CV2 Examples

February 3, 2017

1 Program 1 - horizortsfreq

Program to generate a horizontal spatial frequency, store it in a file, and display it on the screen * Import numpy and cv2.

```
In [ ]: import cv2
        import numpy as np
```

- 1000 sine values between 0 and 2pi*f for f periods over the width of the picture, add a 1.0 to get all positive numbers, divide by 2 to get the normalized range between 0 and 1, which imwrite expects for floats:

```
In [ ]: f=100
        sinewave=(1.0+np.sin(np.linspace(0,2*np.pi*f,1000)))/2
```

- Diagonal matrix with sine wave on the diagonal:

```
In [ ]: d=np.diag(sinewave)
```

- Matrix with sinewave from left to right, identically on each row, with 500 rows:

```
In [ ]: A=np.dot(np.ones((500,1000)),d)
```

- Display the resulting frame.

```
In [ ]: cv2.imshow('Horizontale Ortsfrequenz',A)
```

- Write photo to jpg file. Mult with 255 because imwrite expects a range of 0...255:

```
In [ ]: cv2.imwrite('horizOrtsfreq.jpg', 255*A)
```

- Keep photo window open until key 'q' is pressed:

```
In [ ]: while(True):
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break
```

- When everything done, release the capture

```
In [ ]: cv2.destroyAllWindows()
```

2 Program 2 - pyrecfftanimation

Using Pyaudio, record sound from the audio device and plot the fft magnitude spectrum life, for 8 seconds. Usage example:

```
python pyrecfftanimation.py
```

- Import relevant libraries

```
In [ ]: import pyaudio
        import struct
        import numpy as np
        import matplotlib.pyplot as plt
        import matplotlib.animation as animation
```

- Defining the variables.

```
In [ ]: CHUNK = 2048 #Blocksize
        WIDTH = 2 #2 bytes per sample
        CHANNELS = 1 #2
        RATE = 32000  #Sampling Rate in Hz
        RECORD_SECONDS = 70

        fftlen=CHUNK/2
```

- Set up for the plot.

```
In [ ]: [fig, ax] = plt.subplots()
        plt.ylabel('dB')
        plt.xlabel('FFT bins/Subbands')
        plt.title('Live FFT Magnitude Spectrum of Microphone Signal')

        x = np.arange(0, fftlen)        # x-array
        #Set scale on y-axis and generate line object with it:
        [line, ]= ax.plot(x, 100.0**np.sin(x))
```

- Function to process the audio blockwise to plot live spectrum of it.

```
In [ ]: def animate(i):
            # update the data
            #Reading from audio input stream into data with block length "CHUNK":
            data = stream.read(CHUNK)
            #Convert from stream of bytes to a list of short integers (2 bytes here) in "samples
            #shorts = (struct.unpack( "128h", data ))
            shorts = (struct.unpack( 'h' * CHUNK, data ));
            samples=np.array(list(shorts),dtype=float);

            #plt.plot(samples)  #<-- here goes the signal processing.
            line.set_ydata(20.0*np.log((np.abs(np.fft.fft(samples[0:fftlen])/np.sqrt(fftlen))+1)
            #line.set_ydata(samples)
            return line,
```

- Initialising the data to be represented.

```
In [ ]: def init():
            line.set_ydata(np.ma.array(x, mask=True))
            return line,
```

- Initialize the soundcard(audio port) for input and output operations.

```
In [ ]: p = pyaudio.PyAudio()
```

- Print out device information about input channels and their corresponding sampling rate.

```
In [ ]: for i in range(0, a):
            print("i = ",i)
            b = p.get_device_info_by_index(i)['maxInputChannels']
            print(b)
            b = p.get_device_info_by_index(i)['defaultSampleRate']
            print(b)
```

- Define the stream of data to be processed for input and output on audio port.

```
In [ ]: stream = p.open(format=p.get_format_from_width(WIDTH),
                        channels=CHANNELS,
                        rate=RATE,
                        input=True,
                        output=True,
                        #input_device_index=3,
                        frames_per_buffer=CHUNK)
```

- Real time plot.

```
In [ ]: print("* recording")
        ani = animation.FuncAnimation(fig, animate,  init_func=init, interval=25, blit=True)
        plt.show()
```

- When everything done, release the capture.

```
In [ ]: print("* done")

        f.close()
        stream.stop_stream()
        stream.close()
```

---

3 Program 3 - pyrecspecwaterfall

Using Pyaudio, record sound from the audio device and plot a waterfall spectrum display, for 8 seconds. Usage example:

```
python pyrecspecwaterfall.py
```

- Import the relevant modules.

```
In [ ]: import pyaudio
        import struct
        import numpy as np
        import cv2
```

- Define the variables.

```
In [ ]: CHUNK = 1024 #Blocksize
        WIDTH = 2 #2 bytes per sample
        CHANNELS = 1 #2
        RATE = 32000  #Sampling Rate in Hz
```

- Initialise the sound card(audio port) and print out the device information.

```
In [ ]: p = pyaudio.PyAudio()

        a = p.get_device_count()
        print("device count=",a)

        for i in range(0, a):
            print("i = ",i)
            b = p.get_device_info_by_index(i)['maxInputChannels']
            print(b)
            b = p.get_device_info_by_index(i)['defaultSampleRate']
            print(b)
```

- Define the stream and its parameters.

```
In [ ]: stream = p.open(format=p.get_format_from_width(WIDTH),
                        channels=CHANNELS,
                        rate=RATE,
                        input=True,
                        output=True,
                        #input_device_index=3,
                        frames_per_buffer=CHUNK)
```

- Start recording and plot the real time waterfall(in opposite direction), where the horizontal axis is the magnitude and vertical axis is the time.

```
In [ ]: while(True):
```

- Reading from audio input stream into data with block length "CHUNK":

```
In [ ]:     data = stream.read(CHUNK)
```

- Convert from stream of bytes to a list of short integers (2 bytes here) in "samples":

```
In [ ]:     shorts = (struct.unpack( 'h' * CHUNK, data ));
            samples=np.array(list(shorts),dtype=float);
```

- Shift "frame" 1 up:

```
In [ ]:     frame[0:(rows-1),:]=frame[1:rows,:];
```

- Compute magnitude of 1D FFT of sound with suitable normalization for the display:write magnitude spectrum in lowes row of "frame":

```
In [ ]:     R=0.25*np.log((np.abs(np.fft.fft(samples[0:fftlen])[0:(fftlen/2)]/np.sqrt(fftlen))+1
```

- Color mapping: Red:

```
In [ ]:     frame[rows-1,:,2]=R
```

- Green:

```
In [ ]:     frame[rows-1,:,1]=np.abs(1-2*R)
```

- Blue:

```
In [ ]:     frame[rows-1,:,0]=1.0-R
```

- Display the resulting frame

```
In [ ]:     cv2.imshow('frame',frame)
```

- Keep window open until key 'q' is pressed:

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

- When everything done, release the capture

```
In [ ]: cv2.destroyAllWindows()

        stream.stop_stream()
        stream.close()
        p.terminate()
```

4 Program 4 - videofft0ifftresampley

Program to capture a video from a camera, compute the Y-component, downsample it by a factor of N horizontally and vertically, and display it live on the screen. * Import numpy and cv2.

```
In [ ]: import numpy as np
        import cv2

        cap = cv2.VideoCapture(0)
```

- Downsampling factor N:

```
In [ ]: N = 4
        [ret, frame] = cap.read()
        [rows,cols,c]=frame.shape;
        r=rows
        c=cols
        Ds0=np.zeros((rows,cols));
        Ds=Ds0
```

- Mask to set to zero the 7/8 highest frequencies, only kep the 1/8 lowest frequencies in each direction:For rows:

```
In [ ]: Mr=np.ones((r,1))
        Mr[(r/8.0):(r-r/8.0),0]=np.zeros((3.0/4.0*r))
```

- For columns:

```
In [ ]: Mc=np.ones((1,c))
        Mc[0,(c/8.0):(c-c/8)]=np.zeros((3.0/4.0*c))
```

- Together

```
In [ ]: M=np.dot(Mr,Mc)
```

```
In [ ]: while(True):
```

- **Encoding side**: Capture frame-by-frame

```
In [ ]:     [ret, frame] = cap.read()
```

- Berechnung der Luminanz-Komponente Y: Y= 0.114*B*+*0.587*G+0.299*R : /256 because the result is float values which imshow expects in range 0...1:

```
In [ ]:     Y=(0.114*frame[:,:,0]+0.587*frame[:,:,1]+0.299*frame[:,:,2])/256
```

- Downgesamplets Y, nur jedes Nte pixel horizontal und vertikal wir uebertragen:

```
In [ ]:     #2D-FFT of Y
            X=np.fft.fft2(Y)
```

- Set to zero the 7/8 highest spacial frequencies in each direction:

In [ ]:
```
X=X*M
```

- inverse 2D-FFT:

In [ ]:
```
Y0=np.abs(np.fft.ifft2(X))
```

- Downgesamplets Y0, nur jedes Nte pixel horizontal und vertikal wir uebertragen:

In [ ]:
```
Ds0[0::N,::N]=Y0[0::N,::N]
```

- **Decoding Side:** Lowpass filter the sampled frame:

In [ ]:
```
Ds0fft=np.fft.fft2(Ds0)
```

- Set to zero the 7/8 highest spacial frequencies in each direction:

In [ ]:
```
Ds1fft=Ds0fft*M
```

- Inverse 2D-FFT:

In [ ]:
```
Yrek=np.abs(np.fft.ifft2(Ds1fft))
```

- Display the resulting frames

In [ ]:
```
cv2.imshow('Decoder: Gefilterte Sampled Frames',Yrek*N*N)
cv2.imshow('Decoder: FFT Bereich nach Null Setzen',np.abs(Ds1fft)/100)
cv2.imshow('Decoder: FFT Bereich der Sampled Frames ',np.abs(Ds0fft)/100)
cv2.imshow('Encoder: Sampled Frames',Ds0)
cv2.imshow('Encoder: Video nach Nullsetzen der hohen Ortsfrequenzen',Y0)
cv2.imshow('Encoder 2D FFT von Y und Null Setzen',np.abs(X)/(480))
#cv2.imshow('Original',frame)
cv2.imshow('Encoder: Luminanz Y',Y)
```

- Press 'q' to close all the windows.

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

- When everything done, release the capture

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

---

5 Program 5 - videofft0ifftresampleykey

Program to capture a video from a camera, compute the Y-component, downsample it by a factor of N horizontally and vertically, and display it live on the screen With keyboard switchable low pass filter and samplingkey f toggles the low pass filter, key s the samplingWith explanation text and state display in the image windows.

- Import numpy and cv2.

```
In [ ]: import numpy as np
        import cv2

        cap = cv2.VideoCapture(0)
```

- Downsampling factor N:

```
In [ ]: N=4;
        [ret, frame] = cap.read()
        [rows,cols,c]=frame.shape;
        r=rows
        c=cols
        Ds0=np.zeros((rows,cols));
        Ds=Ds0
```

- Mask to set to zero the 7/8 highest frequencies, only kep the 1/8 lowest frequencies in each direction:For rows:

```
In [ ]: Mr=np.ones((r,1))
        Mr[(r/8.0):(r-r/8.0),0]=np.zeros((3.0/4.0*r))
```

- For columns:

```
In [ ]: Mc=np.ones((1,c))
        Mc[0,(c/8.0):(c-c/8)]=np.zeros((3.0/4.0*c))
```

- Together:

```
In [ ]: M=np.dot(Mr,Mc)

        ytext=np.zeros((rows,cols))
        cv2.putText(ytext,"Down- and upsampling and LP filtering Demo", (20,50), cv2.FONT_HERSHE
        cv2.putText(ytext,"Toggle LP filter in 2D-FFT on/off: key f", (20,100), cv2.FONT_HERSHEY
        cv2.putText(ytext,"Toggle sampling on/off: key s", (20,150), cv2.FONT_HERSHEY_SIMPLEX, 0
        cv2.putText(ytext,"Quit: key q", (20,200), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255))

        filteron=False
        samplingon=False
```

```
In [ ]: while(True):
```

- **Encoding side:** Capture frame-by-frame

In [ ]:
```
[ret, frame] = cap.read()
```

- Berechnung der Luminanz-Komponente Y: Y= 0.114*B*+0.587G+0.299*R : /256 because the result is float values which imshow expects in range 0...1:

In [ ]:
```
Y=(0.114*frame[:,:,0]+0.587*frame[:,:,1]+0.299*frame[:,:,2])/256
cv2.imshow('Encoder, Original: Luminance Y',Y+ytext)
```

In [ ]:
```
if filteron==True:
```

- 2D-FFT of Y

In [ ]:
```
    X=np.fft.fft2(Y)
```

- `Set to zero the 7/8 highest spacial frequencies in each direction:`

In [ ]:
```
    X=X*M
```

- inverse 2D-FFT:

In [ ]:
```
    Y=np.abs(np.fft.ifft2(X))


if samplingon==True:
```

- Downgesamplets Y0, nur jedes Nte pixel horizontal und vertikal wir uebertragen:

In [ ]:
```
    Y0=np.zeros((rows,cols));
    Y0[0::N,::N]=Y[0::N,::N];
    Y=Y0.copy()
```

- **Decoding Side** Make text:

In [ ]:
```
ytext2=np.zeros((rows,cols))
if samplingon:
   cv2.putText(ytext2,"Sampling on", (20,20), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0.9))
   #print("sampling on")
else:
   cv2.putText(ytext2,"Sampling off", (20,20), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0.9))
   #print("sampling off")

if filteron:
   cv2.putText(ytext2,"Filter on", (20,50), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255))
   #print("filter on")
else:
   cv2.putText(ytext2,"Filter off", (20,50), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255))
   #print("filter off")
```

- 2D-DFT:

```
In [ ]:     Dsfft=np.fft.fft2(Y)

            if filteron == True:
               #Lowpass filter the sampled frame:
               #Dsfilt=N*scipy.signal.convolve2d(Ds0,filt,mode='same')
               #Set to zero the 7/8 highest spacial frequencies in each direction:
               Dsfft=Dsfft*M
            #scaling to maintain the energy after sampling and filtering
            if samplingon and filteron:
                Dsfft=Dsfft*N*N
            cv2.imshow('2D Discrete Fourier Transform of (downsampled, filtered) Luminance Y',np
```

- Inverse 2D-FFT

```
In [ ]:     Y=np.abs(np.fft.ifft2(Dsfft))

            cv2.imshow('Decoder: reconstructed Luminance Y',Y+ytext2)
```

- Key Inputs: - 's' to toggle sampling. - 'f' to toggle filtering. - Press the key "q" to quit window.

```
In [ ]:     key=cv2.waitKey(1) & 0xFF;
            if key == ord('s'):
               samplingon = not samplingon;
            if key == ord('f'):
               filteron = not filteron;
            if key == ord('q'):
                break
```

- When everything done, release the capture

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

---

6 Program 6 - videofiltdisp

Program to capture a video from a camera, filter is, and display it live on the screen * Import the following modules cv2, numpy and scipy.signal.

```
In [ ]: import numpy as np
        import scipy.signal
        import cv2

        cap = cv2.VideoCapture(0)
        [retval, frame] = cap.read()
        [r,c,d]=frame.shape
        print(r,c)
```

- 2D-Mask to set to zero the 7/8 highest frequencies, only kep the 1/8 lowest frequencies in each direction:For rows:

In [ ]:
```
Mr=np.ones((r,1))
Mr[(r/8.0):(r-r/8.0),0]=np.zeros((3.0/4.0*r))
```

- For columns

In [ ]:
```
Mc=np.ones((1,c))
Mc[0,(c/8.0):(c-c/8)]=np.zeros((3.0/4.0*c))
```

- Together:

In [ ]:
```
M=np.dot(Mr,Mc)
```

- Conmpute space-domain/inverse 2D Fourier transform of Low Pass filter:

In [ ]:
```
h=np.abs(np.fft.ifft2(M))
hc=np.concatenate((h[:,(c/2):c],h[:,0:(c/2)]),axis=1)
hc=np.concatenate((hc[(r/2):r,:],hc[0:(r/2),:]))
```

- Only keep the piece with the biggest components:

In [ ]:
```
hc=hc[(r/2-10):(r/2+10),(c/2-10):(c/2+10)]
filt=hc

while(True):
```

- Capture frame-by-frame

In [ ]:
```
    [ret, frame] = cap.read()
    Y=(0.114*frame[:,:,0]+0.587*frame[:,:,1]+0.299*frame[:,:,2])/256;
    Yfilt=scipy.signal.convolve2d(Y,filt,mode='same')
```

- Display the resulting filtered frame

In [ ]:
```
    cv2.imshow('Y low-pass filtered',Yfilt)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

- When everything done, release the capture

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

---

7 Program 7 - videorecfft0ifftdisp

Program to capture a video from the default camera (0), compute the 2D FFT on the Green component, take the magnitude (phase) and display it live on the screen * Import cv2 and numpy.

In [ ]:
```
import cv2
import numpy as np

cap = cv2.VideoCapture(0)
```

- Get size of frame:

In [ ]:
```
[retval, frame] = cap.read()
[r,c,d]=frame.shape
print(r,c)
```

- Mask to set to zero the 3/4 highest frequencies, only kep the 1/4 lowest frequencies in each direction:For rows:

In [ ]:
```
Mr=np.ones((r,1))
Mr[(r/8.0):(r-r/8.0),0]=np.zeros((3.0/4.0*r))
```

- For columns:

In [ ]:
```
Mc=np.ones((1,c))
Mc[0,(c/8.0):(c-c/8)]=np.zeros((3.0/4.0*c))
```

- Together:

In [ ]:
```
M=np.dot(Mr,Mc)
```

In [ ]:
```
while(True):
```

- Capture frame-by-frame

In [ ]:
```
    [retval, frame] = cap.read()
    cv2.imshow('Original Video, Gruen Komponente',frame[:,:,1])
```

- Compute magnitude of 2D FFT of green component with suitable normalization for the display:

In [ ]:
```
    X=np.fft.fft2(frame[:,:,1]/255.0)
```

- Set to zero the 7/8 highest spacial frequencies in each direction:

In [ ]:
```
    X=X*M
    frame=np.abs(X)/512.0
    #angle/phase:
    #frame=(3.14+np.angle(np.fft.fft2(frame[:,:,1]/255.0)))/6.28
```

- Display the resulting frame

In [ ]:
```
    cv2.imshow('2D-FFT mit Null Setzen der hoechsten Ortsfrequenzen',frame)
```

- Inverse FFT, with abs to turn complex into float numbers:

In [ ]:
```
    x=np.abs(np.fft.ifft2(X))
    cv2.imshow('Inverse 2D FFT ohne die hoechsten Ortsfrequenzen', x)
```

- Keep window open until key 'q' is pressed:

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

- When everything done, release the capture

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

---

8 Program 8 - videoresampley

Program to capture a video from a camera, compute the Y-component, downsample it by a factor of N horizontally and vertically, and display it live on the screen. * Import numpy and cv2.

In [ ]:
```
import numpy as np
import cv2

cap = cv2.VideoCapture(0)

cv2.namedWindow('Original')
cv2.namedWindow('Luminanz Y')
cv2.namedWindow('Unterabgetastetes Y')
```

- Downsampling factor N:

In [ ]:
```
N=4;
[ret, frame] = cap.read()
[rows,cols,c]=frame.shape
Ds=np.zeros((rows,cols))
```

In [ ]:
```
while(True):
```

- Capture frame-by-frame

In [ ]:
```
    [ret, frame] = cap.read()
```

- Berechnung der Luminanz-Komponente Y: Y= 0.114*B+0.587*G+0.299*R : /256 because the result is float values which imshow expects in range 0...1:

In [ ]:
```
    Y=(0.114*frame[:,:,0]+0.587*frame[:,:,1]+0.299*frame[:,:,2])/256;
```

- Downgesamplets Y, nur jedes Nte pixel horizontal und vertikal wir uebertragen:

In [ ]:
```
    Ds[0::N,::N]=Y[0::N,::N];
```

- Display the resulting frame

In [ ]:
```
    cv2.imshow('Original',frame)
    cv2.imshow('Luminanz Y',Y)
    cv2.imshow('Unterabgetastetes Y',Ds)
```

- Press 'q' to quit the windows.

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

- When everything done, release the capture

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