CS- 512-Assignment 2

Rakshith Churchagundi Amarnath
CWID: A20424771
Fall2018

ABSTRACT

We write a program to perform simple image manipulation using OpenCV. The program loads an image by either reading it from a file or capturing it directly from the camera. Computer vision is a multidisciplinary field that deals a lot with image processing and very intuitive level of understanding and analyzing images . It closely seeks to automate the human visual system, which can be a problem while implementing in terms of computer language , a lot of complications can be involved . The problem we are facing is a simple image manipulation using OpenCV, The input can be an image of a certain generic formats being given to a function which in turn processes the image and performs various tasks such as converting it to a grayscale, cycling through different color channels and outputting, the original image in different windows with varying colors. Sampling the images by scaling up and down with the implementation of features such as smoothing, rotating with respect to an angle at a given axis in 3D and 2D worlds, translate and scale it up and down, using filters to remove noises in the images and for further enhancing of images. Computer vision seeks to apply its theories and models for the construction of computer vision systems.

Problem Statement:

The problem we are facing here is in processing the image with a user written program which explicitly takes an image and loads it and then perform various functions such as

- **Loading an image**: Here we have to load an image from a directory or we have to allow the camera to capture an image and store in a buffer and perform transformation on that image to enhance that and makes it easy to read by the vision algorithms.
- **Saving the file**: Saving of an image to different locations after being read.
- **Color processing an image**: Converting the original image to different scales - grayscale is among the top functions that is performed after inputting an image into vision algorithms
- **Converting an image**: conversion includes color to black and white or even it can be from a bit conversion of an image.
- **Transformation of images**: Here we need to perform different types of transformation such as translation, scaling and rotating an image
- **Noise reducing** : it's the most crucial part in processing the image, distortion and any amount of grains has to be reduced to optimize our functions
- **Smoothing** – Smoothing is often used to reduce noise within an image. It can also be used to produce a less pixelated image(blur).

- **Factoring an image**: Sampling up and down, with and without the smoothing filter.
- **Color gradient and magnitude**: The range of positions of different color pixels in an image is located along with the variation in the intensity of the image.
- **Complexity of a problem**: we have to analyze the problem's complexity to such an extent that we should be able to implement different command line arguments with a great flow for our algorithm.

Proposed solution:

Open CV linked with anaconda python. Processing the image which was directly captured by a camera on runtime. Processing an image after its done loading – transforming an image to make it best fit for the functions so that it gives an efficient output every time a different size image with different dimensions are given as an input different formats of images. Installing different libraries to make the new functions compatible with the existing code such as matplotlib to maintain a good structure for the program we have written after adding many functions .

Algorithm:

The basic algorithm or the approach starts with giving an image of certain dimensions and format which will then be altered with respect to different co-ordinate systems. The user can select among different options which is implemented by using an if-else statements. Different functions gets executed when their keys are given as an input. After performing the respective functionality, the control of the program will be returned to main menu again form which users can perform other operations related to the image processing. For a particular image which is either uploaded from the directory or captured live using a webcam, it can be processed using different functionalities stated below

Step 1: start

Step 2: select an appropriate option

Step 3: image is uploaded or captured via webcam

Step 4*:* The available options will be displayed and the image operation is selected from the menu

Step 5: As per the option selected, the function performing the operation are called

Step 6: After the operation is performed, the function control will again provide the option menu to do operations.

Implementation Details:

The implementation for the above proposed problem statement is as follows:
We set the path using the below functions

```
parser=argparse.ArgumentParser()
parser.add_argument("--path",help="path")
args=parser.parse_args()
```

The above functions help us to know where the command passed is saved.

- **Loading an image**

  Problem design issues and problems faced:

  Here we are finding it difficult to know how to set the path to where image is be captured and loaded. We have to know how the waitKey() function works. Found it difficult to capture the image and display it.

  So, Its solved by using the below code implementations and the setting path issue was solved by using the function:

  ```
  parser=argparse.ArgumentParser()
  parser.add_argument("--path",help="path")
  args=parser.parse_args()
  ```

  To use this I imported the argparse() package.

  The implementation for this problem is as follows:

  ```
  if(len(sys.argv)< 2):
  image=cv2.VideoCapture(0)
  check,img=image.read()
  ```

  --These commands helps us capture the image from the camera.

  ```
  cv2.imshow('Capturing',img)
  ```

  -This command helps us set the frame to display the image captured in the previous step.

  ```
  Kernel_cv=cv2.waitKey(0)
  image.release()
  else:
  img=cv2.imread("%s"%(args.path),1)
  ```

  -This command helps us read the image from the path set.

  ```
  cv2.imshow('Image',img)
  ```

  -This command helps to display the image on the screen

  ```
  Kernel_cv=cv2.waitKey(0)
  ```

- **Color gradient and magnitude**

  Problem design issues and problems faced:

Here I didn’t know that the order in cv2 was BGR and not RGB. So, had difficulty in solving that. There were problems setting the other 2 channels to 0.
I solved the above problems faced using the following code implementations:

```
blue_img=np.copy(img)
```
-This command helps us load the color “blue” in the image. But this happens only when the other 2 channels are made ‘0’.
```
green_img=np.copy(img)
```
- This command helps us load the color “green” in the image. But this happens only when the other 2 channels are made ‘0’.
```
red_img=np.copy(img)
```
- This command helps us load the color “red” in the image. But this happens only when the other 2 channels are made ‘0’.
```
blue_img[:,:,1:]=0
```
-This command helps me set the first and second channels to 0.
```
green_img[:,:,(0,2)]=0
```
-This command helps me set the first and third channels to 0.
```
#2,3 channels are made 0
red_img[:,:,:2]=0
```
-This command helps me set the second and third channels to 0.

The following code helps us cycle through the color channels on pressing the key ‘c’.
```
if(kernel_cv==ord('c')):
cv2.imshow('Blue Channel',blue_img)
```
-These commands help cycling to the image in ‘blue’

```
r=cv2.waitKey(0)
```
-This command delays the rendering of images to windows by some ‘n’ milliseconds.

```
if(r==ord('c')):
cv2.imshow('Green Channel',green_img)
```
-These commands help cycling to the image in ‘green’

```
g=cv2.waitKey(0)
```
-This command delays the rendering of images to windows by some ‘n’ milliseconds.

```
if(g==ord('c')):
cv2.imshow('Red Channel',red_img)
```
-These commands help cycling to the image in ‘red’

- **Saving the image to file**

To save the image, the following code is used:
if k==ord('w'):
cv2.imwrite("out.jpg",img)
-this imwrite() command helps us save the image captured and save it in the 'out.jpg' file.

- **Color processing of an image**
  <u>Problem design issues and problems faced</u>:
  The process of Color processing of an image consists of converting the image into grayscale and transforming image, was a little tricky for me as I was not properly aware of how to scale and convert the image from color to grayscale. To overcome that, below code implementations helped me.

  image_gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
  -*cvtColor* function is used to change the color of the image and the *COLOR_BGR2GRAY* to change it into gray.
  cv2.imshow("Image_grayscale",image_gray)
  Here the imshow() function shows the image into window.
  g=cv2.waitKey(0)

- **Smoothing**
  <u>Problem design issues and problems faced</u>:
  Here I had problem in finding out how to use the scaling factor so as to smooth the image. It was bit tricky to come out with a solution.
  So, I used the sliderHandler() function as below and then implemented the smoothing function.
  def sliderHandler(n):
  global img
  kernel_cv=np.ones((n,n),np.float32)/(n*n)
  dst=cv2.filter2D(img,-1,kernel)
  cv2.imshow('processed',dst)
  -These commands show the processed image.
  if(kernel_cv==ord('s')):
  img=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
  -This command converts the image to gray scale.
  cv2.imshow('blur',img)
  cv2.createTrackbar('s','blur',0,10,sliderHandler)
  cv2.waitKey(0)
  -These commands helps to scale and navigate between blurr to sharp smoothing.

- **Factoring an image: Upsampling and downsampling**

By factoring of an image, I am creating an "access" image that is a miniaturized or maximized duplicate of your optical resolution "master" scan, basically a way of compressing the image with minimum loss of pixel data. Here, in the problem, I was asked to downscale an image by a factor of 2 without and with smoothing the pixel, for that, I have used resize function of OpenCV, which was very useful to use but the set output image size

To achieve that, below code is used for **downscale the image by 2 without smoothing**

```
print(img.shape)
```
-printing the original image

```
lower_reso = cv2.resize(img,(int(img.shape[1]/4),int(img.shape[0]/4)))
```
-The resize function is taking the original image and shaping it by dividing the number of rows(img.shape[0])and column(img.shape[1]) by 4 for the factoring the image by 2.

```
print(lower_reso.shape)
```
-Printing the low res image

```
cv2.imshow('Modified_Image',lower_reso)
cv2.waitKey(0)
```

This is code for **downscale the image by 2 with smoothing**

```
print(img.shape)
```
-Printing the original image

```
lower_reso = cv2.pyrDown(img)
```
-Here we are using pyrDown instead of resize Because pyrDown not only downsamples it but also blurs an image which is required for the action of smoothing

```
print(lower_reso.shape)
```
-Printing the low res image

```
factorBy4=cv2.pyrDown(lower_reso)
```
-using pyrDown will scale down the image only by factor 1 so applying it again to downsample the image to factor 2.

```
print(factorBy4.shape)
cv2.imshow('factorBy4',factorBy4)
cv2.imshow('Modified_Image',lower_reso)
```
- Printing and showing both the images.

```
cv2.waitKey(0)
```

- **Filtering**

```
allocate outputPixelValue[image width][image height]
allocate window[window width * window height]
edgex := (window width / 2) rounded down
edgey := (window height / 2) rounded down
for x from edgex to image width - edgex
for y from edgey to image height - edgey
i = 0
for fx from 0 to window width
for fy from 0 to window height
window[i] := inputPixelValue[x + fx - edgex][y + fy - edgey]
 i := i + 1
 sort entries in window[]
  outputPixelValue[x][y] := window[window width * window height / 2]
```

- **X-derivative**

  The following code implementations helped me get the x-derivative of the image.

```
if (kernel_cv==ord('x')):
image_gray=cv2.cvtColor(imge,cv2.COLOR_BGR2GRAY)
sobelx = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]], dtype = np.float)
sobely = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]], dtype = np.float)
gx = cv2.filter2D(image_gray, -1, sobelx)
gy = cv2.filter2D(image_gray, -1, sobely)
g = np.sqrt(gx * gx + gy * gy)
g *= 255.0 / np.max(g)
cv2.imshow('x_derivative',gx)
```

  -These commands helps us retrieve the x-derivative of the image.

```
g=cv2.waitKey(0)
cv2.destroyAllWindows()
```

- **Y-derivative**

```
if (kernel_cv==ord('y')):
image_gray=cv2.cvtColor(imge,cv2.COLOR_BGR2GRAY)
sobelx = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]], dtype = np.float)
sobely = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]], dtype = np.float)
gx = cv2.filter2D(image_gray, -1, sobelx)
gy = cv2.filter2D(image_gray, -1, sobely)
g = np.sqrt(gx * gx + gy * gy)
g *= 255.0 / np.max(g)
cv2.imshow('y_derivative',gy)
```

-This helps to normalize output to fit the range. This command helps to get the x-derivative of the image.

```
g=cv2.waitKey(0)
cv2.destroyAllWindows()
```

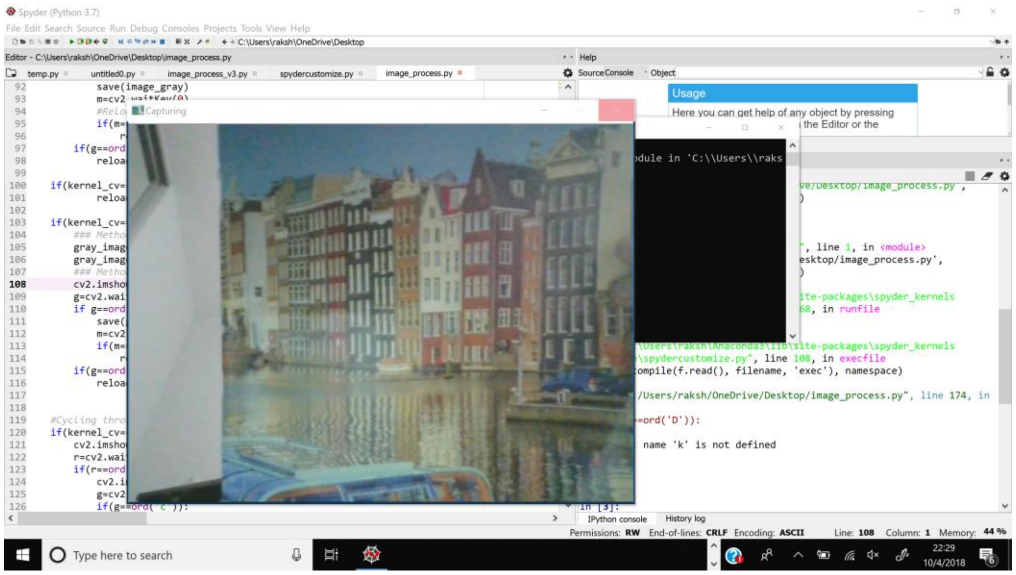
Results and Discussions

- Loading the image

  Compiling and running the program captures the image from camera.

  -Pressing key ‘i’ reloads the image.

  -Pressing key ‘w’ saves the image



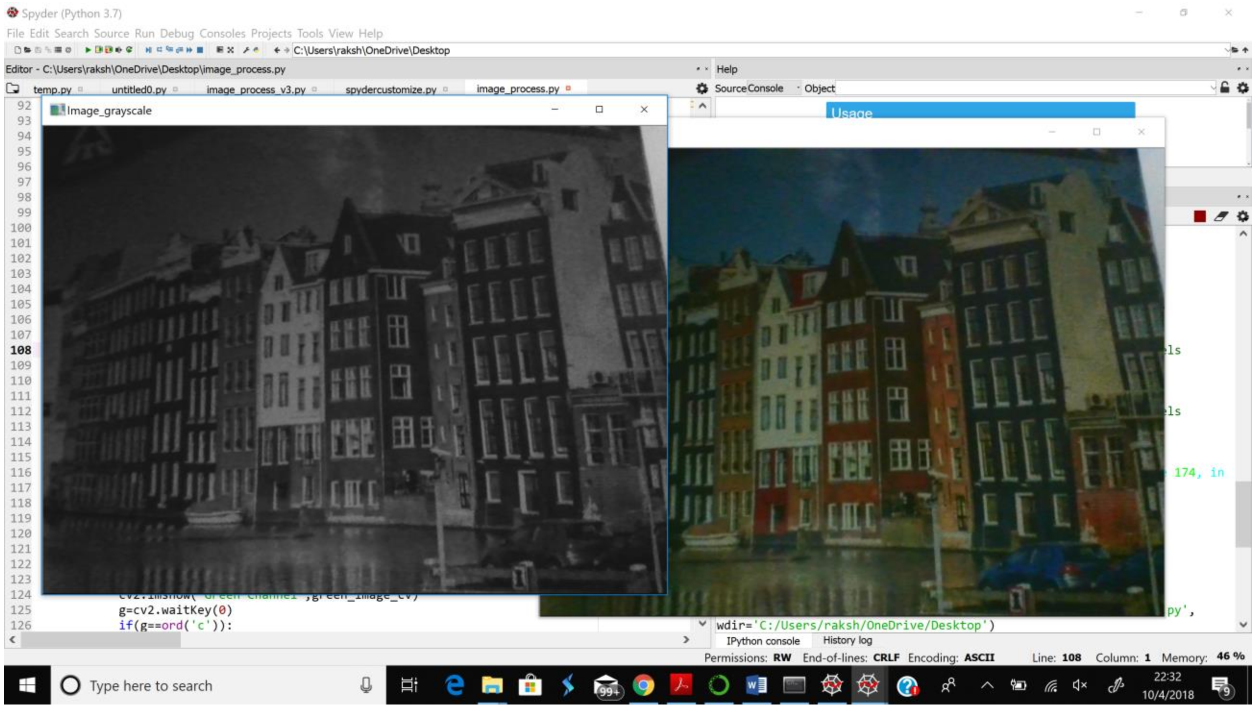
```
            save(image_gray)
            m=cv2.waitKey(0)
            #Relo
            if(m=
                r
        if(g==ord
            reloa

    if(kernel_cv=
            reloa

    if(kernel_cv=
        ### Metho
        gray_imag
        gray_imag
        ### Metho
        cv2.imsho
        g=cv2.wai
        if g==ord
            save(
            m=cv2
            if(m=
                r
        if(g==ord
            reloa


    #Cycling thro
    if(kernel_cv=
        cv2.imsho
        r=cv2.wai
        if(r==ord
            cv2.i
            g=cv2
            if(g==ord('c')):
```

```
odule in 'C:\\Users\\raks
ve/Desktop/image_process.py",
)

", line 1, in <module>
esktop/image_process.py',
)

ite-packages\spyder_kernels
68, in runfile

\Users\raksh\Anaconda3\lib\site-packages\spyder_kernels
\spydercustomize.py", line 108, in execfile
ompile(f.read(), filename, 'exec'), namespace)

/Users/raksh/OneDrive/Desktop/image_process.py", line 174, in

=ord('D')):

 name 'k' is not defined

In [3]:
```

```
Permissions: RW   End-of-lines: CRLF   Encoding: ASCII   Line: 108   Column: 1   Memory: 44 %
```

Gray Scale:

Pressing ‘g’ will give the output in a grayscale image.

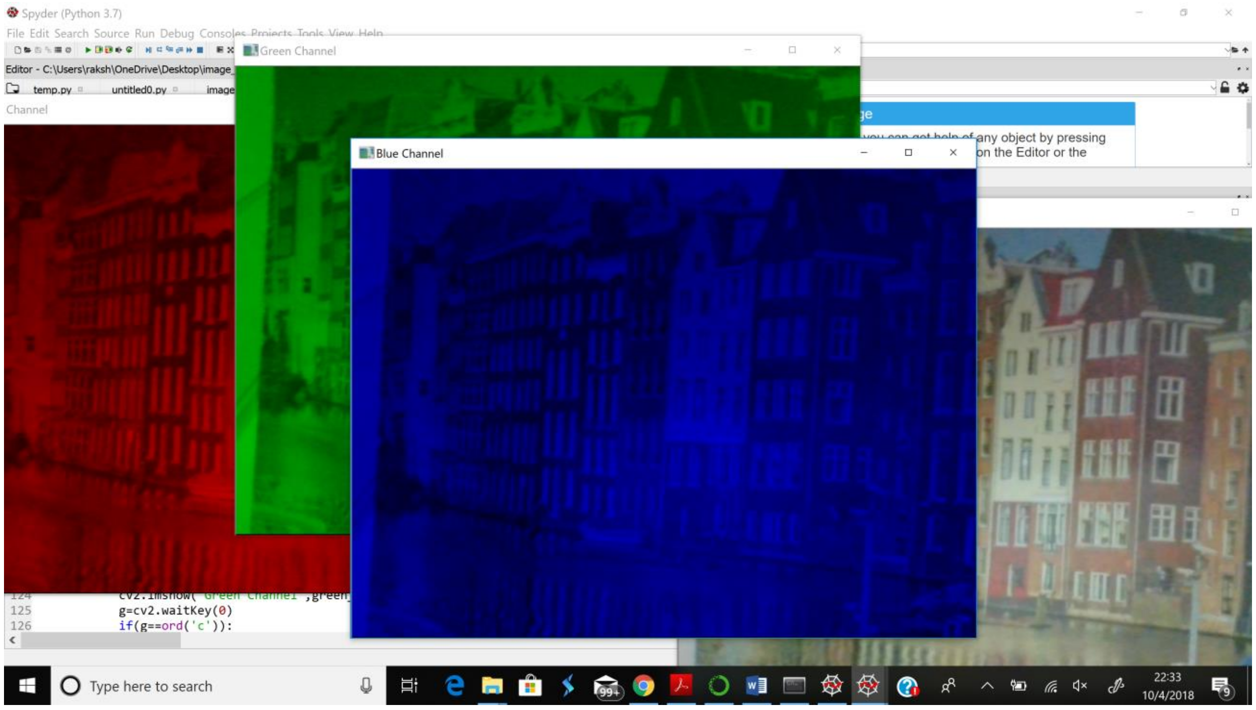


```
        cv2.imshow('Green Channel',green_image_cv)
        g=cv2.waitKey(0)
        if(g==ord('c')):
```

```
wdir='C:/Users/raksh/OneDrive/Desktop')
```

RGB color Gradient

Pressing the ‘c’ key, the output window will cycle with RGB colors

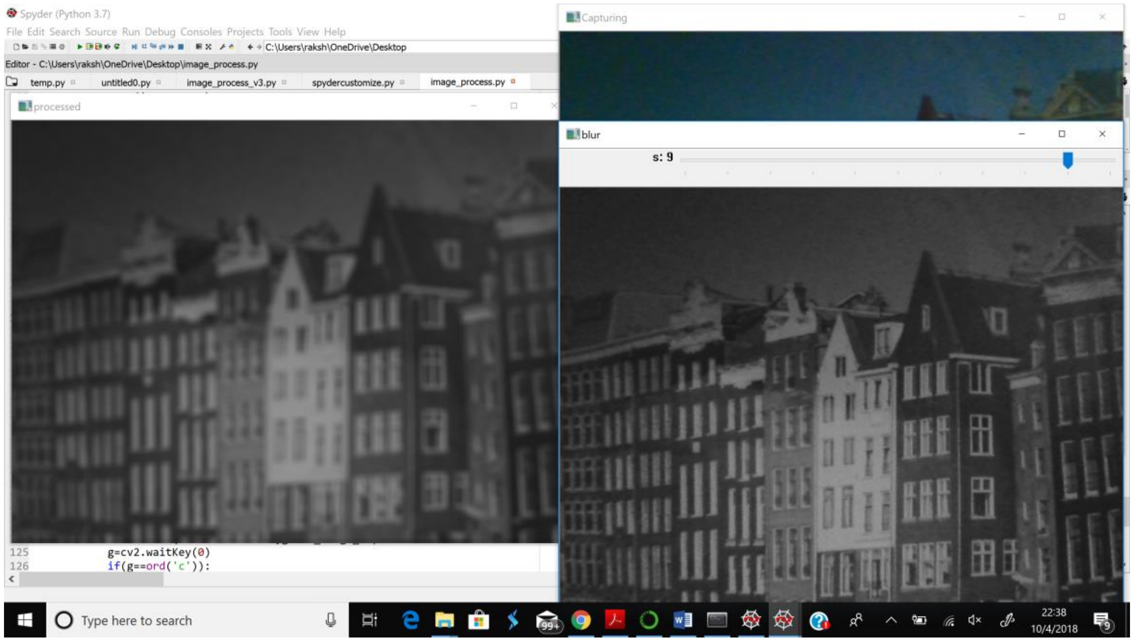


```
        cv2.imshow('Green Channel',green
        g=cv2.waitKey(0)
        if(g==ord('c')):
```

Smoothing:

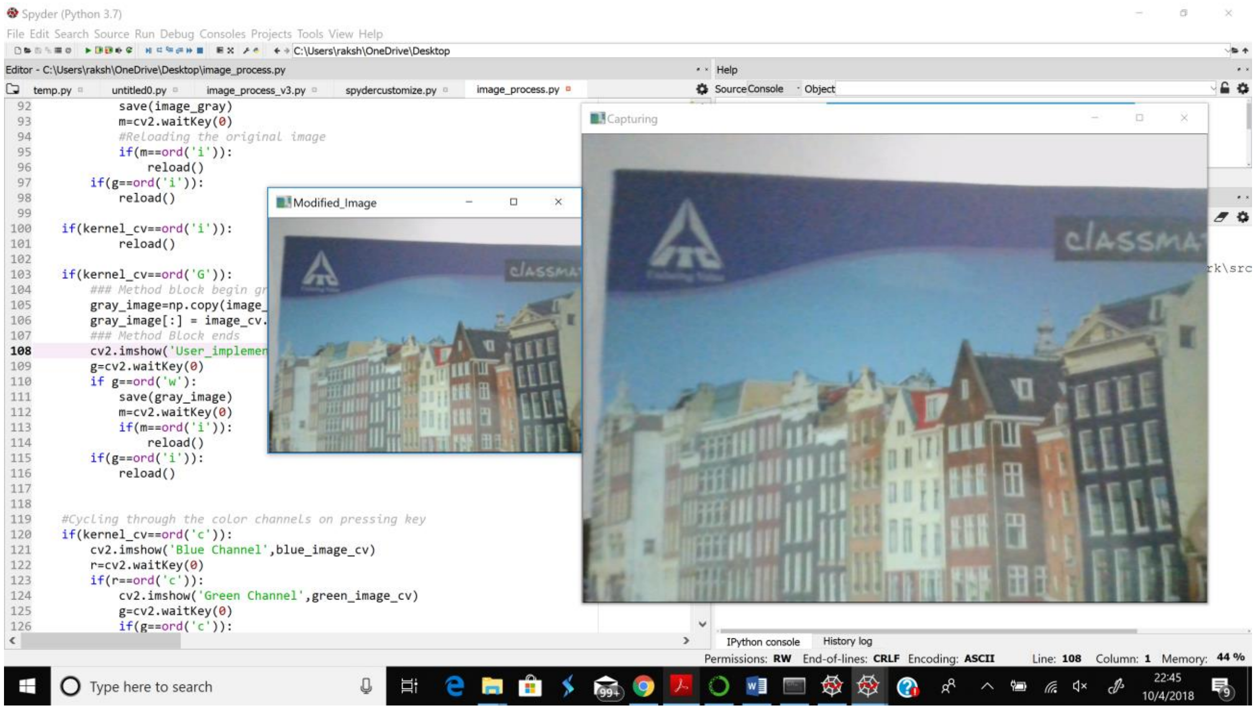
Pressing ‘s’ will smoothens the image – a blurred image with respect to the slider increased will lead to the smooth image in another window.

Pressing ‘S’ will also leads to the smoothen image output which uses convolution filter.


Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\raksh\OneDrive\Desktop
Editor - C:\Users\raksh\OneDrive\Desktop\image_process.py
temp.py
untitled0.py
image_process_v3.py
spydercustomize.py
image_process.py
processed
Capturing
blur
s: 9
Type here to search
22:38
10/4/2018

```
            g=cv2.waitKey(0)
            if(g==ord('c')):
```

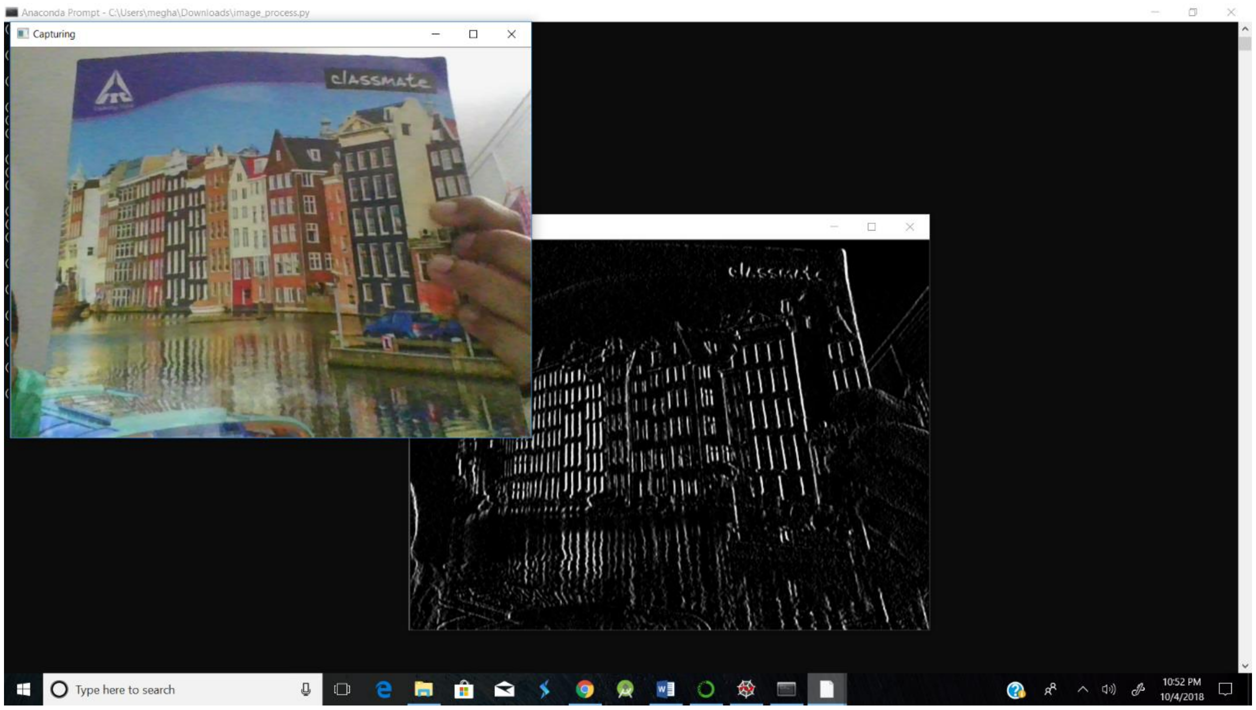
Sampling: pressing ‘d’ will down sample without smoothing. And ‘D’ will down sample with sample.



```
                save(image_gray)
                m=cv2.waitKey(0)
                #Reloading the original image
                if(m==ord('i')):
                    reload()
            if(g==ord('i')):
                reload()

        if(kernel_cv==ord('i')):
                reload()

        if(kernel_cv==ord('G')):
            ### Method block begin gr
            gray_image=np.copy(image_
            gray_image[:] = image_cv.
            ### Method Block ends
            cv2.imshow('User_implemen
            g=cv2.waitKey(0)
            if g==ord('w'):
                save(gray_image)
                m=cv2.waitKey(0)
                if(m==ord('i')):
                    reload()
            if(g==ord('i')):
                reload()


        #Cycling through the color channels on pressing key
        if(kernel_cv==ord('c')):
            cv2.imshow('Blue Channel',blue_image_cv)
            r=cv2.waitKey(0)
            if(r==ord('c')):
                cv2.imshow('Green Channel',green_image_cv)
                g=cv2.waitKey(0)
                if(g==ord('c')):
```

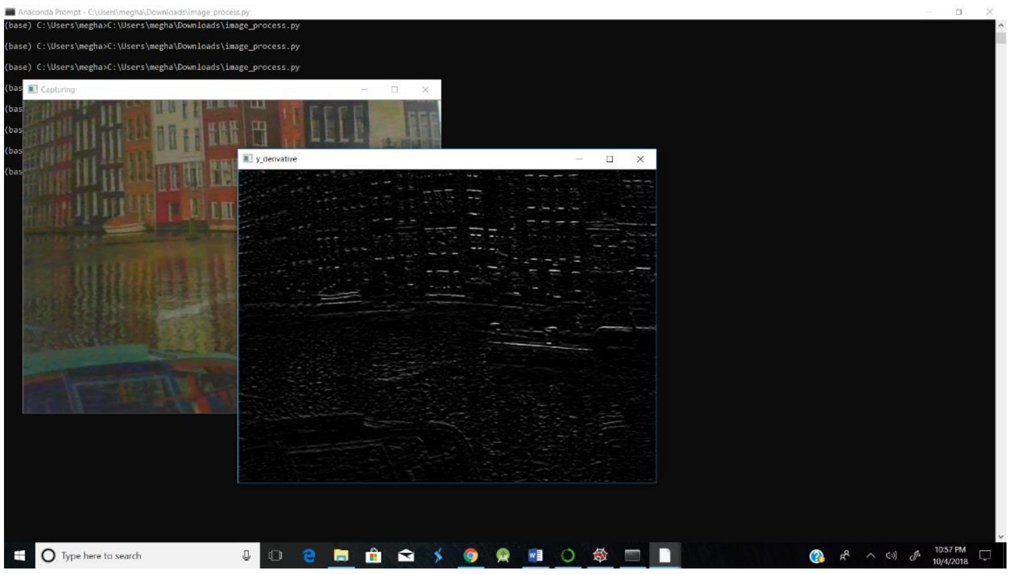
X-derivatives:

Pressing ‘x’ converts the image to grayscale and perform convolution with x-derivative filter.



Y-derivative:

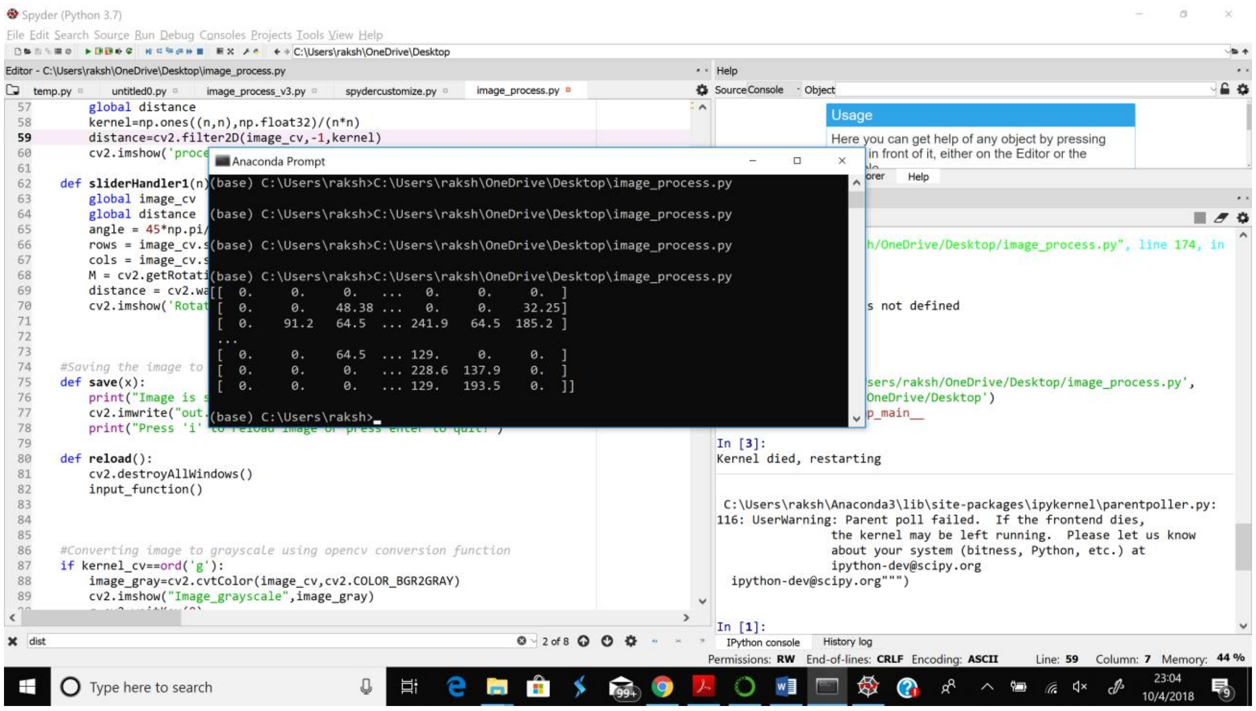
Pressing ‘y’ converts the image to grayscale and perform convolution with y-derivative filter



```
(base) C:\Users\megha>C:\Users\megha\Downloads\image_process.py

(base) C:\Users\megha>C:\Users\megha\Downloads\image_process.py

(base) C:\Users\megha>C:\Users\megha\Downloads\image_process.py

(bas

(bas

(bas

(bas

(bas
```

Magnitude:

Pressing m shows the magnitude of the gradient normalized to the range[0,255].

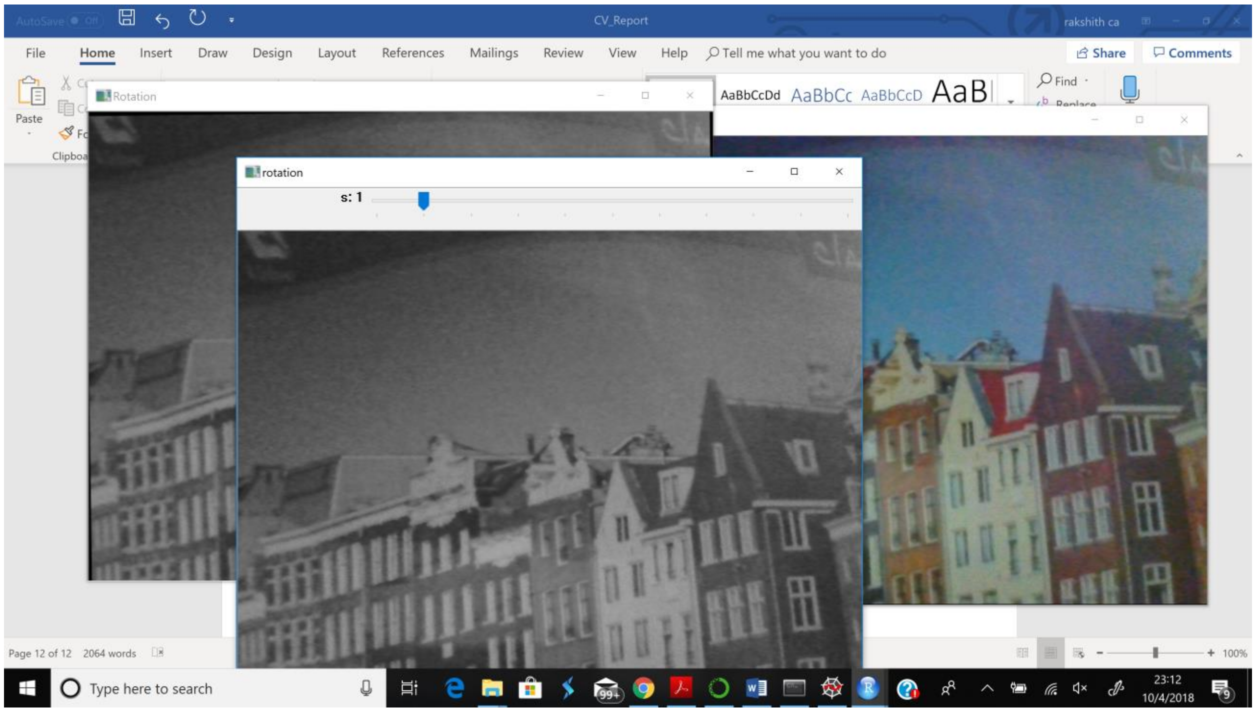


```
    global distance
    kernel=np.ones((n,n),np.float32)/(n*n)
    distance=cv2.filter2D(image_cv,-1,kernel)
    cv2.imshow('proce

def sliderHandler1(n)
    global image_cv
    global distance
    angle = 45*np.pi/
    rows = image_cv.s
    cols = image_cv.s
    M = cv2.getRotati
    distance = cv2.wa
    cv2.imshow('Rotat



#Saving the image to
def save(x):
    print("Image is s
    cv2.imwrite("out.
    print("Press 'i' to reload image or press enter to quit!")

def reload():
    cv2.destroyAllWindows()
    input_function()



#Converting image to grayscale using opencv conversion function
if kernel_cv==ord('g'):
    image_gray=cv2.cvtColor(image_cv,cv2.COLOR_BGR2GRAY)
    cv2.imshow("Image_grayscale",image_gray)
```

```
(base) C:\Users\raksh>C:\Users\raksh\OneDrive\Desktop\image_process.py

(base) C:\Users\raksh>C:\Users\raksh\OneDrive\Desktop\image_process.py

(base) C:\Users\raksh>C:\Users\raksh\OneDrive\Desktop\image_process.py

(base) C:\Users\raksh>C:\Users\raksh\OneDrive\Desktop\image_process.py
[[  0.     0.     0.   ...   0.     0.     0.  ]
 [  0.     0.    48.38 ...   0.     0.    32.25]
 [  0.    91.2   64.5  ... 241.9   64.5  185.2 ]
 ...
 [  0.     0.    64.5  ... 129.     0.     0.  ]
 [  0.     0.     0.   ... 228.6  137.9    0.  ]
 [  0.     0.     0.   ... 129.   193.5    0.  ]]

(base) C:\Users\raksh>
```

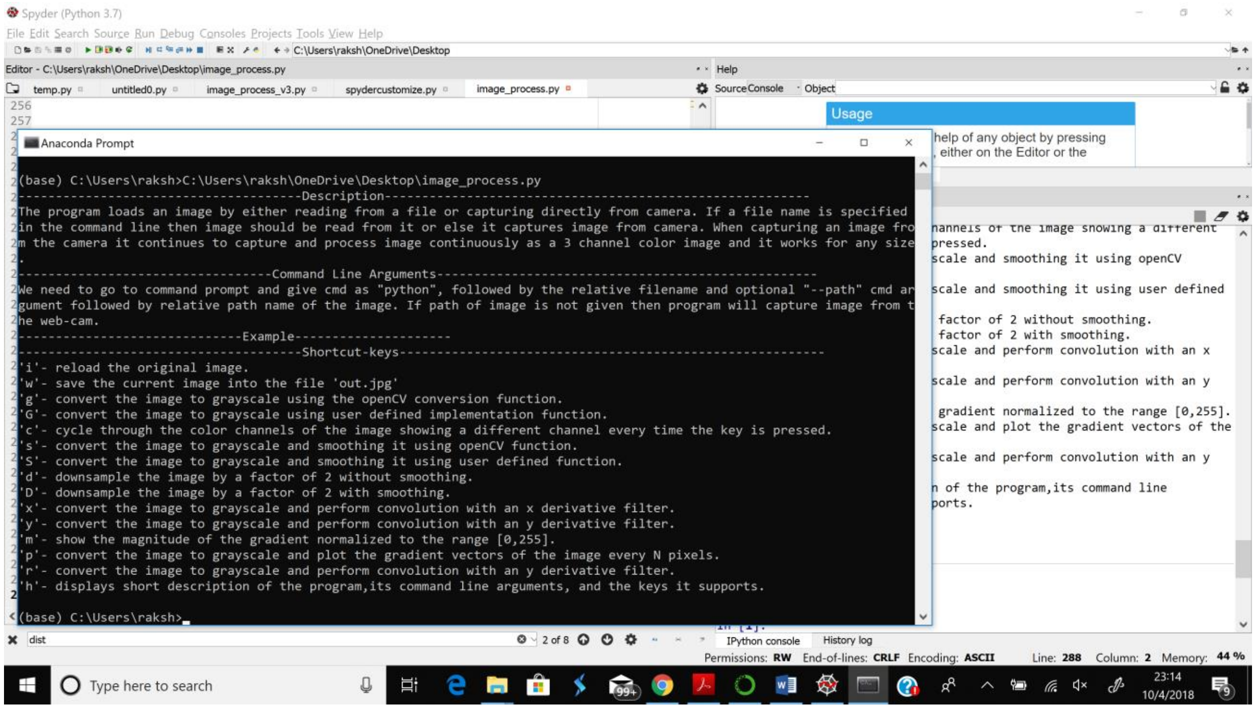
```
h/OneDrive/Desktop/image_process.py", line 174, in

s not defined

sers/raksh/OneDrive/Desktop/image_process.py',
OneDrive/Desktop')
p_main__

In [3]:
Kernel died, restarting

 C:\Users\raksh\Anaconda3\lib\site-packages\ipykernel\parentpoller.py:
116: UserWarning: Parent poll failed.  If the frontend dies,
                the kernel may be left running.  Please let us know
                about your system (bitness, Python, etc.) at
                ipython-dev@scipy.org
  ipython-dev@scipy.org""")

In [1]:
```

Rotation:Pressing ‘r’ will rotate the image.



Short Description:

Pressing ‘h’ will give the short description.



```
(base) C:\Users\raksh>C:\Users\raksh\OneDrive\Desktop\image_process.py
------------------------------------Description------------------------------------------------------
The program loads an image by either reading from a file or capturing directly from camera. If a file name is specified
in the command line then image should be read from it or else it captures image from camera. When capturing an image fro
m the camera it continues to capture and process image continuously as a 3 channel color image and it works for any size
.
--------------------------------Command Line Arguments-------------------------------------------------
We need to go to command prompt and give cmd as "python", followed by the relative filename and optional "--path" cmd ar
gument followed by relative path name of the image. If path of image is not given then program will capture image from t
he web-cam.
-----------------------------Example---------------------
------------------------------------Shortcut-keys------------------------------------------------------
'i'- reload the original image.
'w'- save the current image into the file 'out.jpg'
'g'- convert the image to grayscale using the openCV conversion function.
'G'- convert the image to grayscale using user defined implementation function.
'c'- cycle through the color channels of the image showing a different channel every time the key is pressed.
's'- convert the image to grayscale and smoothing it using openCV function.
'S'- convert the image to grayscale and smoothing it using user defined function.
'd'- downsample the image by a factor of 2 without smoothing.
'D'- downsample the image by a factor of 2 with smoothing.
'x'- convert the image to grayscale and perform convolution with an x derivative filter.
'y'- convert the image to grayscale and perform convolution with an y derivative filter.
'm'- show the magnitude of the gradient normalized to the range [0,255].
'p'- convert the image to grayscale and plot the gradient vectors of the image every N pixels.
'r'- convert the image to grayscale and perform convolution with an y derivative filter.
'h'- displays short description of the program,its command line arguments, and the keys it supports.

(base) C:\Users\raksh>
```

References:

1. https://stackoverflow.com/search?q=opencv
2. https://opencv.org/
3. https://docs.opencv.org/3.0-beta/doc/tutorials/tutorials.html
4. https://github.com/opencv/opencv