videoencframewkYCrCb420mc

March 9, 2017

1 VideoencframewkYCrCb420mc

1.0.1 Program to capture video from a camera and store it in an recording file, in Python txt format, using cPickle

1.0.2 Also, frame is divided into luminance and two color components. By using 4:2:0 scheme, color components are then downsampled by the factor of 2. To reduce aliasing artifacts pyramidial filter is used.

1.0.3 With motion estimation and display of motion vectors

1.0.4 This is a framework for a simple video encoder to build.

1.0.5 It writes into file 'videorecord.txt'

```
- Gerald Schuller, May 2015
```

- **Import relevant modules:**

```
In [1]: import numpy as np
        import cv,cv2
        import cPickle as pickle
        import scipy.signal
```

- **Define the variables required:**

```
In [2]: cap = cv2.VideoCapture(0)
        N=2
        g=open('videorecord.txt', 'w')
        # filter is created by convolving two rectangular filters
        #Triangular filter kernel:
        filt1=np.ones((N,N))/N;
        filt2=scipy.signal.convolve2d(filt1,filt1)/N
```

- **Extract size of the fram from the capture:**

```
In [3]: [retval, frame] = cap.read()
        [rows,columns,d] = frame.shape
        print(rows,columns)
```

```
(480, 640)
```

- **Grid of 8x8 blocks:**

```
In [4]: gc = np.zeros((1,columns))
        gc[0,0:columns:8] = np.ones(columns/8)
        gr = np.zeros((rows,1))
        gr[0:rows:8,0] = np.ones(rows/8)
        grid = np.ones((rows,1))*gc + gr*np.ones((1,columns))
        grid3 = np.zeros((rows,columns,3))
        grid3[:,:,1] = grid
```

- **Previous Y frame:**

```
In [5]: Yprev = np.zeros((rows, columns))
```

- **Vectors for current frame as graphic:**

```
In [6]: framevectors = np.zeros((rows, columns, 3))
```

- **Motion vectors, for each block a 2-d vector:**

```
In [7]: mv = np.zeros((rows/8, columns/8, 2))
```

- **Processing 25 frames, computing YCbCr components, then applying filter and donsampling followed by motion estimation using block-wise approach:**

```
In [8]: for n in range(25):
            print("Frame no ",n)
            ret, frame = cap.read()
            [rows, columns, c] = frame.shape

            if ret == True:

                #Here goes the processing to reduce data...
                reduced = np.zeros((rows,columns,c))
                Y = (0.114*frame[:,:,0] + 0.587*frame[:,:,1] + 0.299*frame[:,:,2]);

                Cb = (0.4997*frame[:,:,0] - 0.33107*frame[:,:,1] - 0.16864*frame[:,:,2]);

                Cr = (-0.081282*frame[:,:,0] - 0.418531*frame[:,:,1] + 0.499813*frame[:,:,2]);
                reduced[:,:,0]=Y
                reduced[:,:,1]=Cb
                reduced[:,:,2]=Cr

                #print(grid3.shape)
                #print(framevectors.shape)
```

```
cv2.imshow('Original',frame/255.0+framevectors)
#cv2.imshow('Luminanz Y',Y[0:200,0:100]/255)
#cv2.imshow('Unterabgetastetes Y',Ds)
#cv2.imshow('Farbkomponente U',np.abs(Cr))
#cv2.imshow('Farbkomponente V',np.abs(Cb))


#Two color components are filtered first
Crfilt = scipy.signal.convolve2d(Cr,filt2,mode='same')
Cbfilt = scipy.signal.convolve2d(Cb,filt2,mode='same')

#Downsampling
DCr = Crfilt[0::N,::N];
DCb = Cbfilt[0::N,::N];
#cv2.imshow('Crfiltered',DCr)
#cv2.imshow('Cbfiltered',DCb)

#Motion estimation, correlate current Y block with previous 16x16 block which co
#Start pixel for block wise motion estimation:
block = np.array([8,8])
framevectors = np.zeros((rows,columns,3))
#for loops for the blocks:
#print("for loops for the motion vectors:")

for yblock in range(5):
    #print("yblock=",yblock)
    block[0] = yblock * 8 + 200
    for xblock in range(5):
        #print("xblock=",xblock)
        block[1] = xblock * 8 + 300
        #print("block= ", block)
        #current block:
        Yc = Y[block[0]:block[0] + 8, block[1]:block[1] + 8]
        #print("Yc= ",Yc)
        #previous block:
        #Yp=Yprev[block[0]-4 : block[0]+12 ,block[1]-4 : block[1]+12]
        #print("Yp= ",Yp)
        #correlate both to find motion vector
        #print("Yp=",Yp)
        #print(Yc.shape)
        #Some high value for MAE for initialization:
        bestmae = 10000.0;
        #For loops for the motion vector, full search at +-8 integer pixels:
        for ymv in range(-8,8):
            for xmv in range(-8,8):
                diff = Yc - Yprev[block[0]+ymv : block[0]+ymv+8, block[1]+xmv: b
                mae = sum(sum(np.abs(diff)))/64
                if mae < bestmae:
```

```
                                    bestmae = mae
                                    mv[yblock, xblock, 0] = ymv
                                    mv[yblock, xblock, 1] = xmv

                            #Ycorr=scipy.signal.correlate2d(Yp, Yc,mode='valid')
                            #print("Ycorr= ", Ycorr)
                            #motion vector:
                            #1-d Index of maximum
                            #index1d=np.argmax(Ycorr)
                            #print("inedx1d=",index1d)
                            #convert it to 2d index:
                            #index2d=np.unravel_index(index1d,(9,9))
                            #print("arg max of correlation: ", index2d)
                            #2-d index minus center coordinates (4,4) is the motion vector
                            #print("mv=",mv[0,0,:])
                            #print(np.subtract(index2d,(4,4)))
                            #mv[0,0,:]=np.subtract(index2d,(4,4))
                            #print("mv[0,0,:]=",mv[0,0,:])
                            #print(tuple(np.add(block,mv[0,0,:]).astype(int)))
                            #cv2.line(framevectors, block, (block[0]+mv[0],block[1]+mv[1]),(1.0,1.0,
                            cv2.line(framevectors, (block[1], block[0]), (block[1]+mv[yblock,yblock,
                Yprev=Y.copy();
                #converting  images back to integer:
                Y=np.array(Y, dtype='uint8')
                DCr=np.array(DCr, dtype='int8')
                DCb=np.array(DCb, dtype='int8')
                #"Serialize" the captured video frame (convert it to a string)
                #using pickle, and write/append it to file g:
                pickle.dump(Y,g)
                pickle.dump(DCr,g)
                pickle.dump(DCb,g)

                if cv2.waitKey(1) & 0xFF == ord('q'):
                    break
            else:
                break
```

```
('Frame no ', 0)
('Frame no ', 1)
('Frame no ', 2)
('Frame no ', 3)
('Frame no ', 4)
('Frame no ', 5)
('Frame no ', 6)
('Frame no ', 7)
('Frame no ', 8)
('Frame no ', 9)
('Frame no ', 10)
```

```
('Frame no ', 11)
('Frame no ', 12)
('Frame no ', 13)
('Frame no ', 14)
('Frame no ', 15)
('Frame no ', 16)
('Frame no ', 17)
('Frame no ', 18)
('Frame no ', 19)
('Frame no ', 20)
('Frame no ', 21)
('Frame no ', 22)
('Frame no ', 23)
('Frame no ', 24)
```

- **Release when finished with motion estimation and compensation:**

```
In [9]: cap.release()
        g.close()
        cv2.destroyAllWindows()
```