Π Α Ν Ε Π Ι Σ Τ Η Μ Ι Ο Π Ε Ι Ρ Α Ι Ω Σ
Τμήμα Πληροφορικής

Εργασία Μαθήματος «Συστήματα Πολυμέσων»

	Εργασία Συστήματα Πολυμέσων
Ονόματα φοιτητών – Αριθμοί Μητρώων	Παναγιώτης Καφαντάρης – Π15054
Ημερομηνία παράδοσης	11/07/2020

Συστήματα Πολυμέσων

ΠΕΡΙΕΧΟΜΕΝΑ

Σκοπός Εργασίας

Η εργασία πραγματεύεται την προγραμματιστική υλοποίηση συμπίεσης αρχείων. Πιο συγκεκριμένα οι ασκήσεις μας δινουν τη δυνατοτητα να ασχοληθούμε με την συμπίεση και εκτίμηση εικόνας-πλαισίου που συναντάμε σε ένα βίντεο. Σχετικά με το προγραμματισμό των παρακάτω που θα αναλύσουμε εξηγώ τα πάντα με πολυ εξονυχιστικά σχόλια.

Προγραμματιστικές ασκήσεις
Μερος Α

Άσκηση 17 (αρχείο exercise_17.py)

ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΕΣ ΑΣΚΗΣΕΙΣ

17. **[ΒΔ07]** Στην άσκηση αυτή, θα υλοποιήσετε την τεχνική της αντιστάθμισης κίνησης και θα μελετήσετε πώς επηρεάζει τα σφάλματα πρόβλεψης. Δείγματα αρχείων βίντεο, μαζί με τη σχετική πληροφορία διαμόρφωσης, παρέχονται στην ενότητα αρχείων προς λήψη του ιστοτόπου www.cengage.com. Εκεί θα βρείτε και κώδικα για την ανάγνωση και προβολή εικόνας. Τροποποιήστε τον κώδικα για να διαβάζει και να προβάλλει πλαίσια βίντεο. Υποθέστε ότι το πρώτο πλαίσιο θα είναι πάντα ένα πλαίσιο Ι και ότι τα υπόλοιπα πλαίσια θα είναι τύπου Ρ. Η υπόθεση αυτή ευσταθεί στην περίπτωση των σύντομων ακολουθιών βίντεο που επεξεργάζεστε, που έχουν μήκος το πολύ 100 πλαισίων.

Από εκφώνηση η πρώτη εργασία, μας ζητά τη δημιουργία πλαισίων μέσα από ένα τυχαίο βίντεο. Για καλύτερα αποτελέσματα έχουμε πάρει βίντεο (τα οποία υπάρχουν μέσα στα αρχεία μας) τα οποία η κάμερα είναι σχετικά στατική και υπάρχει γρηγόρη κίνηση του αντικειμένου αλλά διαλέξαμε εκείνα τα οποία καταγράφουν την κίνηση σε slow motion. Ειδικότερα το αρχείο το οποίο μπορούμε να δούμε τη διαδικασία κατά την οποία μετατρεπουμε το βίντεο σε πλαίσια είναι το **frame_maker.py**. Οπότε σε περίπτωση που θέλει κάποιος να τρέξει τα επόμενα αρχεία τα οποία θα αναλύσουμε για ένα διαφορετικό βίντεο το πρώτο πράγμα που πρέπει να κάνει είναι να ανατρέξει στο frame_maker και να βάλει κάποιο άλλο βίντεο που θέλει εκείνος.

```
dir_path = os.path.dirname(os.path.realpath(__file__))
#pairnoume to video pou theloume
cap = cv2.VideoCapture(dir_path+'/bouncing ball.mov')

try:#o fakelos pou tha apothikeusoume ta frames
    if not os.path.exists(dir_path+'/images/bouncing-ball-frames'):
        os.makedirs(dir_path+'/images/bouncing-ball-frames')
except OSError:
    print ('Error')

currentFrame = 0
while(True):
    #kanoume capture ta frames
    ret, frame = cap.read()
```

Έτσι τα επόμενα δύο ερωτήματα τα οποία θα αναλύσουμε θεωρούμε ως I_πλαίσιο το frame0 και όλα τα υπόλοιπα ως P_πλαίσια.

<u>Επισήμανση</u>

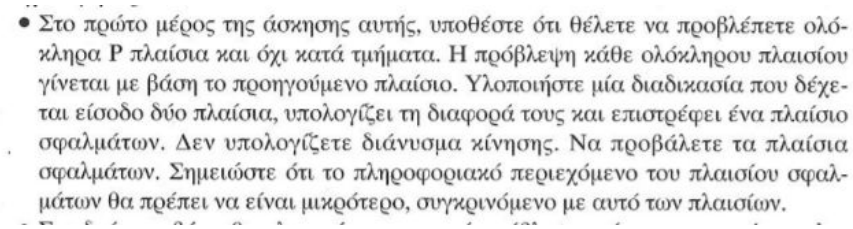
Σε περίπτωση που κάποιος θέλει να τρέξει τα παρακάτω υποερωτήματα για να δει πως λειτουργούν, καλό είναι να μειώσει τον αριθμό επαναλήψεων.

```
#upoerwthma 2 ta idia me prin me kapoies diafores
def Get_Results_MotionEst(i_frame):
    results=[]
    for i in range(1,100):
        path = 'images/bouncing-ball-frames/bouncing-ball'+str(i)+'.jpg'
        p_frame=rgbTogray(path)
        prediction_frame=LogSearch(p_frame,i_frame).ReconstructImage() #problepoume thn eikona

        #meta briskoume th diafora tous
```

Τρέχουμε τον αλγόριθμο για 100 πλαισια ωστόσο για την αντιστάθμιση κίνησης ο χρόνος είναι αρκετά μεγάλος(περίπου 6 λεπτά για να ολοκληρωθούν όλα), οπότε προτείνω να μειωθεί πριν το τρέξετε καθώς το μέγεθος των εικόνων είναι μεγάλο με αποτέλεσμα να υπάρχουν μεγάλες επαναλήψεις.

Υποερώτημα 1 (def Get_Results_ImgDifference(i_frame))



- Στο πρώτο μέρος της άσκησης αυτής, υποθέστε ότι θέλετε να προβλέπετε ολόκληρα P πλαίσια και όχι κατά τμήματα. Η πρόβλεψη κάθε ολόκληρου πλαισίου γίνεται με βάση το προηγούμενο πλαίσιο. Υλοποιήστε μία διαδικασία που δέχεται είσοδο δύο πλαίσια, υπολογίζει τη διαφορά τους και επιστρέφει ένα πλαίσιο σφαλμάτων. Δεν υπολογίζετε διάνυσμα κίνησης. Να προβάλετε τα πλαίσια σφαλμάτων. Σημειώστε ότι το πληροφοριακό περιεχόμενο του πλαισίου σφαλμάτων θα πρέπει να είναι μικρότερο, συγκρινόμενο με αυτό των πλαισίων.

'Οπως είπαμε και πριν θεωρούμε το frame0 ως I_πλαίσιο και τα υπόλοιπα P_πλαίσια, η εργασία ζητά να προβλεφθούν ολόκληρα τα πλαίσια και όχι κατά τμήματα. Έτσι αρχικά βρίσκουμε τη διαφορά τους(**I_frame - P_frame**) και τα κωδικοποιούμε σύμφωνα με την **JPEG** διαδικασία. Ωστόσο πρώτα όμως για να αποφύγουμε τις δύσκολες πράξεις μετατρέπουμε κάθε frame σε gray_scale για να είναι πίνακας 2D(αν αφήναμε τα χρώματα θα ήταν 3D), βρίσκουμε τη διαφορά τους και στη συνέχεια το συμπιέζουμε με τον τρόπο που αναφέραμε και θα αναλύσουμε παρακάτω. Τέλος γυρνάμε πίσω την κωδικοποιημένη λέξη και μετράμε το μήκος της. Αυτή η λέξη είναι η κωδικοποιημένη εντροπία στην οποία όπως θα αναλύσουμε και παρακάτω έχει κωδικοποιηθεί σύμφωνα με τους πίνακες Huffman. Αφού πάρουμε τα μήκη τα γράφουμε σε ένα αρχείο που έχουμε όλα τα αποτελέσματα ώστε να τα εξετάσουμε τη σημασία τους αργότερα. Η είκονα σφάλαματος αποθηκεύεται ξεχωριστά ώστε να σωθεί ως βίντεο στο τέλος της διαδικασίας, για καλύτερη εκτίμηση.

1. **JPEG διαδικασία κωδικοποίηση (def compress(Image)) (αρχειο __init__.py)**

Αρχίζουμε τη διαδικασία μετατρέποντας τις τιμές των εικονοστοιχείων σε float ώστε να έχουμε καλύτερες διαιρέσεις. Στη συνέχεια πολύ σημαντικό είναι να κάνουμε padding ώστε να σιγουρευτούμε ότι η εικόνα που θα εξετάσουμε θα μπορεί να χωριστεί 8 * 8 blocks. Δηλαδή αυτό που κάνουμε είναι να συμπληρώσουμε με μηδενικά τα όρια της εικόνας αν αυτό χρειαστεί για να μπορέσουμε να έχουμε μέγεθος που θα μπορεί να χωριστεί σε 8 * 8. Κατόπιν χωρίζουμε σε τμήματα 8 * 8 την εικόνα προκειμένου να εφαρμόσουμε DCT(Discrete Cosine Transform)

```
def dct2d(arr):
    return dct(dct(arr, norm='ortho', axis=0), norm='ortho', axis=1)
```

και κβάντιση σύμφωνα με τον πίνακα JPEG για κάθε block.

```python
#Quantization
def quantize(block):
    quantization_table = QUANTIZATION_TABLE
    return block / (quantization_table)
```

Όλα με τη βοήθεια του αρχείου utils.py

JPEG πίνακας

```python
#o jpeg quantization pinakas
QUANTIZATION_TABLE = np.array((
    (16, 11, 10, 16, 24, 40, 51, 61),
    (12, 12, 14, 19, 26, 58, 60, 55),
    (14, 13, 16, 24, 40, 57, 69, 56),
    (14, 17, 22, 29, 51, 87, 80, 62),
    (18, 22, 37, 56, 68, 109, 103, 77),
    (24, 36, 55, 64, 81, 104, 113, 92),
    (49, 64, 78, 87, 103, 121, 120, 101),
    (72, 92, 95, 98, 112, 100, 103, 99)
))
```

Επόμενο βήμα είναι να μετατρέψουμε τις τιμές που έχουμε πάρει μετά το τέλος της κβάντισης σε integers δηλαδή στρογγυλοποιημένες τιμές. Από την παραπάνω διαδικασία προκύπτουν οι DC και AC συντελεστές οι οποίοι πρέπει να κωδικοποιηθούν. Γενικοτερα οι DC συντελεστες αντιπροσωπεύουν τη συνολική ενέργεια ενός 8*8 block ενω οι AC τη συχνότητα. Στατιστικα χαμηλές συχνότητες περιέχουν περισσότερη ενέργεια απ οτι μεγαλύτερες συχνότητες. Παρόλα αυτά το οπτικό σύστημα του ανθρώπου είναι πιο ευαίσθητο σε μικρές συχνότητες παρά σε μεγαλύτερες. Μετά την κβάντιση υπάρχει μεγάλη πιθανότητα οι τιμές των μεγάλων συχνοτήτων να είναι μηδέν. Έτσι κωδικοποιούν ξεχωριστά τους δύο αυτούς συντελεστές.

```python
#upoerwtima 1
def Get_Results_ImgDifference(i_frame):
    results=[]
    for i in range(1,100):#se posa frames thelete na epanalhpthei h diadikasia
        path = 'images/bouncing-ball-frames/bouncing-ball'+str(i)+'.jpg'
        p_frame=rgbTogray(path) #metatrepoume se gray-scale
        frame_diff = cv2.absdiff(p_frame,i_frame) #pairnoume th diafora tous
        compressed = compress(frame_diff) #sumpiezoume

        #apothikeuoume to apotelesma ths eikonas
        saving_img_path='images/differencial-img/'
        cv2.imwrite(os.path.join(saving_img_path , 'Error-img-'+str(i)+'.jpg'), frame_diff)

        #apothikeuoume to mhkos ths entropias ths
        results.append(str(len(compressed['data'])))
        print(str(i)+'%')
    #to grafoume sto arxeio
    with open("Results-Diff.txt", "w") as text_file:
        text_file.write("%s\n" % "Entropy length for image difference(1-100):")
        for i in results:
            text_file.write("%s\n" % i)
        text_file.close()
```

(αρχεία για παρακάτω συντελεστές EntropyEncoding.py)

1.1 DC συντελεστες και DPCM

Οι συντελεστές αυτοί είναι οι μεγαλύτερες τιμές κάθε block 8 * 8 (μετά από DCT και κβάντιση) και πολλές φορές είναι πολύ κοντά και στα προηγούμενα(blocks). Εφαρμόζουμε DPCM(differential pulse code modulation) κωδικοποίηση πού είναι η διαφορά μεταξύ του τρέχον και του προηγούμενου block. Αυτό που θέλουμε είναι να μικρύνουμε τον αριθμό γιατί μικρότερος αριθμός σημαίνει λιγότερα bits.

```python
def DC_enc(block):
    #upologizoume th diafora twn DC me ta prohgoumena block kai gurname to pinaka pou dhmiourgoume
    return (
        (item - block[i - 1]) if i else item
        for i, item in enumerate(block)
    )
```

Στη συνέχεια με αυτή τη πληροφορία και ενός πίνακα huffman (ο οποίος έχει μέσα τα range τα οποία είναι η διαφορά των DC) θα επιστρέψουμε την αντίστοιχη κωδικοποιηση σε bits. Αυτή είναι η κωδικοποίηση της εντροπίας η οποία θα αναλυθεί στη συνέχεια

1.2 AC συντελεστες, RLE και zig-zag

Οι συντελεστές αυτοι είναι οι υπόλοιποι 63 στο block απ τα 64(αυτό το ένα είναι το DC). Η σκέψη πίσω απ τη κωδικοποίηση του είναι οτι η πληροφορία ίσως είναι συγκεντρωμένη σε λιγότερο χαμηλές συχνότητες και οι υψηλές είναι πιθανό να είναι μηδέν. Διασχίζουμε όλες αυτές τις τιμές του με μέθοδο **zig-zag** προκειμένου από ένα πίνακα δύο διαστάσεων να έχουμε πλέον

μία λίστα. Ο λόγος για τον οποίο το κάνω αυτό είναι να δούμε πόσα μηδενικά έχει το block που θα συμβαλλει στη συμπιεση καθώς δεν μας ενδιαφέρουν τα περιττά bits. Έπειτα εφαρμόζουμε **κωδικοποίηση μήκους διαδρομής(RLE)**.

```
def GET_AC(self):
    #upologizoume ta AC me length length kwdikopoihsh
    self.AC_lengthLength = []

    #gia kathe block trexoume zig zag algorithmo gia ola ta stoixeia tou AC
    for block in self.data:
        self.AC_lengthLength.extend(
            AC_RunLength_enc(tuple(ZigZag_Algorithm(block))[1:])
        )
```

1.3 Zig-zag

Ελέγχουμε αρχικά αν η μήτρα είναι τετράγωνη τότε μπορούμε να συνεχίσουμε. Γυρνάμε τη συνάρτηση ως μία λίστα με τα values του block, η οποια τρεχει μεσα στο block ως εξης:

1)‘Οταν βρεθεί τα όρια του block να αλλάξει κατεύθυνση και αν προέρχεται από περιττό εικονοστοιχείο τότε πρόσθεσε +1 στο x αλλιώς στο y

2)Ενώ αν θέλουμε να κινηθεί διαγώνια τότε του λέμε οτι θα πρέπει να βρίσκει ίδιας μορφής με αυτό εικονοστοιχεία που είναι κοντά του και δεν έχει επισκεφθεί. Δηλαδη αν βρισκομαστε σε περριτά εικονοστοιχεία τότε θα πρέπει να συνεχίσει να κινείται στα περιττά και αντίστοιχα για τα ζυγά. Αυτό δημιουργήθηκε μέσω παρατήρησεις των εικονοστοιχείων για τη διαδρομή που θέλουμε να κάνουμε κάθε φορά.

****περιττο ή αρτιο εικονοστοιχειο λεμε το αθροισμα των συντεταγμενων τους**

1.4 Κωδικοποίηση Μήκους διαδρομης(RLE)

Η κωδικοποίηση μήκους διαδρομής έχει ως βασικό σκοπό να ομαδοποιήσει τις τιμές των εικονοστοιχείων με βάση το μήκος, δηλαδή αν η τιμή εμφανιστεί 63 φορές τότε θα είναι της μορφής (63,0). Έτσι έχουμε δύο περιπτώσεις:

1. να υπάρχουν τα μηδενικά στο τέλος μέσα στη λίστα η οποία έχει προκύψει απο zig zag.
2. είτε θα είναι ενδιάμεσα, παλι απο zig zag.

Έτσι μπορούμε να κωδικοποιηθούν τις δύο αυτές περιπτώσεις χρησιμοποιώντας δύο special σύμβολα την ZRL(zero run length)=’11110000’=(15,0) και την EOB(end of block)=’00000000’=(0,0)

Η EOB μας δείχνει ότι τα εναπομείναντα στοιχεία σε μια zig zag λιστα ειναι μηδεν. Η ZRL αντιπροσωπεύει το μήκος διαδρομής με 15 μηδενικούς συντελεστές ακολουθούμενοι από έναν συντελεστή μηδενικού ευρους. Δηλαδή τη κωδικοποίηση 16 μηδενικών. Διαδρομές με μήκος

μικρότερο από 16 κωδικοποιούνται με μη μηδενικούς AC συντελεστές απο τη zig zag λίστα. Η κωδικοποίηση της εντροπίας μέσω πινάκων huffman για τους AC θα δούμε και παρακάτω.

1.5 Κωδικοποίηση εντροπίας των DC και AC μέσω Huffman

Από παραπάνω έχοντας βρει τους συντελεστές είμαστε έτοιμη να τους κωδικοποιήσουμε βάση των πινάκων huffman. Η γενικότερη λογική είναι ότι θα ανατρέχουμε στο πίνακα ώστε με βάση τη τιμή να δούμε σε ποιά κωδικοποίηση αντιστοιχεί. Ο τρόπος με τον οποίο θα ξεχωρίζουμε τους δύο συντελεστές μέσα στο κώδικα κυρίως, είναι οτι ο πρώτος αντιπροσωπεύει αριθμό ενώ ο δεύτερος λίστα.

```
def encode(self):
    '''
    h kwdikopoihsh twn DC kai AC opws eipame kai parapanw ginetai me bash to
    jpeg huffmana pinaka. Edw gurname ena dictionary me to duadikh roh twn DC:01 kai AC:01
    '''
    entropy_enc = {}
    '''pairnoume th duadikh roh twn DC,AC mesw huffman ap tis listes pou
       proekupsan sthn arxikopoihsh'''
    #mas sumferei na einai string
    #edw afou exoume parei prwta ta prapanw kaloume th sunarthsh pou tha ta kwdikopoihsh kata huffman
    entropy_enc[DC] = ''.join(encode_huffman(v)
                              for v in self.diff_dc)
    entropy_enc[AC] = ''.join(encode_huffman(v)
                              for v in self.lengthLength_ac)
    return entropy_enc
```

1.5.1 Entropy DC

Το πρώτο πράγμα που κοιτάζουμε πριν ξεκινήσουμε τη κωδικοποίηση είναι να δούμε αν η τιμή είναι μέσα στο διάστημα (-2048,2048), εφόσον είναι ψάχνουμε σε ποιό range βρίσκεται η διαφορά του DC με βάση τον πίνακα huffman με τις κατηγορίες. Σύμφωνα με αυτόν βρίσκουμε τη θέση του στοιχείου και γυρνάμε το αντίστοιχο size και κωδικό. Ο κωδικός που γυρίσαμε θα μπει στο τέλος της λέξης αφού όμως σύμφωνα με το size που βρήκαμε ανατρέξουμε στο πίνακα με τις κωδικές λέξεις για DC. Εκεί βρίσκουμε ποιός κωδικός αντιστοιχεί και τέλος τον ενώνουμε με τον αρχικό κωδικό.

```
#elegxoume an einai lista, dld tupou collection.iterable
if not isinstance(value, collections.Iterable):  # afou den einai ara einai DC
    if value <= -2048 or value >= 2048:
        raise ValueError(
            'uparxei error giati to DC einai para polu megalo'
        )
    #briskoume th thesh tou value mesa sto pinaka
    size, code = get_HuffmanRange(HUFFMAN_CATEGORIES, value)

    #me bash th thesh gurname thn antistoixh kwdikopoihsh
    if size == 0:#an einai to prwto dld value=0
        return HUFFMAN_CATEGORY_CODEWORD[DC][size]

    return (HUFFMAN_CATEGORY_CODEWORD[DC][size]
            + '{:0{padding}b}'.format(code, padding=size))
```

Για παράδειγμα αν η διαφορά των DC που βρήκαμε είναι -8=0111 τότε το size επιστρέφει 4 και στον άλλο πίνακα το 4 περιγράφεται ως 101 άρα η κωδικοποίηση του DC(ο συντελεστής, οχι η διαφορά του) είναι 1010111

Οι Πίνακες με τις κατηγορίες και τους κωδικούς

```
HUFFMAN_CATEGORIES = (
    (0, ),
    (-1, 1),
    (-3, -2, 2, 3),
    (*range(-7, -4 + 1), *range(4, 7 + 1)),
    (*range(-15, -8 + 1), *range(8, 15 + 1)),
    (*range(-31, -16 + 1), *range(16, 31 + 1)),
    (*range(-63, -32 + 1), *range(32, 63 + 1)),
    (*range(-127, -64 + 1), *range(64, 127 + 1)),
    (*range(-255, -128 + 1), *range(128, 255 + 1)),
    (*range(-511, -256 + 1), *range(256, 511 + 1)),
    (*range(-1023, -512 + 1), *range(512, 1023 + 1)),
    (*range(-2047, -1024 + 1), *range(1024, 2047 + 1)),
    (*range(-4095, -2048 + 1), *range(2048, 4095 + 1)),
    (*range(-8191, -4096 + 1), *range(4096, 8191 + 1)),
    (*range(-16383, -8192 + 1), *range(8192, 16383 + 1)),
    (*range(-32767, -16384 + 1), *range(16384, 32767 + 1))
)
```

```
HUFFMAN_CATEGORY_CODEWORD = {
    DC: bidict({
            0:  '00',
            1:  '010',
            2:  '011',
            3:  '100',
            4:  '101',
            5:  '110',
            6:  '1110',
            7:  '11110',
            8:  '111110',
            9:  '1111110',
            10: '11111110',
            11: '111111110'
        }),
```

1.5.2 Entropy AC

Οι συντελεστές αυτοί κωδικοποιούνται ως ζευγάρια το μέγεθος και η τιμή. Οπως εχουμε αναφερει παραπανω, αντιπροσωπεύουν μία λίστα από το πόσες φορές εμφανίζονται οι τιμές. Έχει γίνει μια αρχική απλοποίηση τους μέσω του αλγορίθμου μήκους διαδρομής χρησιμοποιώντας τα σπέσιαλ σύμβολα(EOB,ZRL). Ακριβώς με πριν ανατρέχουμε στο πίνακα με τις κατηγορίες. Βρίσκουμε τη τιμή αυτή μέσω του κλειδιού (δηλαδή της τιμής **u (-1024,1024)** που μετρήσαμε πόσες φορές-length εμφανίστηκε) και παίρνουμε το size και τον κωδικό. Το size μαζί με το μήκος-length είναι αυτά που θα μας υποδείξουν σε ποιό σημείο του πίνακα πρέπει να κοιτάμε. Βρίσκουμε τη κωδική λέξη και προσθέτουμε στο τέλος τον πρώτο κωδικό απ το πίνακα

με τις κατηγορίες. Αν φυσικά η τιμή στην αρχή είναι ίση είτε με EOB είτε με ZRL τότε πηγαίνουμε απευθείας στο πίνακα με τις κωδικές λέξεις και αποφεύγουμε τις κατηγορίες

```
else:   # afou einai lista einai AC
    value = tuple(value)
    if value == EOB or value == ZRL: #an to AC einai kapoio ap ta duo special sumbola gurname ameswsw th kwdikopoihsh
        return HUFFMAN_CATEGORY_CODEWORD[AC][value]

    length, key = value
    if key == 0 or key <= -1024 or key >= 1024:
        raise ValueError(
            'error: to AC anhkei mono sto diasthma (-1024,0)u(0,1024)'
        )
    #antistoixa me DC
    size, code = get_HuffmanRange(HUFFMAN_CATEGORIES, key)
    return (HUFFMAN_CATEGORY_CODEWORD[AC][(length, size)]
            + '{:0{padding}b}'.format(code, padding=size))
```

```
AC: bidict({
        EOB: '1010',  #(0, 0)
        ZRL: '11111111001',  #(15, 0)

        (0, 1):  '00',
        (0, 2):  '01',
        (0, 3):  '100',
        (0, 4):  '1011',
        (0, 5):  '11010',
        (0, 6):  '1111000',
        (0, 7):  '11111000',
        (0, 8):  '1111110110',
        (0, 9):  '1111111110000010',
        (0, 10): '1111111110000011',

        (1, 1):  '1100',
        (1, 2):  '11011',
        (1, 3):  '1111001',
        (1, 4):  '111110110',
        (1, 5):  '11111110110',
        (1, 6):  '1111111110000100',
        (1, 7):  '1111111110000101',
        (1, 8):  '1111111110000110',
        (1, 9):  '1111111110000111',
        (1, 10): '1111111110001000',

        (2, 1):  '11100',
        (2, 2):  '11111001',
        (2, 3):  '1111110111',
        (2, 4):  '111111110100',
```

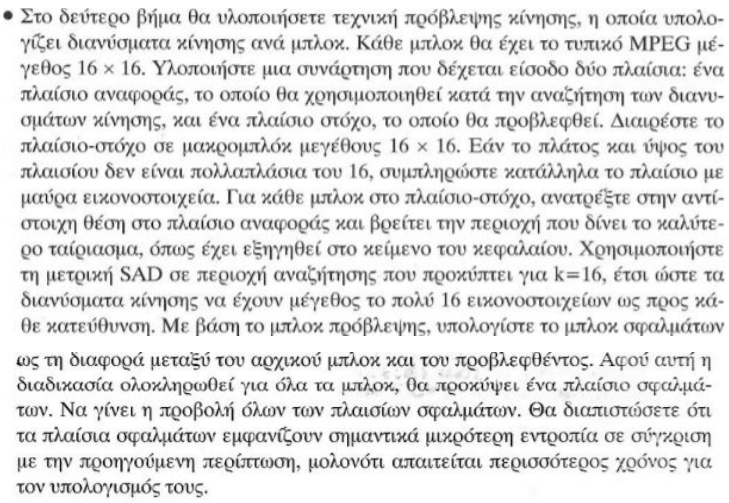
Ο πίνακας κωδικών για AC

<u>Επισήμανση</u>
Δε πρέπει να ξεχνάμε οτι όλα αυτά τρέχουν για κάθε block, άρα όλα είναι επαναληπτικές διαδικασίες που θα εκτελεστούν πολλές φορές.

Κωδικοποίηση τελειώνει και πλέον μπορούμε να έχουμε το μήκος της εντροπίας για να το συγκρίνουμε με το επόμενο ερώτημα

Οι εικόνες αποθηκεύονται στο **images/differencial-img
Η εντροπία στο **Results-Diff.txt

Υποερώτημα 2 (def Get_Results_MotionEst(i_frame))



- Στο δεύτερο βήμα θα υλοποιήσετε τεχνική πρόβλεψης κίνησης, η οποία υπολογίζει διανύσματα κίνησης ανά μπλοκ. Κάθε μπλοκ θα έχει το τυπικό MPEG μέγεθος 16 × 16. Υλοποιήστε μια συνάρτηση που δέχεται είσοδο δύο πλαίσια: ένα πλαίσιο αναφοράς, το οποίο θα χρησιμοποιηθεί κατά την αναζήτηση των διανυσμάτων κίνησης, και ένα πλαίσιο στόχο, το οποίο θα προβλεφθεί. Διαιρέστε το πλαίσιο-στόχο σε μακρομπλόκ μεγέθους 16 × 16. Εάν το πλάτος και ύψος του πλαισίου δεν είναι πολλαπλάσια του 16, συμπληρώστε κατάλληλα το πλαίσιο με μαύρα εικονοστοιχεία. Για κάθε μπλοκ στο πλαίσιο-στόχο, ανατρέξτε στην αντίστοιχη θέση στο πλαίσιο αναφοράς και βρείτει την περιοχή που δίνει το καλύτερο ταίριασμα, όπως έχει εξηγηθεί στο κείμενο του κεφαλαίου. Χρησιμοποιήστε τη μετρική SAD σε περιοχή αναζήτησης που προκύπτει για k=16, έτσι ώστε τα διανύσματα κίνησης να έχουν μέγεθος το πολύ 16 εικονοστοιχείων ως προς κάθε κατεύθυνση. Με βάση το μπλοκ πρόβλεψης, υπολογίστε το μπλοκ σφαλμάτων ως τη διαφορά μεταξύ του αρχικού μπλοκ και του προβλεφθέντος. Αφού αυτή η διαδικασία ολοκληρωθεί για όλα τα μπλοκ, θα προκύψει ένα πλαίσιο σφαλμάτων. Να γίνει η προβολή όλων των πλαισίων σφαλμάτων. Θα διαπιστώσετε ότι τα πλαίσια σφαλμάτων εμφανίζουν σημαντικά μικρότερη εντροπία σε σύγκριση με την προηγούμενη περίπτωση, μολονότι απαιτείται περισσότερος χρόνος για τον υπολογισμός τους.

Η άσκηση ζητά την εκτίμηση του πλαισίου στόχου βαση των I,P frames ώστε να συγκρίνουμε τη συμπίεση της εικόνας σφάλματος σε σχέση με τη πρώτη περίπτωση. Η λογική είναι ότι θα διαιρέσουμε την εικονα σε 16*16 μακρο μπλοκ στη συνέχεια θα βρούμε τα διανύσματα τους το οποίο μεταφράζεται στο που μεταφέρθηκε το μακρο μπλοκ αυτό στην επόμενη εικόνα. Αυτό που κάνουμε για να το βρούμε είναι μία **λογαριθμική αναζήτηση** οπου έχουμε θέσει ως ένα **step=8**. Φυσικά και εδώ έχει γίνει μετατροπή σε gray scale πριν γίνει η εκτίμηση.

```
for i in range(1,100):
    path = 'images/bouncing-ball-frames/bouncing-ball'+str(i)+'.jpg'
    p_frame=rgbTogray(path)
    prediction_frame=LogSearch(p_frame,i_frame).ReconstructImage() #problepoume thn eikona

    #meta briskoume th diafora tous
    frame_diff = cv2.absdiff(prediction_frame,p_frame)
    compressed = compress(frame_diff) #meta sumpiezoume
```

2 Λογαριθμική αναζήτηση (LogarithmicSearch.py)

Αρχίζουμε την αναζήτηση μας στο διάστημα x,y **u (-8,8)**, εκεί προσπαθούμε να βρούμε κάθε φορά το καλύτερο ταίριασμα με το αντίστοιχο makroblock στο προηγούμενο frame και αυτό επιτυγχάνεται χρησιμοποιώντας την **SAD**(sum absolute difference) μετρική.

```
step = self.step

while(stop==0):

    min_value = self.SAD(n,m)

    #logarithmikh anazhthsh, sugkriseis twn 9 eikonostoixeiwn
    #   #   #

    #   #   #

    #   #   #
    for i in range(n-step,n+step+1,step):
        if(i==n): #sugkrinoume prwta auta pou einai panw katw aristera kai dexia tou kentrou
            for j in range(m-step,m+step+1,step):
                value = self.SAD(i,j)
                [n,m,value,min_value]=self.findMinLocation(n,m,i,j,value,min_value)
        else: #gia ta plagia
            value = self.SAD(i,m)
            temp=m
            [n,m,value,min_value]=self.findMinLocation(n,m,i,temp,value,min_value)
    #exoume brei tis nees suntetagmenes
```

2.1 SAD

Η διαδικασία που ακολουθούμε είναι να αφαιρέσουμε τις τιμές των εικονοστοιχείων των makroblock και να τις προσθέσουμε σε ένα συνολικό άθροισμα, δηλαδή εξετάζουμε ποιά είναι τα μάκρο μπλοκ αυτά που έχουν το καλύτερο ταίριασμα στο διαστημα που αναφέραμε παραπάνω και όπως θα δούμε στη συνέχεια αυτό μικραίνει αναλόγως τα ευρήματα.

```
def SAD(self,n,m): #sum absolute value
    sum=0.0
    target_picture=None
    target_picture=self.target_picture

    y, x = self.position(n,m) #briskoume tis suntetagmenes tou

    #epeidh to padding einai problhma, exoume ena exupno tropo na elegxoume ta oria
    #an xefugei tote prosthetoume th pio megalh timh wste na sigoureutoume oti de tha e
    if x>=len(target_picture[0]) or x<0 or y>=len(target_picture) or  y<0:
        sum =sum + sys.float_info.max/10
    else: #an einai entos oriwn prosthetoume apla thn apoluth diafora tous
        sum = sum + np.abs(self._macroblock(n,m,isTarget=False)-self._macroblock(n,m))
    return sum
```

Βάση των αποτελεσμάτων της μετρικής αυτής διαλέγουμε τη μικρότερη τιμή από τη σύγκριση των 9 makroblock. Όμως η τιμή του step αλλάζει ως εξής:

Αν η μικρότερη τιμή είναι το ίδιο το κέντρο τότε διαιρούμε το βήμα με το 2 και ξανακάνουμε αναζήτηση σε μικρότερο διάστημα πλέον

Αν δεν είναι το κέντρο τότε μεταφερόμαστε στο νέο μακρομπλοκ και επαναλαμβάνουμε την αναζήτηση με το τρέχον βήμα

Αν το βήμα γίνει 1 τότε κάνουμε μια τελευταία φορά αναζήτηση στα 9 μακρομπλοκ σε διάστημα x,y u (-1,1) και βλέπουμε ποιό έχει τη μικρότερη τιμή SAD

```
if([n,m] == [n_start,m_start]): #an einai to
    step=int(step/2)
if([n,m]!=[n_start,m_start]): #an den einai t
    n_start=n
    m_start=m
if(step ==1): #an to step ginei 1 stamatame
    stop=1
```

Τέλος γυρνάμε το διάνυσμα που βρήκαμε και στη συνέχεια τα τοποθετούμε σε μια μεγάλη λίστα για να ανακατασκευάσουμε την εικόνα.

Για την ανακασκευή της εικόνας(**def ReconstructImage()**) προκειμένου να εχουμε πιο γρηγορους υπολογισμους χρησιμοποιουμε μονο τα διανυσματα τα οποια δεν ειναι μηδεν. Καθώς διάνυσμα μηδέν σημαίνει για μας οτι το μακρομπλοκ δε κινήθηκε, επομένως θα είναι στην ίδια θέση με τη προηγούμενη εικόνα. Έτσι ασχολούμαστε μόνο με τα μετακινηθέντα και με βάση τη referenced frame τοποθετούμε τα εικονοστοιχεία των μακρομπλοκ στις νέες τους θέσεις.

```
makblocks_Vectors = self.motionEstimation() #briskoume ta dianusmata

for y_ in range(y_macroblocs):
    for x_ in range(x_macroblocs):
        x = x_ * self.n
        y = y_ * self.n
        offset_y = makblocks_Vectors[y_][x_][1] #ta dianusmata
        offset_x = makblocks_Vectors[y_][x_][0]
        if((offset_y!=0 or offset_x!=0) and remove==False): #gia na elaxistopoihsoume to xrono, elegxoume poia den
            for n1 in range(self.n):    #na asxolhthoume me ta mhdenika kathws shmainei oti den allaxan thesh
                for m1 in range(self.n):
                    ey = y + n1 #oi nees suntetagmenes
                    ex = x + m1

                    #se periptwsh pou bgei ektos oriwn
                    if ey < 0 or len(referenced_picture) <= ey or ex < 0 or len(referenced_picture[0]) <= ex:
                        continue
                    cy = y - offset_y #oi suntetagmenes sto prohgoumeno plaisio
                    cx = x - offset_x

                    #se periptwsh pou bgei ektos oriwn
                    if cy < 0 or len(referenced_picture) <= cy or cx < 0 or len(referenced_picture[0]) <= cx:
                        continue
                    #anakataskeuasoume to plaisio
                    target_picture[helper-ey][ex] =referenced_picture[helper-cy][cx]
```

Για την εικόνα σφάλματος αφαιρούμε στη συνέχεια την ανακατασκευασμένη εικόνα που βρήκαμε με την τρέχον προκειμένου να δούμε πόσο μακριά έπεσε η πρόβλεψη μας. Στη συνέχεια συμπιέζουμε συμφωνα με τη jpeg διαδικασία και βλέπουμε το μήκος της εντροπίας το προκύπτει να είναι αρκετά μικρότερο από τη προηγούμενη περίπτωση καθώς όπως βλέπουμε και στην εικόνα σφάλματος η διαφορά είναι σχεδόν μηδαμινή και αυτό μεταφράζεται και στην

εντροπία. Συγκεκριμένα είναι κοντά στα 10.000 και παραπάνω bits το οποίο σημαίνει ότι παρόλο που κάνουμε περισσότερη διαδικασία να το βρούμε η συμπίεση μας θα είναι καλύτερη.

Οι εικόνες αποθηκεύονται στο **images/MotionEst-img
Τα μήκη με τις εντροπίες στο **Results-MotionEst.txt

Ασκηση 18 (exercise_18.py)

18. **[ΒΔ08]** Σε αυτήν την άσκηση θα δείτε ότι η τεχνική της τμηματικής πρόβλεψης με βάση την αντιστάθμιση κίνησης, μπορεί επίσης να χρησιμοποιηθεί σε εφαρμογές εκτός συμπίεσης. Μία τέτοια ενδιαφέρουσα εφαρμογή είναι η απομάκρυνση αντικειμένων ή προσώπων από τη ροή του βίντεο. Για παράδειγμα, έστω βίντεο στο οποίο η κάμερα δεν έχει κινηθεί και το παρασκήνιο είναι σχετικά στατικό, αλλά κινούνται ορισμένα αντικείμενα στο προσκήνιο. Στόχος σας είναι να προσεγγίσετε το αντικείμενο χρησιμοποιώντας μπλοκ και στη συνέχεια να αντικαταστήσετε αυτά τα μπλοκ με παρασκήνιο, σαν να μην ήταν ποτέ παρόν το αντικείμενο. Στη γενική περίπτωση, η λύση είναι πολύ δύσκολη, αλλά στο πλαίσιο αυτής της άσκησης θα επεξεργαστείτε ορισμένες απλούστερες ιδέες. Κατά την υλοποίηση, βεβαιωθείτε ότι μπορείτε να χειρίζεστε το μέγεθος του μπλοκ ως παράμετρο, προκειμένου να ελέγξετε πόσο καλά λειτουργεί ο αλγόριθμος απομάκρυνσης αντικειμένων για διάφορα μεγέθη μακρομπλόκ.

Σκοπός αυτού του ερωτήματος είναι η αφαίρεση του αντικειμένου από τα διάφορα frames που προέκυψαν απ το βίντεο. Τα αντικείμενα θέλουμε να χάσουν τη θέση τους και να αντικατασταθούν απ το background. Για την επίλυση του ζητήματος θα χρησιμοποιηθεί πάλι το ίδιο αρχείο με το οποίο κάναμε λογαριθμικη αναζήτηση και βρήκαμε τα διανύσματα. Η ίδια λογική θα εφαρμοστεί όπως στο motion estimation με ελάχιστες διαφορές.

Υποερώτημα 1 (def remove_object(i_frame)

- Κατ' αρχήν, φορτώστε ένα σύνολο πλαισίων βίντεο. Υποθέστε ότι το πρώτο πλαίσιο είναι αποκλειστικά πλαίσιο παρασκηνίου και δεν περιέχει αντικείμενα σε κίνηση. Δοθέντος ενός πλαισίου, n, προχωρήστε στη διαίρεσή του σε μπλοκ. Υπολογίστε ένα διάνυσμα κίνησης ανά μπλοκ με βάση το προηγούμενο πλαίσιο (αναφοράς). Για τα μπλοκ παρασκηνίου, θα πρέπει να προκύψουν μηδενικά (ή σχεδόν μηδενικά) διανύσματα κίνησης, γιατί η κάμερα ήταν στατική. Για τα μπλοκ των αντικειμένων σε κίνηση, θα πρέπει να προκύψουν μη μηδενικά διανύσματα κίνησης. Παρακολουθήστε όλα αυτά τα διανύσματα κίνησης. Μπορείτε ακόμη και να τα απεικονίσετε για κάθε μπλοκ, καθώς προβάλετε το βίντεο.

Από προηγούμενη άσκηση έχουμε δείξει πώς ακριβώς φορτώνουμε το σύνολο των frames ενος βίντεο οπότε δεν χρειάζεται να εξηγηθεί. Στη δικιά μας περίπτωση δεν χρειαζόμαστε να έχουμε ως πρώτο πλαίσιο κάποιο το οποίο στο background του δεν έχει κάποιο αντικείμενο διότι η λύση που θα περιγράψουμε επιλύει την άσκηση χωρίς αυτή την

παράμετρο. Όπως ακριβώς και πριν που περιγράψαμε το motion estimation διαιρούμε το πλαίσιο σε block δίνοντας ωστόσο τη δυνατότητα στο χρήστη να μπορεί να επηρεάσει το μέγεθος του block το οποίο έχει τεθεί ισο με 16, παρόλα αυτά μπορεί να παρέμβει και να αλλάξει το νούμερο.

```
prediction_frame=LogSearch(p_frame,i_frame).ReconstructImage(remove=True)
#an thelete na allaxete to megethos tou block prepei na grapsete LogSearch(p_frame,i_frame,n=enas arithmos).Reconst...
```

Ξανά επαναλαμβάνοντας τη διαδικασία αντιστάθμισης κίνησης για να ανακατασκευάσουμε την εικόνα χωρίς το αντικειμενο με βάση target και referenced frame και εξετάζουμε ποιά διανύσματα είναι μη μηδενικά και ποιά όχι. Ωστόσο σε αυτή την περίπτωση δεν κρίνεται αναγκαίο να δειχθεί κάτι καθώς αυτή την πληροφορία θα την χρησιμοποιήσουμε για να απαντήσουμε στο δεύτερο υποερώτημα.

Υποερώτημα 2(ιδια συναρτηση)

- Στη συνέχεια, βρείτε τα μπλοκ που αντιστοιχούν σε μη μηδενικά διανύσματα κίνησης. Αντικαταστήστε κάθε τέτοιο μπλοκ με το αντίστοιχο μπλοκ παρασκηνίου. Τα εικονοστοιχεία παρασκηνίου θα πρέπει να προέρχονται από προηγούμενο πλαίσιο, στο οποίο δεν υπήρχε αντικείμενο σε κίνηση. Η αντικατάσταση όλων αυτών των μπλοκ θα οδηγήσει στην απομάκρυνση των αντικειμένων, διότι τα αντικείμενα αντικαθίστανται από το παρασκήνιο. Επαναλάβατε τη διαδικασία, χρησιμοποιώντας μακρομπλόκ διαφορετικού μεγέθους.

Εξετάζουμε κάθε φορά αν το διάνυσμα είναι μηδενικό ή όχι. Σε περίπτωση που δεν είναι μηδέν κάτι που μεταφράζεται ως ότι υπάρχει κίνηση δηλαδή βρήκαμε το αντικείμενο. Άρα μπορούμε να τα αντικαταστήσουμε με τα μπλοκ παρασκηνίου μέσω της reference frame.

Δε χρειάζεται η πρώτη εικόνα να μην έχει αντικείμενο καθως παίρνουμε το τελευταίο block με διάνυσμα μηδεν. Αυτό σημαίνει ότι όταν εμείς θα βρεθούμε κοντά, όταν εμείς θα βρεθούμε σε ένα εικονοστοιχείο που το διάνυσμα του δεν είναι μηδέν δηλαδή το αντικείμενο μας, το αμέσως προηγούμενο το οποίο βρέθηκε ως μηδενικό θα είναι πολύ κοντά στο αντικείμενό μας και επομένως δεν θα επηρεάσει, δεν θα υπάρχει μεγάλη διαφορά στο φωτισμό του background και θα είναι αρκετά κοντά στο πρωτότυπο background.

```
makblocks_Vectors = self.motionEstimation() #briskoume ta dianusmata

for y_ in range(y_macroblocs):
    for x_ in range(x_macroblocs):
        x = x_ * self.n
        y = y_ * self.n
        offset_y = makblocks_Vectors[y_][x_][1] #ta dianusmata
        offset_x = makblocks_Vectors[y_][x_][0]
        if((offset_y!=0 or offset_x!=0) and remove==False): #gia na elaxistopoihsoume to xrono, elegxoume poia den
            for n1 in range(self.n):    #na asxolhthoume me ta mhdenika kathws shmainei oti den allaxan thesh
                for m1 in range(self.n):
                    ey = y + n1 #oi nees suntetagmenes
                    ex = x + m1

                    #se periptwsh pou bgei ektos oriwn
                    if ey < 0 or len(referenced_picture) <= ey or ex < 0 or len(referenced_picture[0]) <= ex:
                        continue
                    cy = y - offset_y #oi suntetagmenes sto prohgoumeno plaisio
                    cx = x - offset_x

                    #se periptwsh pou bgei ektos oriwn
                    if cy < 0 or len(referenced_picture) <= cy or cx < 0 or len(referenced_picture[0]) <= cx:
                        continue
                    #anakataskeuasoume to plaisio
                    target_picture[helper-ey][ex] =referenced_picture[helper-cy][cx]
```

Έχουμε επαναλάβει αρκετές φορές διαδικασία και φαίνεται ότι block μεγέθους λιγότερο από 16 δίνουν μεγαλύτερη λεπτομέρεια αλλά ίσως χάνουν σε ποσότητα ενώ μπλοκ μεγαλύτερο από 16 κερδίζουν σε ποσότητα αλλά χάνουν σε λεπτομέρεια.

Οι εικόνες αποθηκεύονται στο **images/Img-with-no-obj

Υποερωτημα 3

- Είναι πιθανό να αντιμετωπίσετε ασυνέχειες και παρενέργειες στα όρια των μπλοκ που αντικαθίστανται. Πώς θα ελεχιστοποιήσετε αυτά τα φαινόμενα;

Φυσικά την ώρα που τρέχουμε τον αλγόριθμο εντοπίζονται πολλές ασυνέχειες, υπάρχουν κάποια blocks τα οποία έχουν διαπιστωθεί με διάνυσμα μηδέν ενώ ειναι μερος του αντικειμένου. Επίσης στα όριά του βλέπουμε ότι επειδή το block μπορεί να είναι μεγαλύτερο να να καλύψει περισσότερη επιφάνεια από ότι πρέπει.

Για να ελαχιστοποιήσουμε τις παρενέργειες αυτές πρέπει αρχικά να μειώσουμε το μέγεθος του block προκειμένου να έχουμε καλύτερη λεπτομέρεια. Ωστόσο θα χάσουμε σε ποσότητα γιατί ο καλύτερος τρόπος δεν είναι να κάνουμε λογαριθμική αναζήτηση με βάση ένα προηγούμενο frame. Λόγω χρόνου δεν εγινε δυνατο να εφαρμοστεί αυτή η λύση. Το καλύτερο δυνατό αποτέλεσμα θα το έδινε μία διαδικασία η οποία θα εκτιμούσε κάθε φορά ένα frame με βάση ένα μεγαλύτερο ευρος από frames, αυτό θα συνέβαινε γιατί. αν το αντικείμενο κινηθει ελάχιστα τότε είναι πολύ πιθανό κάποια μακρομπλοκ να εμφανίσουν μηδενικα διανυσματα ενω δε θα πρεπε. Εχοντας μία μεγαλύτερη διαφορά όμως στην διαδρομή του αντικειμένου αυτό θα δώσει και μεγαλύτερα διανύσματα και πιο διακριτά. Εχει σαν αποτέλεσμα να εκμηδενιστεί το

λάθος το οποίο εμφανίζονται αυτα τα μηδενικα διανυσματα που αποτελούν μέρος αντικειμένου. Ταυτόχρονα όπως είπαμε και πριν το block θα έχει μειωμένη διάσταση και αρα θα μπορούμε να πάρουμε μεγαλύτερη λεπτομέρεια.

Φυσικά μεγαλύτερη ακόμη επιτυχία θα μπορούσε να υλοποιηθεί μην έχοντας μόνο τα προηγούμενα frames αλλά και κάποια από τα επόμενα έτσι ώστε να ξέρουμε με σιγουριά την φωτεινότητα του background γιατί αν το αντικείμενο πιθανότατα περάσει από μία γραμμή στην οποία αλλάζει αρκετά η φωτεινότητα θα δυσκολευτεί να να εφαρμόσει σωστά τον αλγόριθμο και θα εμφανισει μεγάλη ασυνέχεια όπου ο φωτισμός δεν είναι αυτός που πρέπει.

Όλο αυτό το εγχείρημα θα απαιτούσε να ξέρουμε σε κάθε επανάληψη πόσο μεγάλο είναι το σφάλμα. Ώστε αν αρχίζει η τιμή αυτή να μεγαλώνει και να ξεπερνά μια συγκεκριμένη τιμή που θα θέσουμε εμείς τότε να θέτουμε καινούργιο referenced frame(I frame). Κάτι τέτοιο θα μείωνε δραματικά τις επαναλήψεις και άρα και το χρόνο εκτέλεσης.

Υποερώτημα 4

Η προτεινόμενη λύση θα λειτουργήσει ικανοποιητικά, μόνο όταν υποθέσετε ότι η κάμερα δεν κινείται, ότι τα αντικείμενα που κινούνται υπόκεινται σε λεία κίνηση και ότι δεν παρατηρούνται αλλαγές κατεύθυνσης. Όμως, στη γενική περίπτωση, η κάμερα κινείται, τα αντικείμενα κινούνται με όχι αυστηρό τρόπο και επιπλέον, παρατηρούνται αλλαγές στον φωτισμό, καθώς η κάμερα κινείται.

- Αλλά ας υποθέσουμε ότι μπορείτε να εντοπίσετε όλα τα μακρομπλόκ ενός πλαισίου που αντιστοιχούν στο παρασκήνιο. Πώς μπορείτε να αξιοποιήσετε αυτό το γεγονός, πλέον της χρήσης αντιστάθμισης κίνησης σε επίπεδο μακρομπλόκ;

Από τη στιγμή που μπορούμε σε μία τόσο δύσκολη κατάσταση να εντοπίσουμε όλα τα μακρομπλοκ παρασκηνίου μέσω της αντιστάθμισης κίνησης τότε έχουμε τη δυνατότητα να χειριστούμε όπως εμείς νομίζουμε το παρασκήνιο. Τέτοιες τεχνικές εφαρμόζονται σε διάφορα βίντεο όπου υπάρχει green screen πίσω από κάτι το οποίο κινείται. Λέμε στον υπολογιστή αν εντοπίζεις μόνο πράσινα μακρομπλοκ διαχώρισε το από τα αντικείμενα τα οποία κινούνται. Αυτό έχει σκοπό να μπορούμε να αλλάζουμε το παρασκήνιο από πίσω και να δίνουμε την εντύπωση στο θεατή ότι το εκάστοτε background είναι μέρος του βίντεο. Έτσι σε τελική ανάλυση όταν γνωρίζουμε όλα τα μακρομπλοκ του παρασκηνίου μπορούμε πολύ εύκολα να το αλλάξουμε και γενικότερα να το χειριστούμε όπως εμείς πιστεύουμε δημιουργώντας την ψευδαίσθηση ότι είναι μέρος του βιντεο πχ. Οτι ένας άνθρωπος βρίσκεται σε μια τοποθεσία ενώ δεν είναι στη πραγματικότητα

Μέρος Β

Άσκηση (exercise_merosB.py)

Συστήματα Πολυμέσων – Εργασία 2020 – Μέρος Β

- Έστω ομάδα N φοιτητών που έχει δηλωθεί στο μάθημα (1<=N<=3).
- Έστω ότι A_i οι αντίστοιχοι πενταψήφιοι αριθμοί μητρώου των φοιτητών αυτών.
- Κατασκευάστε συνθετική εικόνα διαστάσεων 104 γραμμών x 200 στηλών ως εξής: Κάθε γραμμή σχηματίζεται από πεντάδες αριθμών και κάθε πεντάδα είναι ένας A_i που επιλέγεται κάθε φορά τυχαία. Επίσης, σε κάθε πεντάδα, προτού αυτή ενσωματωθεί στην εικόνα, επιλέγεται τυχαία ένα ψηφίο και αντικαθίσταται με το 5.
- Αφού κατασκευάσετε την εικόνα, διαιρέστε τη σε macroblock 8x8 και σε κάθε macroblock εφαρμόστε τη διαδικασία που περιγράφεται στο Σχήμα 7-7 (σελ. 226) του βιβλίου.

Η συγκεκριμένη εργασία είναι ατομική οπότε δεν έχουν δηλωθεί άλλα άτομα στο μάθημα. Αυτό έχει σαν αποτέλεσμα να υπάρχει μόνο ένα μητρώο το οποίο θα μπει την εικόνα μας. έχουμε τοποθετήσει λοιπόν το μητρώο σε μία λιστα προκειμένου να βάλουμε κάθε φορά τα νούμερα ξεχωριστά στη γραμμή ως εξης:

Εικονοστοιχεία 1 2 3 4 5 6 7 ...
Τιμες 1 5 0 5 4 1 5 ...

```
#ftiaxnoume thn eikona
j=0
for r in range(0,104):
    for c in range(0,200):
        A_img[r][c]=int(row[j])
        j=j+1
        if(j==5):
            j=0
```

Πρώτο πράγμα που κάνουμε είναι να θέσουμε διαστάσεις εικόνας και στη συνέχεια να αντικαταστήσουμε έναν αριθμό τυχαία με το 5. Κατόπιν τοποθετούμε τους αριθμούς αυτούς στην εικόνα και τη κατασκευάζουμε. στη συνέχεια της συμπιέζουμε

```
#sumpiezoume th B sumfwna me to sxhma 7.7
B_img=compress(A_img,withExtract=True)
```

κάνοντας DCT και κβάντιση

```
#exoume xwrisei thn eikona se block kai gia kathe block kanoyme ta parakatw
for i, block in enumerate(data):
    #kanoume DCT
    data[i] = dct2d(block)

    #Quantization me bash ton pinaka JPEG gia grey scale
    data[i] = quantize(data[i])
```

και στη συνέχεια κάνουμε μία αποσυμπίεση κάνοντας αντίστροφο κβάντιση και αντίστροφο DCT σύμφωνα με το σχήμα 7.7(__init__.py).

```
else: #gia askhsh meros B
    size=[nrows, ncols]
    for i, block in enumerate(data):
        # Inverse Quantization
        data[i] = dequantize(block)

        # inverse DCT
        data[i] = idct2d(data[i])
```

Ερωτημα 1

Ερώτημα 1
Ποιος είναι ο μέσος λόγος συμπίεσης της εικόνας που προκύπτει συνενώνοντας τα macroblocks των κβαντισμένων DCT συντελεστών (έστω εικόνα B) σε σχέση με την αρχική εικόνα (έστω εικόνα A), αν η διαδικασία κατασκευής της εικόνας A επαναληφθεί 100 φορές. Χρησιμοποιήστε μόνο κωδικοποίηση Huffman για τις εικόνες A και B και όχι άλλες μεθόδους όπως, π.χ., Κωδικοποίηση Μήκους Διαδρομής.

Για να βρούμε το λόγο συμπίεσης της εικόνα και να το εξετάσουμε προϋποθέτει πρώτα να κωδικοποιήσουμε τις εικόνες σύμφωνα με τα δέντρα Huffman. Θεωρούμε ως εικόνα A την αρχικά κατασκευασμένη εικόνα με τα μητρώα και ως εικόνα B, την εικόνα A που έχει όμως υποστεί τη διαδικασία που περιγράφεται στο σχήμα 7.7. Δηλαδή κάνουμε DCT -> κβάντιση και μετά αντίστροφο DCT-> αντίστροφη κβάντιση και στο τέλος συγκεντρώνουμε (**block_combine**) τα blocks και δημιουργούμε τη νέα εικόνα.

```
#sunduazei mia list apo blocks (m * n) se pinaka(m,n)
def block_combine(arr, nrows, ncols):
    if arr.size != nrows * ncols:
        raise ValueError(f'The size of arr ({arr.size}) should be equal to '
                         f'nrows * ncols ({nrows} * {ncols})')

    _, block_nrows, block_ncols = arr.shape

    return (arr.reshape(nrows // block_nrows, -1, block_nrows, block_ncols)
            .swapaxes(1, 2)
            .reshape(nrows, ncols))
```

Συνέχεια προκειμένου να εξετάσουμε το μέσο λόγο συμπίεσης κωδικοποιούμε πρώτα την εικόνα B και μετά την εικόνα A και τα bits που θα προκύψουν τα διαιρούμε μεταξύ τους για να βρούμε το λόγο. Τα βάζουμε σε πίνακα προκειμένου όταν τελειώσει η επανάληψη να πάρουμε το άθροισμά τους και να το διαιρέσουμε με το εκατό καθώς θέλουμε να επαναληφθεί η διαδικασία τόσες φορές

```
    compression_ratio = (A_bits/B_bits)
    print(A_bits)
    print(B_bits)
    print('Compression_ratio einai ',compression_ratio)
    results.append(compression_ratio)
mean_compress_ration=np.sum(results)/100
print('O mesos logos sumpieshs einai: ',mean_compress_ration)
```

3. Κωδικοποίηση σύμφωνα με το δέντρο huffman (HuffmanTree.py)

3.1 Δημιουργία δέντρου(def tree())

```
root_node = tree(probabilities)
```

Το πρώτο πράγμα που κάνουμε είναι να φτιάξουμε το δέντρο huffman το οποίο θα προκύψει ως εξής:

Δημιουργούμε πρώτα τα φύλλα τα οποία είναι οι πιθανότητες το πόσες φορές προκύπτει μία τιμή σε μία εικόνα. Αυτό το έχουμε μετρήσει από πιο πριν δημιουργώντας ένα histogram όπου μετράμε πόσες φορές εμφανίστηκε μία τιμή και τη διαιρούμε με το συνολικό πλήθος.

```
hist = np.bincount(B_img.ravel(),minlength=256)
probabilities = hist/np.sum(hist) #diairontas me to
```

Έχοντας το πλήθος και τις πιθανότητες κάθε τιμής δημιουργούμε τα φύλλα και τα βάζουμε σε μία λίστα FIFO(first in first out) η οποία θα κατατάξει αρχικά τις πιθανότητες σε αυξουσα σειρα

και στη συνέχεια θα δημιουργήσουμε δεσμούς μέχρι να φτάσουμε στη ρίζα. Αυτό θα επιτευχθεί παίρνοντας κάθε φορά το ζευγάρι των πιθανοτήτων με τις μικρότερες πιθανότητες. Μόλις τις βρούμε τις προσθέτουμε και τις συνδέουμε σε έναν καινούργιο κόμβο ο οποίος έχει το άθροισμά τους. Επαναλαμβάνουμε τη διαδικασία μέχρι να φτάσουμε στη ρίζα όπου θα έχει πιθανότητα 1.

```
def tree(probabilities):
    prq=queue.PriorityQueue() #apotelei mia lista pou k
    #me thn upshloterh proteraiothta dld to fullo me to
    #einai mia FIFO (first-in-first-out)
    for i,probabilities in enumerate(probabilities):
        leaf=Node() #ftiaxnoume to fyllo
        leaf.data=i
        leaf.prob=probabilities
        prq.put(leaf) #pairname to fullo sth parapanw l

    while(prq.qsize()>1): #otan exoume brei ta duo full
        n_node=Node()   #ayta me tis mikroteres pithano
        l=prq.get()     #aristero fullo
        r=prq.get()     #dexi fullo

        n_node.left=l
        n_node.right=r
        n_prob=l.prob+r.prob #prosthetoume tis pithanot
        n_node.prob=n_prob  #kai etsi bazoume sto kaino
        prq.put(n_node) #to prosthetoume sth lista
    #h diadikasia epanalambanetai mexri na ftasoume sth
    return prq.get() #gurname pisw th lista pou ftiaxam
```

3.2 Ανάτρεξη δέντρου(def huffman_traversal())

Αφού δημιουργήσουμε το δέντρο τώρα πρέπει να το κωδικοποιήσουμε δηλαδή να το ανατρέξουμε. Η συγκεκριμένη περίπτωση δεν μας ενδιαφέρει αφού θέλουμε το λόγο συμπίεσης, δεν μας ενδιαφέρει πόσα μηδενικά ή πόσους άσσους έχουμε αλλά μας ενδιαφέρουν πόσα bits οπότε δεν έχουμε πίνακα όπου βάζουμε το 0 και το 1, κρατάμε μόνο το συνολικό άθροισμα των bits το οποίο είναι που θα γυρίσουμε.

Έτσι κάθε φορά λέμε στον αλγόριθμο που αρχικά θα βρίσκεται στη ρίζα οτι κάθε φορά που θα συναντά ένα αριστερό κόμβο και εφόσον υπάρχει πρόσθεσε ένα bit και μετακινήσου εκεί. Αν δεν υπάρχει αριστερός αλλά υπάρχει δεξιός κόμβος τότε πρόσθεσε ένα bit και μετακινήσου εκεί. Η διαδικασία θα επαναλμβάνετε μέχρι να μην υπάρχει άλλος κομβος. Αν δεν υπάρχει κανένας τότε απλά γυρνάμε το αποτέλεσμα με το bits που βρήκαμε.

```
#anatrexoume to dentro
def huffman_traversal(root_node):
    if(root_node.left is not None): #an exei aristero desmo
        huffman_traversal.count+=1 #+1 mia thesh bit
        huffman_traversal(root_node.left) #metakinhsou sta aristera
        huffman_traversal.count-=1 #an den uparxei afairese ena kathws to exou
    if (root_node.right is not None): #an exei dexi desmo
        huffman_traversal.count+=1 #+1 bit
        huffman_traversal(root_node.right) #metakinhsou dexia
        huffman_traversal.count-=1
    else: #an ftasame se fullo
        #apothikeuoume to mexri twra apotelesma
        huffman_traversal.output_bits[root_node.data]=huffman_traversal.count

    return
```

Λεπτομερειες για τους φακελους

exercise_17.py= ασκηση 17
exercise_18.py= ασκηση 18
exercise_merosB.py= ασκηση στο μερος Β
Frame_maker.py= φτιαχνοουμε τα frames απ το βιντεο

Results-Diff=υποερωτημα 1 μηκος εντροπιων
Results-MotionEst=υποερωτημα 2 μηκος εντροπιων
Results-mean-compression-ratio=μεσος λογος συμπιεσης

images/

bouncing-ball-frames= τα frames που προεκυψαν απ το βιντεο μας
differencial-img=οι εικονες σφαλματων απο το υποερωτημα 1 της ασκησης 17
MotionEst-img=οι εικονες σφαλματων απο το υποερωτημα 2 της ασκησης 17
Img-with-no-obj=αφαιρεση αντικειμενου στην ασκηση 18

videos/

Image_difference=βιντεο εικονων σφαλματος υποερωτημα 1
Motion-Estimation=βιντεο εικονων σφαλματος υποερωτημα 2
video-with-no-obj=βιντεο αφαιρεσης αντικειμενου ασκηση 18

JPEG_COMPRESSION/

__init__.py=χρησιμοποιειται κυριως στην αρχη της συμπιεσης
utils.py= βοηθητικες συναρτησεις για τη συπμιεση
EntropyEncoding.py=κωδικοποιηση εντροπιας
HuffmanTree.py=κωδικοποιηση με δεντρο huffman

MOTION ESTIMATION/

LogarithmicSearch.py=λογαριθμικη αναζητηση