REPORT

VISMAYA VEERAMANJU KALYAN

CV512 FALL-18

A20423189

Problem Statement:

The program should take the image as an input using the command line or use the video camera the path of the image is not specified. When the program runs if you press 'h' the program shows you all the options you can use to modify the image in the terminal. The main intension is to shows how a specific function works on an image using OpenCV.

Proposed Solution:

We are using python, OpenCV and Numpy. We are trying to show how to use the inbuilt OpenCV function as well write our own function which does the similar function like how to gray scale an image or see how a single-color channel looks and many more basic concepts convolution. We are trying to explore the fundamentals of computer vision for diving deeper into the subject in this report.

Implementation Details:

The folder contains 3 subfolders named data, doc and scr. The source has the python solution code for this example, data contains the image I have used for testing and doc contains the report and assignment.

We can call the script in the command line using **python as2.py** in which case we use the web camera for showing the results of the function.

We can also send the image with its path to the program in which case the functions are applied on the given image. **python as2.py --image D:\CV\cs512-f18-vismayaveeramanju-kalyan\AS2\data\car.jpeg**

i.e. the program has an option of reading an image from the command line and using it else uses the video camera to do the processing.

We need to first import libraries like OpenCV and numpy.

Imshow(win, img) function is used to display the image with the help of which we can visualize the output of the function.

It has 2 parameters

- Window name to be given
- Image to be displayed.

Problems faced:

- OpenCV with python doesn't run fast on the own implementation of the convolution function.
- The program could be better optimized as the code is bit redundant.
- OpenCV doesn't provide keyboard callbacks.

1. 'I' reloads the image. It uses the OpenCV imread(img , color_space) function to read the image. It has two parameters

- Full path of image.
- The way image should be read.
  IMREAD_COLOR (color image in BGR format) , IMREAD_GRAYSCALE (loads in grayscale mode), IMREAD_UNCHANGED (loads image including alpha channel.

2. 'w' save the current sections image into the output.jpg using the imwrite(filename, dst) function.

- Takes names of the file.
- Image to be saved.

3. 'g' converts the image to gray scale using the cvtColor(image, cv2.COLOR_BGR2GRAY).

- The first parameter is the original image.
- second parameter output image of the same size.
- code – color space conversion code.

4. 'G' converts the image to gray scale using the CIE XYZ systems only Y component as it gives the luminance component.

gray = 0.01 * image[:, :, 2] + 0.813 * image[:, :, 1] + 0.177 * image[:, :, 0]

0.01 * Blue component + 0.814 * Green component + 0.17 * Red component

5. 'c' cycles through the color channels.

b, g, r = cv2.split(image)

6. 's' converts the image to gray scale as explained in 3rd point and smooths the image using the

blur = cv2.blur(gray, (val, val))

It convolves image with a normalized box filter. It simply takes the average of all the pixels under kernel area and replace the central element. We should specify the width and height of kernel. The second parameter is the size of the kernel. In this program we let you change the size of the kernel by giving a track bar on the top. Smaller size kernel doesn't blur the image so much on the other hand larger size kernel smooth the image more and gives a blur image. The trackbar callback function is to smooth the image when the track bar is moved.

7. 'S' converts the image the gray scale and smooths it using our own implemented convolution. This function doesn't work very well on the vedo.

```
def ownConvole(image):
    kernel = np.ones((7, 7), dtype="float") * (1.0 / (7 * 7))
    (iH, iW) = image.shape[:2]
    (kH, kW) = kernel.shape[:2]
    pad = (kW - 1) // 2
    image = cv2.copyMakeBorder(image, pad, pad, pad, pad,cv2.BORDER_REPLICATE)
    output = np.zeros((iH, iW), dtype="float32")
    for y in np.arange(pad, iH + pad):
      for x in np.arange(pad, iW + pad):
        roi = image[y - pad:y + pad + 1, x - pad:x + pad + 1]
        k = (roi * kernel).sum()
        output[y - pad, x - pad] = k

    output = rescale_intensity(output, in_range=(0, 255))
    output = (output * 255).astype("uint8")
    return output
```

The algorithm selects the x,y from the image and places the kernel at this position. takes the element-wise multiplication of the input image region and the kernel, then sum up the values of these multiplication operations into a single value. Slides the kernel to the next pixel. In the above implementations we use BORDER_REPLICATE to fill the boundary values which basically gives the mirror effect.

8. 'd' reduces the size of the image by half, as we are down sampling it by 2. This is done by taking the height and width of the original image and storing the value of the percentage of pixel u want to retain. This percentage is given to the resize function.

```
resized = cv2.resize(image, dim)
```

9. 'D' we smooth the image and then down sample the image by 2.

In this step we smooth and then resize the image. This would reduce the square boxes which is usually noticed on a low-resolution image.

We apply GaussianBlur to smooth the image as the image is uniformly smoothed

(7,7) is the size of the kernel for GaussianBlur.

```
smoothed = cv2.GaussianBlur(image,(7,7),0)
resized = cv2.resize(smoothed, dim)
```

10. 'x' converts the image to grayscale and convolves with the x derivative. Conversion to gray scale is same as the 3rd step. We choose the x- derivative filter to convolve with the image [[-1,0,1],[-1,0,1],[-1,0,1]]. This filter is used as the kernel to convolve the image. We use the filter2D (gray, -1, kernel) function to convolve the image which takes the image and the kernel as the parameter processing. As the intensity

values would be higher we normalize it back with the help of cv2.normalize(x_derivatve, x_derivatve, 0, 255, cv2.NORM_MINMAX)

10. 'y' converts the image to grayscale and convolves with the y derivative. Conversion to gray scale is same as the 3rd step. We choose the y- derivative filter to convolve with the image [[-1,-1,-1],[0,0,0],[1,1,1]]. This filter is used as the kernel to convolve the image. We use the filter2D (gray, -1, kernel) function to convolve the image which takes the image and the kernel as the parameter processing. As the intensity values would be higher we normalize it back with the help of cv2.normalize(x_derivatve, y_derivatve, 0, 255, cv2.NORM_MINMAX)

11. 'm' converts the image into gray scale then we take the x-derivative and y-derivative and convolve with the image as explained in 9th and 10th steps. We now find the magnitude at each pixel using the formula

sum_xy = derivative**2 + derivative**2
mag = np.sqrt(sum_xy)

or

magni = cv2.magnitude(x_derivatve,y_derivatve)

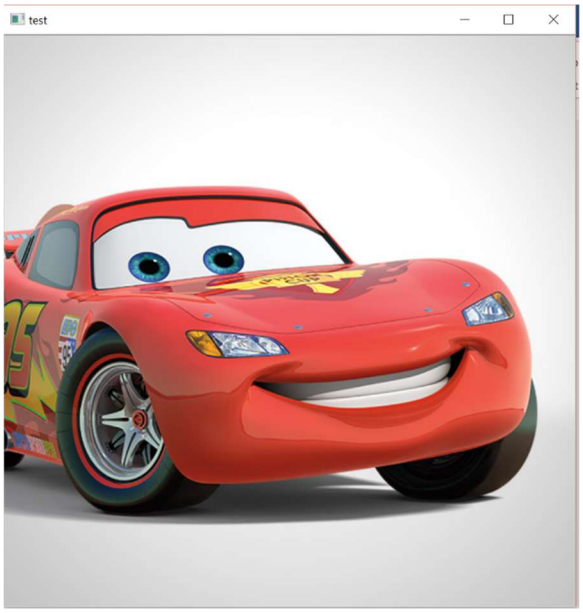
12. 'p' Finds the x and y derivative as seen in the 9th and 10th steps and find the find the gradient vectors magnitude. Use cv2.arrowedLine(gray, (j, i), ((j), (i)), (0, 255, 255)) to draw the arrowed lines.

13. 'r' converts the image to gray scale as in 3rd step rotate it using the track bar to control the rotation angle. The angle is increased by the changing the slider of the track bar. Here cv2.getRotationalMatrix2D and cv2.warpAffine functions are used.

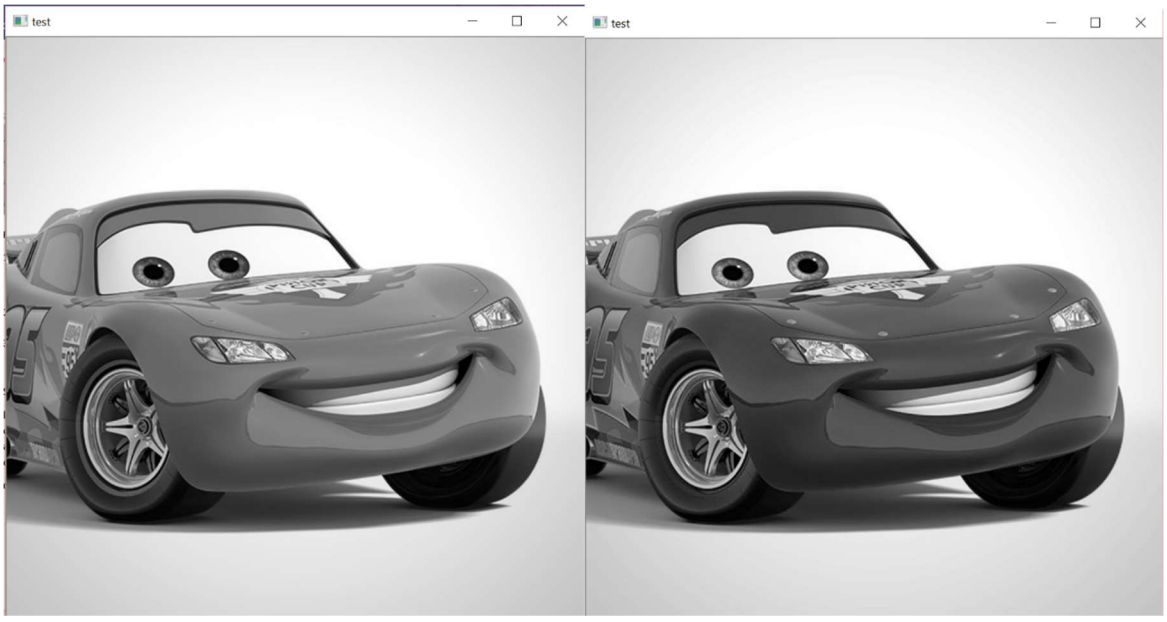
The getRotationalMatrix2D takes the center where u want to rotate the image from(we have chosen the image center for our program), the angle to be rotated at(given by the slider, values range from 0-360^0 and the last parameter is the scale which is 1 in our case. WarpAffine applies an affine transformation to an image.

```
def rotate(val):
   gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
   (h, w) = gray.shape[:2]
   center = (w // 2, h // 2)
   M = cv2.getRotationMatrix2D(center, val, 1.0)
   rotated = cv2.warpAffine(gray, M, (w, h))
   current = rotated
   cv2.imshow('test', rotated)
   return rotated
```

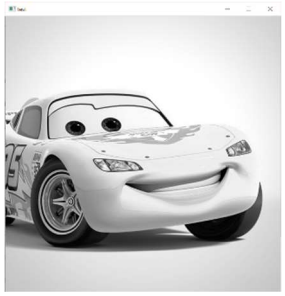
Results and discussion:



Original image.



The 2 figure shows the Gray scale function and the own implementation gray scale function. The gray scale can be taken only from the Y (Luminance) values. **gray = 0.01 * image[:, :, 2] + 0.813 * image[:, :, 1] + 0.177 * image[:, :, 0]**



Red color channel

Green color channel

Blue color channel

The 3 channel together form a RGB OR BRG image. This was done using cv2.split function.

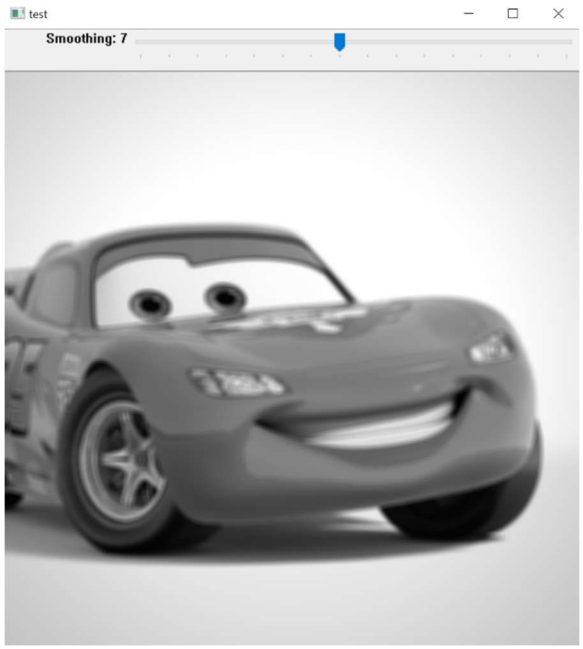
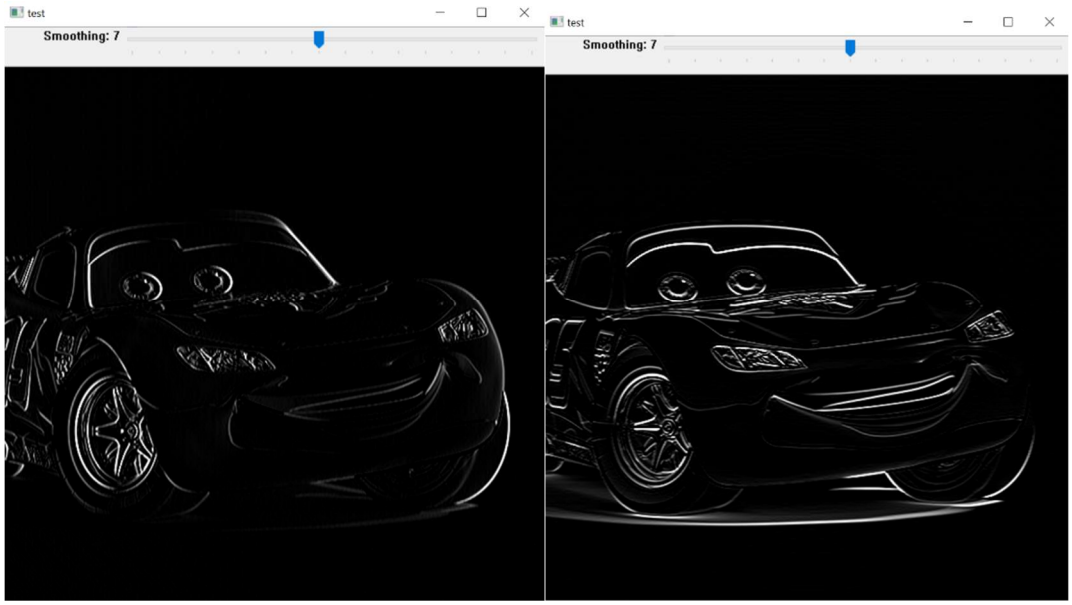


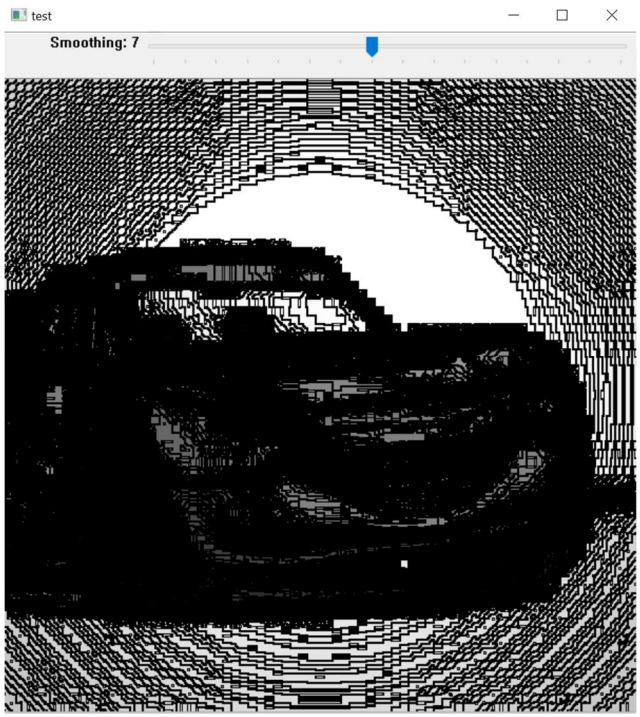
Image applied by changing the value of the smoothing kernel using the trackbar. As the kernel size increases the blur also increases. Here the size of the kernel given was 7. As we keep increasing the trackbar we find the image to be more blurry.



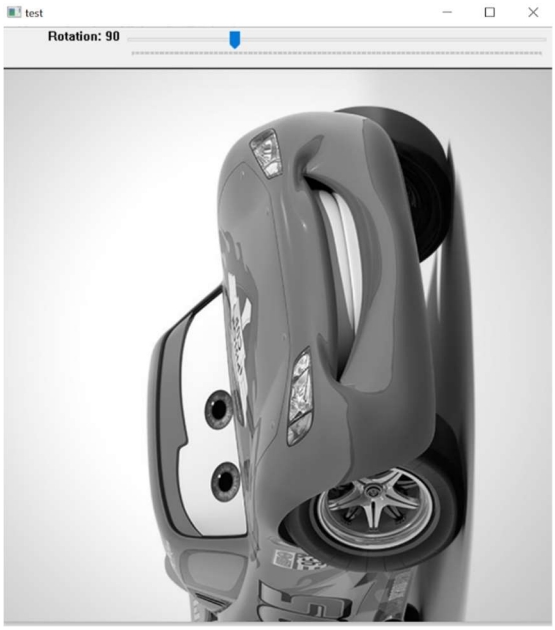
X derivative of the original image on left and Y derivative of original image on right.

These 2 images are similar to that of the sobel x and sobel y derivative. Central derivative of size 3*3 was used to find the x and y derivative of the image. We can see that using both the derivatives we can find more accurate edges of the image instead of just using x or y derivative.

Magnitude of the image. Once the derivative is found using any filter we can also find the magnitude of change. The above image shows the magnitude of the image once the x and y derivative is taken.



Gradient vectors using cv2.cartToPolar(x_derivatve, y_derivatve)
cv2.arrowedLine was used to draw these gradient vectors.



Rotated by 90^0

In the rotation the trackbar lets you adjust the rotation angle. It rotates the image by the center in this program as we have specified.

Reference:

- https://docs.opencv.org/
- https://www.pyimagesearch.com/
- https://stackoverflow.com/
- https://medium.com/@manivannan_data/set-trackbar-on-image-using-opencv-python-58c57fbee1ee