Frame Difference Analyzer

Date: 20.04.2009

Author: Alex Arsenovic

Summary:

I wrote a frame difference analyzer. This is not a functional tool, it is made for visualizing and understanding motion detection/scene detection/ anything related to frame differencing. I got this idea from stock market charts called 'Bollinger Bands' http://en.wikipedia.org/wiki/Bollinger_bands. They are used to visualize stock price evolution with respect to time on a scale which relates to the previous trading value of the stock. I thought it would be good for motion detection, because I wanted a dynamic thresholding for motion. This way the noisiness of source video or constant background motion didn't disturb my algorithm. It just happened that this motion detection program also worked as a scene detection program.

What it does:

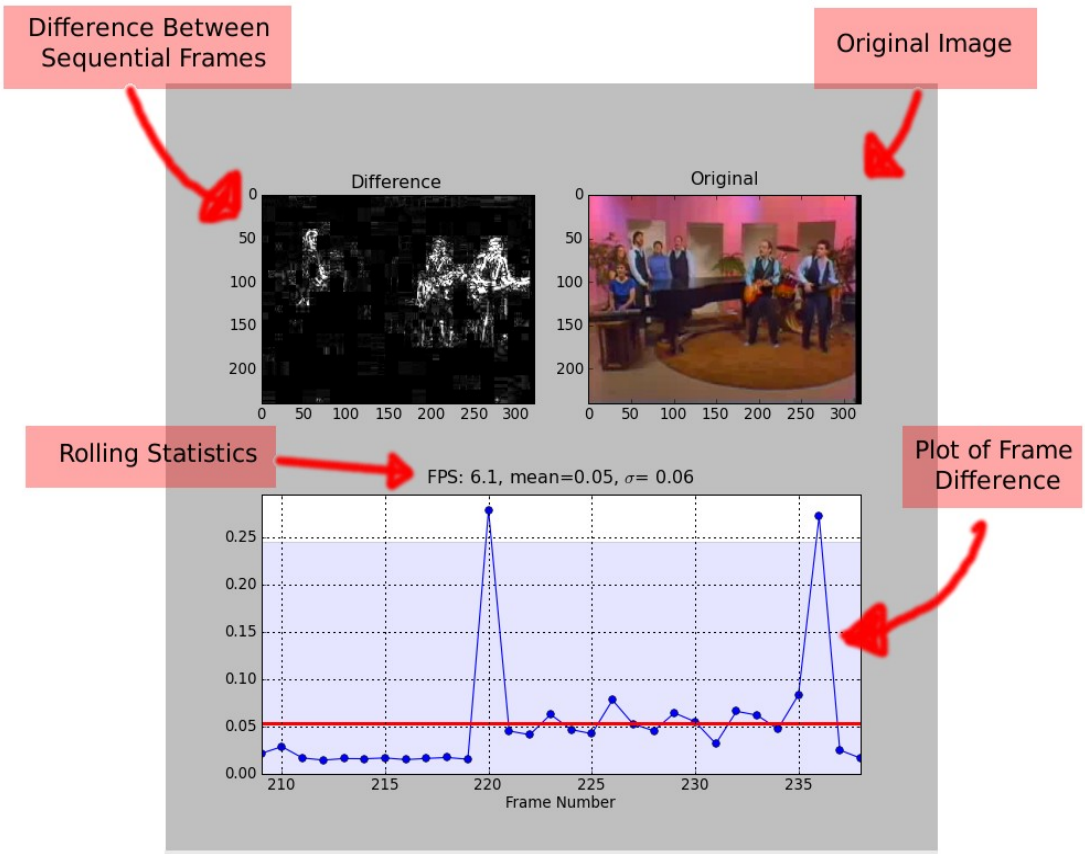
Calculates frame differences between sequential frames in a series of images. Large changes called 'scene changes', are found by calculating a rolling average of the frame differences, and any frame difference beyond a user-configured number of standard deviations is considered a scene change.

The output is broken up into 2 parts: real-time output, and post processing.

- real-time output:
  - shows current frame and difference between current and previous frame.
  - plots the sum of the difference, along with rolling average, and number standard deviations from mean defined by the user.
- post processing:
  - histogram of frame differences for entire movie, with standard deviations labeled
  - histogram of length of scenes, (number of frames between scene changes)
  - time line of movie with scenes colored.

Real-Time Window

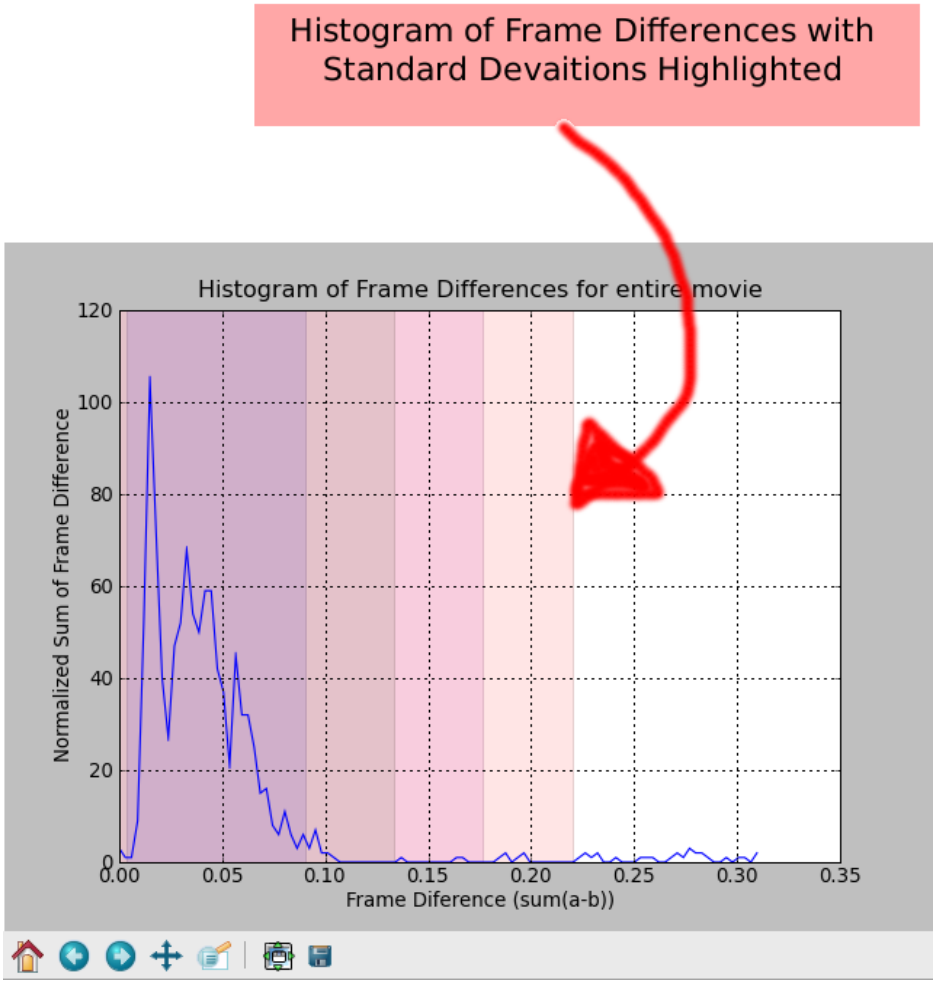
The Real-Time window is displayed if *realtime* switch is set to True. This window displays the current frame of the original movie, along side the difference between the current and previous frame. The difference is calculated by converting the frame to gray scale and doing a simple difference. This difference frame is then summed and normalized and called the 'change' between frames. 'change' is the percent difference between successive frames, where 0 is no change and 1 is 100% change (ie all 1's to all 0's). The 'change' is plotted below the frame displays. The width of the plot is represents the size of you rolling average, so if you are averaging over 100 frames, then the last 100 change values are displayed. Also shown is the rolling average (red line), and the area bound a certain number of standard deviations (light blue). The number of standard deviations is a input parameter. Rolling statistics are shown above the plot at well. When a scene change is detected the y-axis displays "!!! Motion Detected !!" .



Histogram of Frame Difference

The Histogram of frame difference is displayed if *postProcessing* switch is set to True. It displays what the title says. The first 4 standard deviations away from the mean are highlighted. This plot is useful to for choosing the optimal number of standard deviations to required to register a scene change. Frequency this histogram is bimodal, which indicates that there are two distinct groups of frame differences; normal motion, and scene changes. From this example data displayed below we can see that most frame differences are within 1 standard deviation of the mean. This is considered normal frame to frame motion. The values out past 2 standard deviations are probably scene changes.

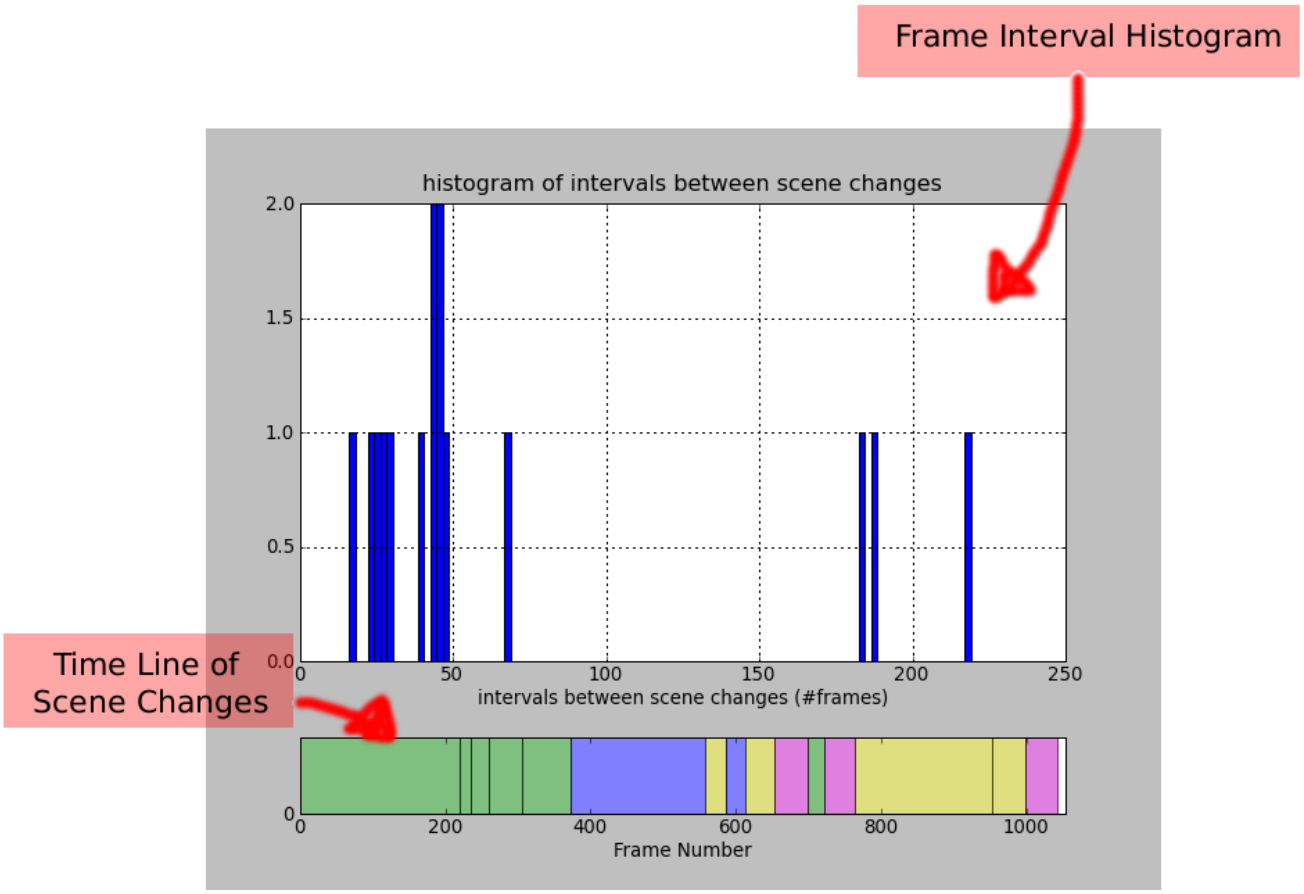
This window can also be used to detect noisiness of a movie . For very noisy movies the lower limit on the histogram is pushed away from zero. this is because the is always change from frame to frame in a noisy movie. I found this to show up on harshly quantized encoded videos.



Scene Interval Histogram, and Time line

The Histogram of frame difference is displayed if *postProcessing* switch is set to True. It displays a histogram of the length of scenes (number of frames between scene changes). This is useful for thresholding scene length. For example if a scene changes lasts longer than 1 frame, such as a pan or flicker-fade (like in Easy Rider), scene changes may last many frames. Depending on what we use our scene information for, we may want to only consider scenes longer than 1 second actual scenes. an example of this would be when you want to import a movie into a video editor and have the scenes cut up into separate clips, but you don't want a bunch of 1 second clips.

Also shown in this window is a time line representing each scene as a colored block. This is just cool to look at to give you a feel for the video structure and I didn't design it for any specific purpose.



How to run it

I developed this using only open-source software. It is written in python with numerical package modules installed. To run this program you will need.

Requirements:

version in bracket is what i have tested it on.

python (2.5.2-16)

matplotlib (0.98.3-5)

scipy (0.6.0-12)

python-gtk (2.12.1-6)

Optional:

ffmpeg (convert video to series of images and back again)

mplayer, mencoder (record from webcam to a series of images)

useful hints at command lines programs:

convert a video to a series of images

```
ffmpeg -i ~/tmp/Rodney_Mullen_To_Clash.flv  -f image2 mullen%4d.png
```

put mplayer in slave mode and capture image when fifo file is echo'd to

```
mplayer -slave -input file=myFifo.tmp -vf screenshot tv:// -tv driver=v4l2 &
echo "screenshot 0" >>  myFifo.tmp
```

CODE

Frame Difference Analyzer

'''
Name: Frame Difference Analyzer
Date: 20.04.2009
Author: alex arsenovic

Summary:
General frame difference analyzer. This is not a functional tool, it is made for visuallizing and understanding motion detection/scene detection/ anything related to frame differencing. This code is not optimized for speed in anyway.

What it does:
Calculates frame differences between sequential frames in a series of images.
large changes are found by calculating a rolling averate of the frame differences, and any frame
difference beyond a user-configured number of standarddeviations is considered a scene change.
The output is broken up into 2 parts: realtime output, and post processing.
- realtime output:
shows current frame and difference between current and previous frame.
plots the sum of the difference, allong with rolling average, and number standarder deviations from mean defined by the user.
- post processing:
histogram of frame differences for entire movie, with standard deviations labeled
histogram of length of scenes, (number of frames between scene changes)
timeline of movie with scenes colored.

Requirements:
version in bracket is what i have tested it on.
python (2.5.2-16)
matplotlib (0.98.3-5)
scipy (0.6.0-12)
python-gtk (2.12.1-6)
'''

```
from scipy.ndimage import *
import glob,os,time,gobject
import matplotlib
matplotlib.use('GTKAgg') # this is used for drawing
from pylab import *


# functions
def rgb2bw(im):
    '''duh'''
    return round_((im[:,:,0]+im[:,:,1]+im[:,:,2])/3)

def rgb2gray(im):
    '''duh'''
```

```
        return ((im[:,:,0]+im[:,:,1]+im[:,:,2])/3)

def point2Gaps(a):
        ''' takes a 1D vector and returns the intervales between sequential elements
        '''
        gaps = zeros(size(a)-1)
        for cnt in range(0,size(a)-1):
                gaps[cnt] = a[cnt+1] - a[cnt]
        return gaps


#################### input       ##########################
picNames = 'jesus'      # spefic to my file structure see load files
changeBufferLength = 30 # length of rolling stats, in frames
minSceneLength = 10     # minimum length of scene, in frames
numStdDev = 3
# output switches
realTimeSwitch = False  # shows realtime analysis along with video
postProcessingSwitch = True     # show post processing results


############## load files       ##########################
files = glob.glob('./'+ picNames + 'Images/*' +picNames+'*.png')
files.sort()

# load initial frame, use this to initialize the imshow, might be done better
im = rgb2gray(imread(files[0]))
changeFrameMax =1 # set this to max value the gray-scaled image can have. depends on
                                # how you convert to grayscale

############### initialize graphics      ##########################
# create subplots for frames
theFig = figure(1)
showSub1 = theFig.add_subplot(221)
theShow1 = imshow(im)
gray()
title('Difference')
showSub2 = theFig.add_subplot(222)
theShow2 = imshow(im)
title('Original')
# create subplot for the frame difference plot
plotSub = theFig.add_subplot(212)
thePlot = plot([0,0],'-o')
grid(1)
xlabel('Frame Number')

# setup rolling average statistics visualizations
meanLine = axhline(0,linewidth=2,color='r')
stdDevSpan = axhspan(0,0,alpha=.1)


################## initial variables      ##########################
# lists to hold frame number and sum of the frame to frame  difference, called 'change'
changeVector=[0]
# holds frame numbers
```

```
frameVector=[0]
frameNo =0
# holds frame numbers which where detected as scene changes
motionFrameVector=[0]

# for fps calculation
tStart = time.time()
t0=tStart
colorString=['y','b','g','m','r']

# needed fro drawing
manager = get_current_fig_manager()



#################### frame difference lope      ########################
def frameDiffLoop(*args):
	global frameNo, im, changeVector, frameVector,t0,motionFrameVector,
colorString,tStop,changeFameMax

	# used for framerate calculation
	perSecond = time.time() - t0
	t0 = time.time()

	# load the frame
	frame = files[frameNo]
	im0 = im
	im = rgb2gray(imread(frame))
	# calculate frame difference
	changeFrame = abs(im- im0)

	# hack needed because the change value from initial frame to first
	# frame is 0. and 0/somthing = nan. could be fixed with a better init
	if changeFrame.sum() == 0:change = 0
	else: change =  changeFrame.sum()/(changeFrame.size * changeFrameMax)

	# update our data vectors
	changeVector.append(change)
	frameVector.append(frameVector[-1]+1)


	#######################  show realtime plot
########################
	thePlot[0].set_data(frameVector,changeVector)
	plotSub.set_ylim([0,max(changeVector)])
	if size(frameVector) > changeBufferLength:
		plotSub.set_xlim([frameVector[-changeBufferLength],frameVector[-1]])
	else:
		plotSub.set_xlim([0,frameVector[-1]])

	# for rolling statistics we can use just the bufferLength most recent
	changeBuffer = changeVector[-changeBufferLength:]
	changeMean = mean(changeBuffer)
	changeStdDev = standard_deviation(changeBuffer)
	upperLimit = changeMean + numStdDev*changeStdDev
```

```
        lowerLimit = changeMean - numStdDev*changeStdDev

        # draw mean line and limits
        meanLine.set_ydata(changeMean)
        stdDevSpan.set_xy(array([\
        [ 0.  ,  lowerLimit],\
        [ 0.  ,  upperLimit],\
        [ 1.  ,  upperLimit],\
        [ 1.  ,  lowerLimit],\
        [ 0.  ,  lowerLimit]]) )


        # draw statistical information
        subplot(212)
        infoString = 'FPS: %01.1f, mean=%0.2f, $\sigma$= %0.2f'%(1./perSecond,
changeMean,changeStdDev)
        title(infoString)


        ##  show the changeFrame
        if change > upperLimit:
                # ASSERT: Motion Detected!
                ylabel('!!!!!! MOTION !!!!!!!')
                motionFrameVector.append(frameNo)
        else:
                ylabel('')

        # update with new frame data
        theShow1.set_data(changeFrame)
        theShow2.set_data(imread(frame))

        if realTimeSwitch:
                theFig.canvas.draw()


        # counter stuff
        frameNo += 1


        # end of movie detection, calls post processing
        if frameNo < size(files):
                return True
        else:
                tStop = time.time()
                if postProcessingSwitch:
                        ######################### post processing
########################
                        # lengths of each scene
                        sceneLengths = point2Gaps(motionFrameVector)
                        # number of scenes which have length > minSceneLength
                        numScenes = sum( array(point2Gaps(motionFrameVector))>minSceneLength)
                        # average frame difference for entire movie
                        changeTotalMean = mean(changeVector)
                        # std deviange for frame difference of entire movie
                        changeTotalStdDev = standard_deviation(changeVector)
```

```
                # histogram of scene lengths
                theFig2 = figure(2,figsize=(10,8))
                ax1 = axes([0.1, 0.3, 0.8, 0.6])
                hist(sceneLengths,bins = 100)
                title('histogram of intervals between scene changes')
                xlabel('intervals between scene changes (#frames)')
                grid(1)

                # timeline of scenes blocks
                ax2 = axes([0.1, 0.1, 0.8, 0.10], yticks=[0])
                xlabel('Frame Number')
                for x in range(0,size(motionFrameVector)-1):
                        axvspan(motionFrameVector[x],motionFrameVector[x+1],alpha=.5,
facecolor=colorString[randint(0,4)])

                xlim([0,frameNo])

                # histogram of frame differences
                theFig3 = figure(3)
                changeVectorHist = histogram(changeVector,bins= size(changeVector)/10)
                plot(changeVectorHist[1],changeVectorHist[0])
                for x in range(1,5):
                        axvspan( changeTotalMean- x*changeTotalStdDev, changeTotalMean +
x*changeTotalStdDev,facecolor=colorString[x],alpha=.1)

                xlabel('Frame Diference (sum(a-b))')
                title('Histogram of Frame Differences for entire movie')
                ylabel('Normalized Sum of Frame Difference ')
                grid(1)
                show()
        return False


################## call to frame Difference loop
########################
gobject.idle_add(frameDiffLoop)
show()

# terminal  output
print 'average framerate: '+repr(int(frameNo/(tStop-tStart)))
print 'number of scenes over ' + repr(minSceneLength) + ' frames: ' +repr(
sum( array(point2Gaps(motionFrameVector))>minSceneLength))
```

Mplayer Webcam Capture

```
import os,time

numFrames=100
waitTime=.1


fifoFileName = 'myFifo.tmp'
os.system('rm shot*.png')
os.system('mkfifo ' + fifoFileName)
os.system('mplayer -slave -input file=myFifo.tmp -vf screenshot tv:// -tv driver=v4l2 &')

time.sleep(2)

for x in range(1,numFrames):
	fileName = 'shot%04d.png'%(x)
	os.system('echo "screenshot 0" >> ' +fifoFileName)
	time.sleep(waitTime)


# very bad way to exit
os.system('killall mplayer')
exit(1)
```

Real Time FFT Display

```
import os,time
import pylab as p
import imageProcessing as a

dir = './images/'
fifoFileName = 'myFifo.tmp'

os.system('rm shot*.png')
os.system('mkfifo ' + fifoFileName)
os.system('mplayer -slave -input file=myFifo.tmp -vf screenshot tv:// -tv driver=v4l2 &')


time.sleep(1)
os.system('echo "screenshot 0" >> ' +fifoFileName)
#time.sleep(1)

p.ion()
f = p.figure(1)
p.gray()
theShow1 = p.imshow(p.rand(480,640), cmap=p.cm.gray)
#f2 = p.figure(2)
#theShow2 = p.imshow(p.rand(480,640), cmap=p.cm.gray)


for x in range(1,20):
	fileName = 'shot%04d.png'%(x)
	print '\n'+ fileName+'\n'
	os.system('echo "screenshot 0" >> ' +fifoFileName)
```

```
        im = p.imread(fileName)

        # Processing
        imGray = a.rgb2bw(im)

        imNew = a.fftPower(imGray)


        p.figure(1)
        theShow1.set_data(imNew)

#       p.figure(2)
#       theShow2.set_data(imFFT)

        p.title(fileName)
        p.draw()

      #time.sleep(1)

# very bad way to exit
os.system('killall mplayer')
```