πρώτα πλαίσια από ένα βίντεο. Το πρώτο πλαίσιο ορίστηκε ως I-frame ενώ τα υπόλοιπα 2 ως P-frames. Τα πλαίσια μετατρέπονται σε grayscale καθώς είναι πιο εύκολος ο υπολογισμός τους έτσι. Για την μείωση της εντροπίας του I-frame και της αποδοτικότερης κωδικοποίησης του χρησιμοποιήθηκε ο προβλέπτης X=A (τεχνική χωρίς απώλειες) και ύστερα κωδικοποίηση Huffman. Για την κωδικοποίηση ενός P-frame, υπολογίζεται αρχικά η διαφορά του με το προηγούμενο πλαίσιο. Ύστερα μειώνεται η εντροπία της διαφοράς που προκύπτει μέσω του προβλέπτη X=A και τέλος το αποτέλεσμα κωδικοποιείται μέσω Huffman.
Για την υλοποίηση του κωδικοποιητή και αποκωδικοποιητή έχουν δημιουργηθεί τα εξής αρχεία: encoder.py (ο κωδικοποιητής), decoder.py (ο αποκωδικοποιητής), huffman.py (χρησιμοποιείται για κωδικοποίηση/αποκωδικοποίηση Huffman) και το bitstring.txt στο οποίο αποθηκεύονται τα bits από την κωδικοποίηση Huffman καθώς και το λεξικό για την αποκωδικοποίηση.

Επίλυση θέματος

Περιγραφή αρχείου encoder.py:

Αρχικά διαβάζεται από το βίντεο το πρώτο πλαίσιο, με την χρήση της βιβλιοθήκης OpenCV (cv2) και ορίζεται ως I-frame. Το I-frame μετατρέπεται σε grayscale για την ευκολία της τροποποίησης του. Στις μεταβλητές width και height αποθηκεύονται το πλάτος και το ύψος του πλαισίου αντίστοιχα.
Για την υλοποίηση του προβλέπτη X=A δημιουργήθηκε ένα for loop που επαναλαμβάνεται για όλο το ύψος του πλαισίου και με την χρήση της βιβλιοθήκης NumPy γίνεται ολίσθηση (roll) των εικονοστοιχείων μια θέση προς τα δεξιά. Επειδή με αυτόν τον τρόπο «χάνεται» η τιμή του πρώτου εικονοστοιχείου προστίθεται στην πρώτη θέση η τιμή μηδέν (0). Το αποτέλεσμα της διαδικασία είναι η εικόνα πρόβλεψης και αποθηκεύεται σε μια νέα λίστα που ονομάζεται img_diff_prediction.

```
# read frame in grayscale
video = cv2.VideoCapture('walk.mp4')
success, image = video.read()
total_frames=0

# first frame is the I-frame
I_frame = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

cv2.imshow("I-frame", I_frame)
cv2.waitKey(0)

# calculate width and height of frame
width = len(I_frame[0])
height = len(I_frame)

img_diff_prediction = []

# roll each row as X=A
for i in range(height):
    img_diff_prediction.append(np.roll(I_frame[i],1))
    # setting first value of each row 0
    img_diff_prediction[i][0] = 0
```

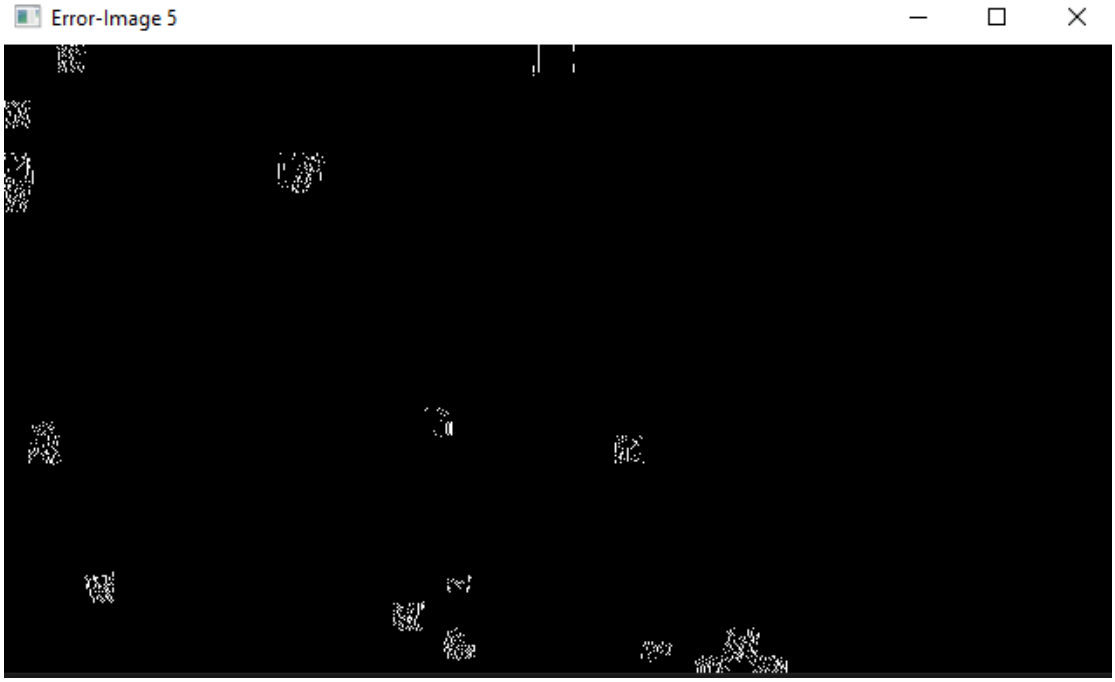
Στη συνέχεια, το αποτέλεσμα πρόβλεψης αφαιρείται από το αρχικό I-frame και δημιουργείται η εικόνα σφάλματος (error-image). Η εικόνα σφάλματος είναι χαμηλότερη σε εντροπία από το αρχικό πλαίσιο και αυτό βοηθάει στην καλύτερη και άμεση κωδικοποίηση της.
Για την κωδικοποίηση εντροπίας της εικόνας σφάλματος, πρώτα υπολογίζονται οι μοναδικές τιμές εικονοστοιχείων που περιέχονται, μαζί με το πόσες φορές επαναλαμβάνονται μέσα στην εικόνα. Αυτός ο υπολογισμός αντιστοιχεί στις πιθανότητες εμφάνισης κάθε τιμής εικονοστοιχείου που χρειάζεται για την κωδικοποίηση Huffman. Οι πιθανότητες αυτές ταξινομούνται κατά αύξουσα σειρά και δίνονται ως όρισμα στη συνάρτηση huffman.encode(), η οποία υπολογίζει και επιστρέφει ένα λεξικό με τους κώδικες Huffman που αντιστοιχούν σε κάθε τιμή εικονοστοιχείου. Η εξήγηση του κώδικα για το αρχείο huffman.py γίνεται παρακάτω.
Τέλος, μέσω ενός διπλού for loop αναζητείται για κάθε εικονοστοιχείο η αντίστοιχη κωδική λέξη στο λεξικό και προστίθεται σε μια μεταβλητή τύπου String bitstring. Η μεταβλητή bitstring στο τέλος της επανάληψης θα περιέχει όλο το πλαίσιο σε bits, τα οποία είναι συνολικά λιγότερα από τα bits της αρχικής εικόνας, επομένως επιτυγχάνεται και συμπίεση χωρίς απώλειες για το πλαίσιο.
Η μεταβλητή bitstring μαζί με το λεξικό με τους κώδικες Huffman που υπολογίστηκαν εγγράφονται σε ένα αρχείο κειμένου με όνομα bitstrng.txt το οποίο χρησιμοποιείται αργότερα από τον αποκωδικοποιητή για την

αποκωδικοποίηση των πλαισίων.

```
# calculate probabilities for each unique number in the frame
# calculate the error image
error_image = I_frame-img_diff_prediction

cv2.imshow("I-frame error image", error_image)
cv2.waitKey(0)

# get the probabilities of unique values in the I-frame difference
uniq, counts = np.unique(error_image,return_counts=True)

# convert probabilities into dictionary
probabilities = dict(np.asarray((uniq,counts)).T)
# sort dictionary by ascending order
probabilities = {k: v for k, v in sorted(probabilities.items(), key=lambda item: item[1])}
# generate huffman code
codes = huffman.encode(probabilities)
# scan error-image and convert each value to it's huffman code
bitstring = ''
for i in range(len(error_image)):
    for j in range(len(error_image[i])):
        bitstring += codes[error_image[i][j]]

f = (open('bitstring.txt', 'a'))
# 1 frame will not add " - " only p frames will write " - " before the bitstring
f.write(bitstring + " | " + str(codes))
f.close
```

Παρόμοια διαδικασία γίνεται και για τον υπολογισμό των P-frames, με κύρια διαφορά τον υπολογισμό των εικόνων διαφορών, μέσω ενός while loop, που τρέχει για όσο υπάρχουν επόμενα πλαίσια. Για κάθε P-frame υπολογίζεται η απόλυτη διαφορά (P-frame – προηγούμενο πλαίσιο), για το πρώτο P-frame το προηγούμενο του είναι το I-frame, ενώ για τα υπόλοιπα είναι το αμέσως προηγούμενο P-frame. Τα προηγούμενα πλαίσια αποθηκεύονται κάθε φορά σε μια μεταβλητή που ονομάζεται prev_frame.
Για κάθε εικόνα διαφοράς δημιουργείται η πρόβλεψης της μέσω του προβλέπτη X=A, ακριβώς με τον ίδιο τρόπο που περιεγράφηκε παραπάνω. Το αποτέλεσμα της πρόβλεψης αρχικά προβάλλεται, και ύστερα αφαιρείται από το πρωτότυπο P-frame. Με αυτόν τον τρόπο δημιουργείται η εικόνα σφάλματος η οποία είναι χαμηλότερης εντροπίας από την αρχική εικόνα διαφοράς. Για να κατανοηθεί η διαφορά στην εντροπία μεταξύ εικόνας διαφοράς και εικόνας σφάλματος προβάλλονται και οι δύο.
Όπως φαίνεται παρακάτω η εικόνα διαφοράς για το P-frame 5 είναι μεγαλύτερης εντροπίας από την εικόνα σφάλματος που προκύπτει από τον προβλέπτη X=A, καθώς παρατηρούνται πολύ πυκνότερα λευκά εικονοστοιχεία.


Image-Difference 5


Error-Image 5

```
69  prev_frame = I_frame
70
71  #while success:
72  while total_frames <= 4:
73      success, P_frame = video.read()
74      P_frame = cv2.cvtColor(P_frame, cv2.COLOR_BGR2GRAY)
75
76      print('read a new frame:', success)
77      total_frames += 1
78      image_difference = abs(P_frame - prev_frame)
79
80      cv2.imshow("Image-Difference {}".format(total_frames),image_difference)
81      cv2.waitKey(0)
82
83      img_diff_prediction = []
84      # roll each row as X=A
85      for i in range(height):
86          img_diff_prediction.append(np.roll(image_difference[i],1))
87          # setting first value of each row 0
88          img_diff_prediction[i][0] = 0
89
90      # calculate the error image
91      error_image = image_difference-img_diff_prediction
92
93      cv2.imshow("Error-Image {}".format(total_frames),error_image)
94      cv2.waitKey(0)
```

Τέλος, ακριβώς με τον ίδιο τρόπο που περιεγράφηκε για το I-frame, υπολογίζονται οι πιθανότητες των εικονοστοιχείων της εικόνας σφάλματος και κωδικοποιούνται μέσω Huffman. Το αποτέλεσμα αποθηκεύεται στο ίδιο αρχείο κειμένου bitstring.txt.
Για την ευκολία της άσκησης γίνεται ο υπολογισμός των πρώτων 5 P-frames αλλά εάν κάνουμε uncomment την ακριβώς από πάνω εντολή (σειρά 71 στον κώδικα) γίνεται υπολογισμός όλων των P-frames του βίντεο.

```
96      # get probabilities of unique values in the l-frame difference
97      uniq, counts = np.unique(error_image,return_counts=True)
98
99      # convert probabilities into dictionary
100     probabilities = dict(np.asarray((uniq,counts)).T)
101     # sort dictionary by ascending order
102     probabilities = {k: v for k, v in sorted(probabilities.items(), key=lambda item: item[1])}
103     # generate huffman code
104     codes = huffman.encode(probabilities)
105     # scan error image and convert each value to it's huffman code
106     bitstring = ''
107     for i in range(len(error_image)):
108         for j in range(len(error_image[i])):
109             bitstring += codes[error_image[i][j]]
110
111     f = (open('bitstring.txt', 'a'))
112     # 1 frame will not add " - " only p frames will write " - " before the bitstring
113     f.write(" - " + bitstring + " | " + str(codes))
114     f.close
115
116
117     prev_frame = P_frame
```

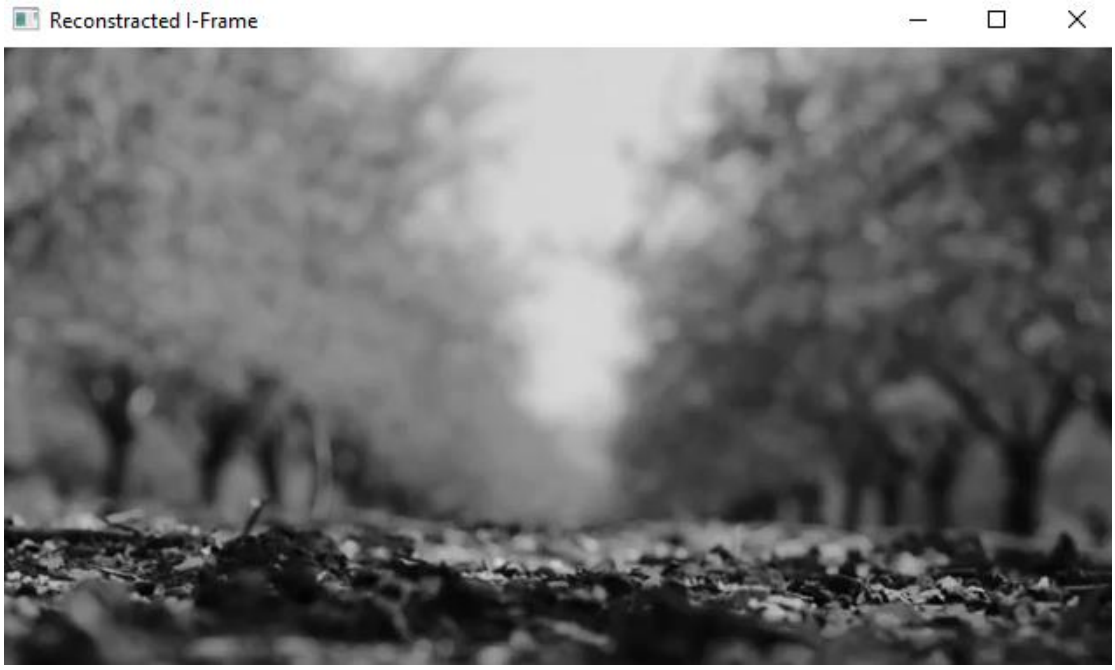
Περιγραφή αρχείου decoder.py

Ο αποκωδικοποιητής αρχικά διαβάζει τα bits του όλων των πλαισίων καθώς και τα αντίστοιχα λεξικά τους, από το αρχείο bitstring.txt και στη συνέχεια τα δίνει σαν όρισμα στην συνάρτηση huffman.decoder(). Το αρχείο huffman.py επεξεργάζεται αρχικά τα bits και επιστρέφει σε λίστα τις εικόνες σφάλματων πρόβλεψης.

Ο αποκωδικοποιητής παίρνει στην συνέχεια τη λίστα με τις εικόνες σφαλμάτων πρόβλεψης και αφού τις μετατρέψει από μονοδιάστατη λίστα σε λίστα ύψους x πλάτους (όπου ύψος και πλάτος των αρχικών πλαισίων), ανακατασκευάζει μέσω του προβλέπτη X=A, το αρχικό I-frame και τις εικόνες διαφορών των P-frame.

Για την ανακατασκευή του I-frame έχει δημιουργηθεί ένα for loop το οποίο αναζητεί την επόμενη τιμή για κάθε στοιχείο μέσα στη συγκεκριμένη σειρά της λίστας και την προσθέτει στην αμέσως προηγούμενη τιμή, ακριβώς όπως ο προβλέπτης X=A.

Το ανακατασκευασμένο I-frame χωρίς απώλειες μπορεί να φανεί παρακάτω:



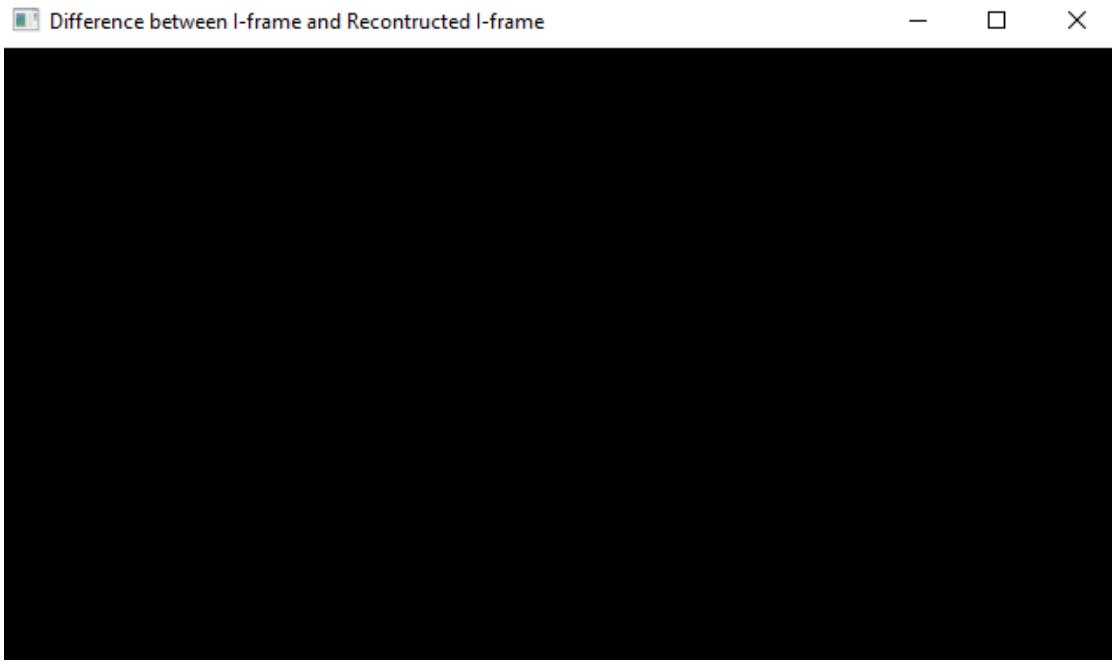
Καθώς και ο κώδικας ανακατασκευής.

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

    # reshape each error-image into 720x1280 matrix
    img_reshape = np.array(list_error_images[i])
    img_reshape = img_reshape.reshape(height,width)

    # first bitstring represents I_frame
    if i == 0:
        print("Reconstructing I-Frame")
        for j in range(len(img_reshape)):

            # get the first value of the image difference
            value = img_reshape[j][0]
            frame.append(value)
            pointer = 0

            for k in range(1,len(img_reshape[j])):

                # calculate the next values of the image difference
                next = img_reshape[j][k]
                next_value = value + next
                frame.append(next_value)
                value = next_value
                pointer += 1

        frame = np.array(frame)
        frame = frame.reshape(height,width)
        reconstructed_I_frame = frame

        cv2.imshow("Reconstracted I-Frame", frame.astype(np.uint8))
        cv2.waitKey(0)

        prev_frame = frame
        frame = []
```

Για την ανακατασκευή των P-frames γίνεται η ίδια διαδικασία με την διαφορά ότι προστίθεται στο τέλος το προηγούμενο ανακατασκευασμένο πλαίσιο.

```
    # next bitstrings represent P_frames
    else:
        # - get the error image
        # - calculate the original image difference using X=A predictor
        # - add the image difference with the previous frame

        print("Recontructing P-Frame {}".format(i))
        for j in range(len(img_reshape)):

            # get the first value of the image difference
            value = img_reshape[j][0]
            frame.append(value)
            pointer = 0

            for k in range(1,len(img_reshape[j])):

                # calculate the next values of the image difference
                next = img_reshape[j][k]
                next_value = value + next
                frame.append(next_value)
                value = next_value
                pointer += 1

        frame = np.array(frame)
        frame = frame.reshape(height,width)

        P_frame = frame + prev_frame

        prev_frame = P_frame
        frame = []

        cv2.imshow("Reconstructed P-Frame {}".format(i), P_frame.astype(np.uint8))
        cv2.waitKey(0)

print("Finished Deconding")
input()
```

Τέλος, για την απόδειξη ότι η συμπίεση και η ανακατασκευή είναι χωρίς απώλειες, μετά το μήνυμα ότι η αποκωδικοποίηση τελείωσε εάν πατηθεί enter, εμφανίζεται η διαφορά μεταξύ αρχικού I-frame – ανακατασκευασμένου I-frame. Το αποτέλεσμα εμφανίζεται παρακάτω και είναι μια εικόνα μηδενικής εντροπίας άρα η διαδικασία είναι χωρίς απώλειες.




```
video = cv2.VideoCapture('walk.mp4')
success, image = video.read()

# first frame is the I-frame
I_frame = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

I_frame_diff = I_frame - reconstructed_I_frame

cv2.imshow("Difference between I-frame and Recontructed I-frame", I_frame_diff.astype(np.uint8))
cv2.waitKey(0)
```

Περιγραφή αρχείου huffman.py

Σε αυτό το αρχείο γίνεται η κωδικοποίηση Huffman, δημιουργούνται δηλαδή κωδικές λέξεις των μοναδικά εμφανιζόμενων εικονοστοιχείων σε ένα πλαίσιο. Αρχικά θα γίνει ανάλυση του κωδικοποιητή Huffman:

Ο κωδικοποιητής είναι μια συνάρτηση (encode()) που παίρνει σαν όρισμα ένα λεξικό που αποτελείται από κλειδιά: τις μοναδικές τιμές εικονοστοιχείων και τιμές: τον αριθμό συχνότητας τους μέσα στο πλαίσιο (πιθανότητες εμφάνισης). Ο κωδικοποιητής, εισάγει τις τιμές σε ένα σωρό, όπου όλες οι μοναδικές τιμές των εικονοστοιχείων εισάγονται σαν κόμβοι που τελικά καταλήγουν φύλλα του δένδρου και ύστερα για όσο είναι το μήκος του σωρού δημιουργεί το δένδρο Huffman ως εξής:

Γίνονται εξαγωγές από τον σωρό και ανά δύο εξαγωγές η πρώτη θεωρείται αριστερό κλαδί και παίρνει την τιμή 0 ενώ η δεύτερη θεωρείται δεξί κλαδί και παίρνει την τιμή 1. Εφόσον έχουμε ταξινομήσει τα εικονοστοιχεία σε αύξουσα σειρά, βάσει την πιθανότητα εμφάνισης.

Ύστερα δημιουργείται ένας νέος κόμβος με πιθανότητα ίση με το άθροισμα των δύο προηγούμενων και η διαδικασία επαναλαμβάνεται όσο υπάρχει έστω

και ένας κόμβος.

```python
# node of the huffman tree
class node:
    def __init__(self, prob, symbol, left=None, right=None):
        self.prob = prob

        self.symbol = symbol

        self.left = left

        self.right = right

        # gets values 0 or 1
        self.huff = ''

    def __lt__(self, nxt):
        return self.prob < nxt.prob
```

```python
# creates huffman tree
# takes a dictionary as a parameter, which contains all the unique symbols with their frequency values
def encode(probabilities):

    nodes=[]

    # push data into a heap
    for x in range(len(probabilities)):
        heapq.heappush(nodes, node(probabilities[list(probabilities.keys())[x]], list(probabilities.keys())[x]))

    # create huffman tree with given data
    while len(nodes)>1:
        left = heapq.heappop(nodes)
        right = heapq.heappop(nodes)

        left.huff = 0
        right.huff = 1

        # connect the 2 smallest nodes
        new_node = node(left.prob + right.prob, left.symbol+right.symbol, left, right)

        heapq.heappush(nodes, new_node)

    return all_nodes(nodes[0])
```

Το αποτέλεσμα, δίνεται σε μια συνάρτηση που ονομάζεται all_nodes() η οποία υπολογίζει αναζητώντας στο δένδρο, ένα λεξικό με κλειδί την τιμή των μοναδικών εικονοστοιχείων και σαν τιμή του κλειδιού την αντίστοιχη κωδική λέξη Huffman. Το λεξικό αυτό επιστρέφεται στον κωδικοποιητή.

```
25  # return huffman codes
26  def all_nodes(node, val=''):
27
28      # read current node huffman code
29      new_val = val + str(node.huff)
30
31      # if it is not a leaf node continue on the next until we reach leaf node
32      if(node.left):
33          all_nodes(node.left, new_val)
34      if(node.right):
35          all_nodes(node.right, new_val)
36
37      # if we find a leaf node print its huffman code
38      if(not node.left and not node.right):
39          codes[node.symbol] = new_val
40
41      return codes
```

Ο αποκωδικοποιητής (decoder()) κάνει την αντίστροφη εργασία. Δέχεται σαν όρισμα τα δυαδικά ψηφία του κάθε πλαισίου και αρχικά τα διαχωρίζει. Στο αρχείο bitstring.txt, Κάθε δυαδική αναπαράσταση ενός πλαισίου διαχωρίζεται με το λεξικό που της αναλογεί με τον χαρακτήρα « | » ενώ κάθε κωδικοποιημένο πλαίσιο διαχωρίζεται με τον χαρακτήρα « - ». Έτσι, ο αποκωδικοποιητής Huffman χωρίζει τα στοιχεία του κάθε πλαισίου και τα εισάγει σε μια λίστα.
Ύστερα, η λίστα αυτή μεταφράζεται από κωδικές λέξεις στις κανονικές τιμές των εικονοστοιχείων, μέσω του λεξικού που προαναφέρθηκε. Το αποτέλεσμα είναι η εικόνα σφάλματος πρόβλεψης.

```
70  def decoder(bitstring):
71      # making a list with the bitstring and the huffman codes of each image-error
72      list_img_diff = []
73      data = [x.split(' | ') for x in str(bitstring).split(' - ')]
74
75      # recreate error-image
76      for i in range(len(data)):
77
78          img_diff = []
79          j = 0
80
81          huffman_bitstring = data[i][0]
82          param_dict = data[i][1]
83          current_bitstring = ''
84
85          # convert string of parameters to dictionary
86          param_dict = ast.literal_eval(param_dict)
87
88          while j < len(data[i][0]):
89
90              if current_bitstring in param_dict.values():
91                  img_diff.append(list(param_dict.keys())[list(param_dict.values()).index(current_bitstring)])
92                  current_bitstring = ''
93              else:
94                  current_bitstring = current_bitstring + huffman_bitstring[j]
95                  j = j + 1
96
97          img_diff.append(0)
98          list_img_diff.append(img_diff)
99
100     return list_img_diff
```

Θέμα 2ο

Εκφώνηση

Θέμα 2 (1.5 βαθμοί): Σε βίντεο της επιλογής σας, διάρκειας 5s – 10s, στο οποίο υπάρχει ήπια κίνηση αντικειμένου και κάμερας, επιλέξτε ένα αντικείμενο και εξαφανίστε το αλγοριθμικά. Δηλαδή, δημιουργήστε και αποθηκεύστε ένα νέο βίντεο στο οποίο δεν θα υπάρχει το αντικείμενο που επιλέξατε. Για τον σκοπό αυτόν, αξιοποιήστε την τεχνική αντιστάθμισης κίνησης. Υλοποιήστε και τεκμηριώστε το σχετικό σύστημα.

Υλοποίηση

Το αρχείο που υλοποιεί αυτό το θέμα είναι το `thema2.py`. Αρχικά φορτώνεται το βίντεο για επεξεργασία. Αρχικοποιούνται κάποιες απαραίτητες μεταβλητές και για όσο διαρκεί το βίντεο αποθηκεύουμε ένα ένα κάθε πλαίσιο. Στην συνέχεια διαιρούμε κάθε ένα από τα πλαίσια του βίντεο σε macroblock διαστάσεων 16*16. Ελέγχουμε τις διαστάσεις του πλαισίου και βρίσκουμε εάν μπορεί να διαιρεθεί η εικόνα σε macroblock μεγέθους 16*16. Σε περίπτωση που δεν διαιρείτε βρίσκουμε τον κοντινότερο ακέραιο που διαιρείται και γεμίζουν τις υπόλοιπες θέσεις με μηδενικά, μαύρα pixel.

```python
import numpy as np
import cv2

def createMacroblocks(frame, size=16):

    # Υπολογίζουμε νεο πλάτος και υψος για να διαιρειτε σε μπλοκ 16*16
    width = frame.shape[0]
    newWidth = (width + size) - (width % size)
    height = frame.shape[1]
    newHeight = (height + size) - (height % size)
    defference = ((0, newWidth - width), (0, newHeight - height), (0, 0))
    newFrame = np.pad(frame, defference, mode='constant')

    # Δημιουργούμε τα macroblock
    macroblocks = []
    for newWidth_ in range(0, newWidth - size, size):
        row = []
        for newHeight_ in range(0, newHeight - size, size):
            macroblock = newFrame[newWidth_:newWidth_ + size, newHeight_:newHeight_ + size]
            row.append(macroblock)
        macroblocks.append(row)

    return macroblocks
```

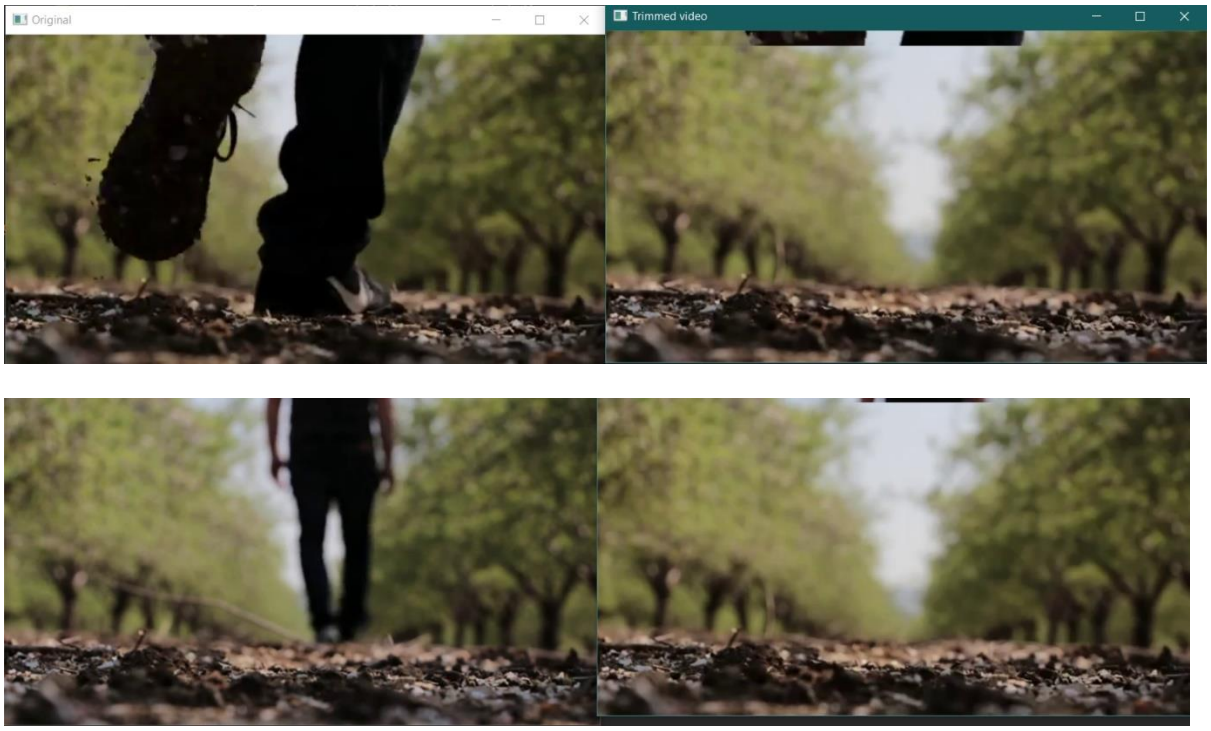
Υποθέτουμε ότι το πρώτο πλαίσιο είναι αποκλειστικά πλαίσιο παρασκηνίου, αυτό διαπιστώνεται και από το default βίντεο που δίνεται για επεξεργασία το

«walk.mp4». Επομένως εξαιρούμε το πρώτο πλαίσιο και αντικαθιστούμε τα μπλοκ κίνησης με τα αντίστοιχα μπλοκ παρασκηνίου από το προηγούμενο πλαίσιο που δεν υπήρχε κίνηση. Με την αντικατάσταση αυτών των μπλοκ τελικά ολόκληρο το αντικείμενο που βρίσκεται σε κίνηση (ο άνθρωπος) εξαφανίζεται και αφού ανακατασκευάσουμε την νέα εικόνα η επεξεργασία έχει ολοκληρωθεί. Εμφανίζουμε το αρχικό και το επεξεργασμένο βίντεο το αποθηκεύουμε με όνομα trim_video.avi

```
25
26  def restoreFrame(macroblocks):
27      # Εννόουμε τα macroblocks για κάθε γραμμή και επιστρέφουμε το καινούργιο πλαίσιο
28      rows = []
29      for row in macroblocks:
30          rows.append(np.concatenate([macroblock for macroblock in row], axis=1))
31
32      frame = np.concatenate([row for row in rows], axis=0)
33      return frame
34
35  # Φορτώνουμε το βίντεο για επεξεργασία
36  video = cv2.VideoCapture('walk.mp4')
37  flag = False
38  previous= None
39  counter = 0
40
41  while video.isOpened():
42      ret, frame = video.read()
43      if not ret:
44          break
45
46      #Φτιάχνουμε macroblocks μεγέθους 16*16
47      Macroblock = createMacroblocks(frame)
48
49      # Εξερούμε το πρωτο πλαισιο.
50      if not flag:
51          flag = True
52          previous = Macroblock
53          continue
```

```
50      if not flag:
51          flag = True
52          previous = Macroblock
53          continue
54
55      # Αντικαθηστούμε τα macroblcoks ωστε να εξαφανισουμε το αντικείμενο.
56      for i in range(1, 22):
57          Macroblock[i] = previous[i]
58
59
60      newFrame = restoreFrame(Macroblock)
61      frameSize = (newFrame.shape[1],newFrame.shape[0])
62
63      if counter==0:
64          out = cv2.VideoWriter('trim_video.avi', cv2.VideoWriter_fourcc(*'XVID'), 20.0, frameSize)
65          counter+=1
66      else:
67          out.write(newFrame)
68
69      cv2.imshow('Original', frame)
70      cv2.imshow('Trimmed video', newFrame)
71
72      # Αποθηκέυουμε το προηγούμενο macroblock
73      previous = Macroblock
74      cv2.waitKey(25)
75
76  video.release()
77  out.release()
78  cv2.destroyAllWindows()
```

Ενδεικτικά screenshot του προγράμματος



Βιβλιογραφία

https://www.youtube.com/watch?v=M284dGA1pmc

Home - OpenCV

NumPy

Parag Havaldar, Gerard Medioni- Συστήματα Πολυμέσων – Αλγόριθμοι, Πρότυπα και Εφαρμογές, Επιμέλεια Ελληνικής έκδοσης Άγγελος Πικράκης, Εκδόσεις Π.Χ.Πσσχαλίδης