CS512 - Assignment 1

Mohammadreza Asherloo
Department of Mechanical, Materials and Aerospace Engineering
Illinois Institute of Technology

October 11, 2020

Problem 1:

Statement:

Opening an image based on it's name or capturing video from camera and modify them based on the key input from user

Solution:

For running the program, we can define an argument. If the user passes the name of the image, program opens that image. If the user does not pass any arguments, program starts capturing continuous images from camera and processing them. We need to constantly wait for the key input from user which needs different implementation for static or live images.

Details:

For opening a program using CMD in windows, user needs to type:

```
python program.py argument
```

here if the program finds the argument, it will open the image whose name is passed to the program through the argument:

```
if len(sys.argv) == 2:
    filename = sys.argv[1]
    image = cv2.imread(filename)
    cv2.imshow(winName, image)
```

If there isn't any argument passed, program tries capturing images:

```
elif len(sys.argv) < 2:
    winName = "Image"
    cam = cv2.VideoCapture(0)
```

```
    while True:
        retval, frame = cam.read()
        cv2.imshow(winName, frame)
```

Regarding taking key inputs from user, we will have two different implementations for static and live images.
For static image we can write after showing the image:

```
while True:
        key = cv2.waitKey(0)
        if key == ord('i'):
            cv2.destroyAllWindows()
            image = cv2.imread(filename)
            cv2.imshow(winName, image)
```

In this case, after showing the image program goes in the "while" loop and by "waitKey(0)" function waits indefinitely for the user input. After taking the input, program modifies the image based on that input: For live images, we need to capture images continuously and we

Input	Function
h	Show help
i	Reload the original image
w	Save the processed image
g or G	Convert to grayscale
s or S	Smooth the image
d	Downsample the image without smoothing
D	Downsample the image with smoothing
x	Convert to grayscale and show x derivative
y	Convert to grayscale and show y derivative
m	Show magnitude of gradient
r	Rotate the grayscale version

need a while loop to process the images after taking the input from user:

```
elif len(sys.argv) < 2:
    winName = "Image"
    cam = cv2.VideoCapture(0)
    while True:
        retval, frame = cam.read()
        s_key = cv2.waitKey(1)
        while s_key == ord('g'):
            retval, frame = cam.read()
            frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
            cv2.imshow(winName, frame)
```

Program waits every 1 ms for an input and this approach allows it to capture continuous

images from camera

Results and discussion:

In the manner of opening and taking input from user, program works totally fine but with one weakness which is the wait time for taking input in the live image mode.
Because the time set for waiting is 1 ms, sometimes pressing the key is before or after this time so we won't see the results from program.

Problem 2:

Statement:

Finding different derivatives of the image (x, y and magnitude of gradients)

Solution:

For x derivative we will use this kernel: $\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$

For y derivative we will use this kernel: $\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$

For magnitude of gradients we will compute x and y derivatives and then compute the magnitude of them for each pixel.

Details:

Program waits for the input and when it gets "x" as the input, it will compute the x derivative of image by convolution of image with the appropriate kernel. After the convolution, it will normalize the results and then shows it.
After showing the result in the live image mode, program waits for the key input from user and if it is "i" it will recapture the original image and if it is "q" it will quit the loop and closes the program (these two inputs are applied in all the loops and modification algorithms).

Live image mode:

```
while s_key == ord('x'):
    retval, frame = cam.read()
    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    kernel = np.array([[-1, 0, 1],
                       [-2, 0, 2],
                       [-1, 0, 1]])
    frame = cv2.normalize(cv2.filter2D(frame, -1, kernel),
                          None, alpha=0, beta=255,
```

```
                            norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_32F)
        cv2.imshow(winName, frame)
        keyy = cv2.waitKey(1)
        if keyy == ord('i'):
            break
        if keyy == ord('q'):
            quit = ord('q')
            break
    while s_key == ord('y'):
        retval, frame = cam.read()
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        kernel = np.array([[-1, -2, -1],
                           [0, 0, 0],
                           [1, 2, 1]])
        frame = cv2.normalize(cv2.filter2D(frame, -1, kernel),
                            None, alpha=0, beta=255,
                            norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_32F)
        cv2.imshow(winName, frame)
        keyy = cv2.waitKey(1)
        if keyy == ord('i'):
            break
        if keyy == ord('q'):
            quit = ord('q')
            break
    while s_key == ord('m'):
        retval, frame = cam.read()
        sx = cv2.Sobel(frame,cv2.CV_32F,1,0,ksize=5)
        sy = cv2.Sobel(frame,cv2.CV_32F,0,1,ksize=5)
        frame = np.hypot(sx,sy)/255
        cv2.imshow(winName, frame)
        keyy = cv2.waitKey(1)
        if keyy == ord('i'):
            break
        if keyy == ord('q'):
            quit = ord('q')
            break
```

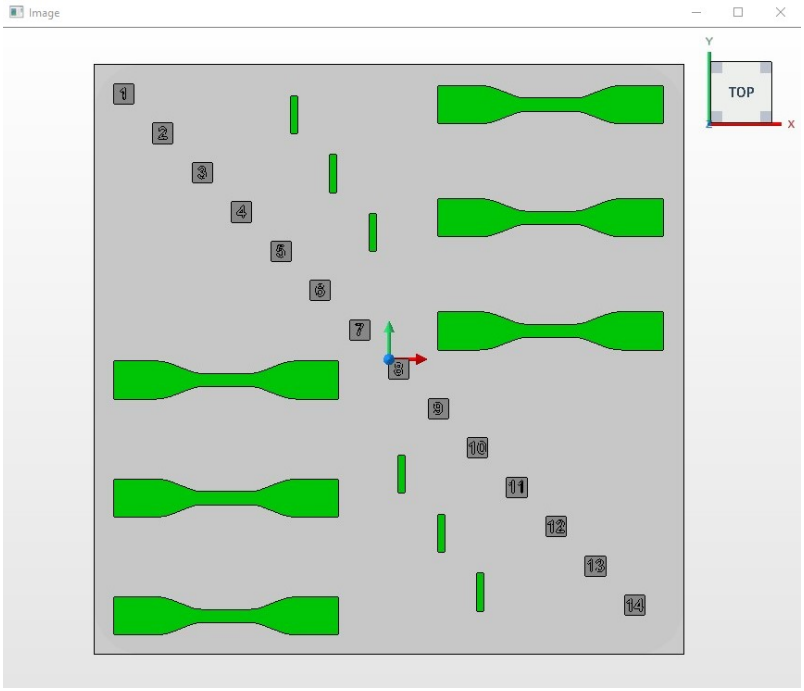
Static image mode:

```
    if key == ord('x'):
        image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        kernel = np.array([[-1, 0, 1],
                           [-2, 0, 2],
                           [-1, 0, 1]])
        dst = cv2.normalize(cv2.filter2D(image_gray, -1, kernel),
                            None, alpha=0, beta=255,
```

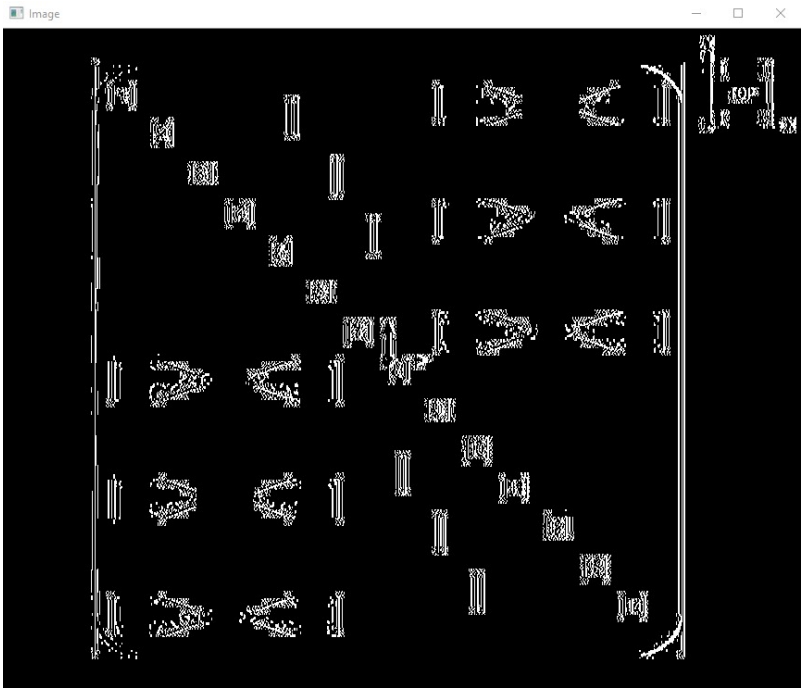
```
                            norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_32F)
        image = dst
        cv2.imshow(winName, dst)
    if key == ord('y'):
        image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        kernel = np.array([[-1, -2, -1],
                           [0, 0, 0],
                           [1, 2, 1]])
        dst = cv2.normalize(cv2.filter2D(image_gray, -1, kernel),
                            None, alpha=0, beta=255,
                            norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_32F)
        image = dst
        cv2.imshow(winName, dst)
    if key == ord('m'):
        sx = cv2.Sobel(image,cv2.CV_32F,1,0,ksize=5)
        sy = cv2.Sobel(image,cv2.CV_32F,0,1,ksize=5)
        sobel = np.hypot(sx,sy)/255
        image = sobel
        cv2.imshow(winName, sobel)
```

Results and discussion:

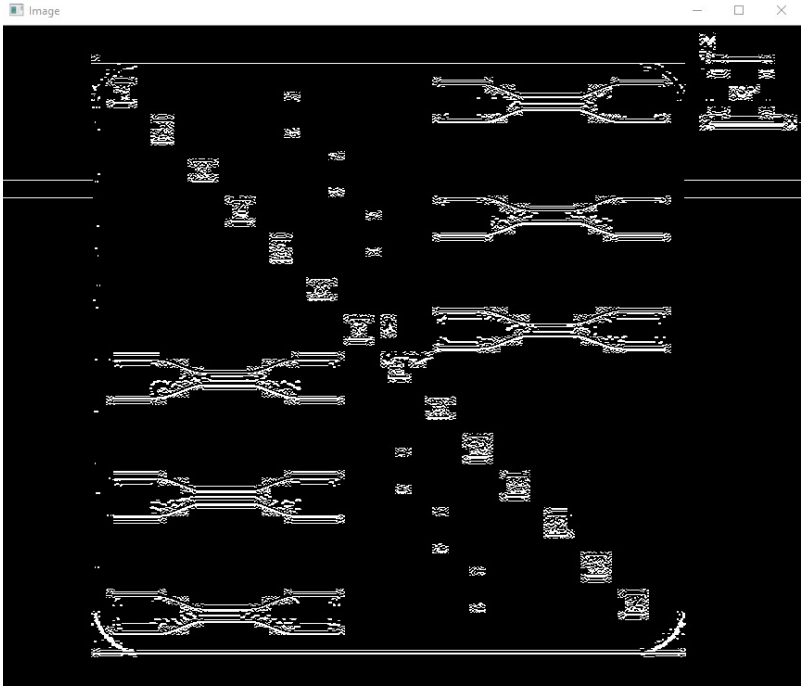
Original image:



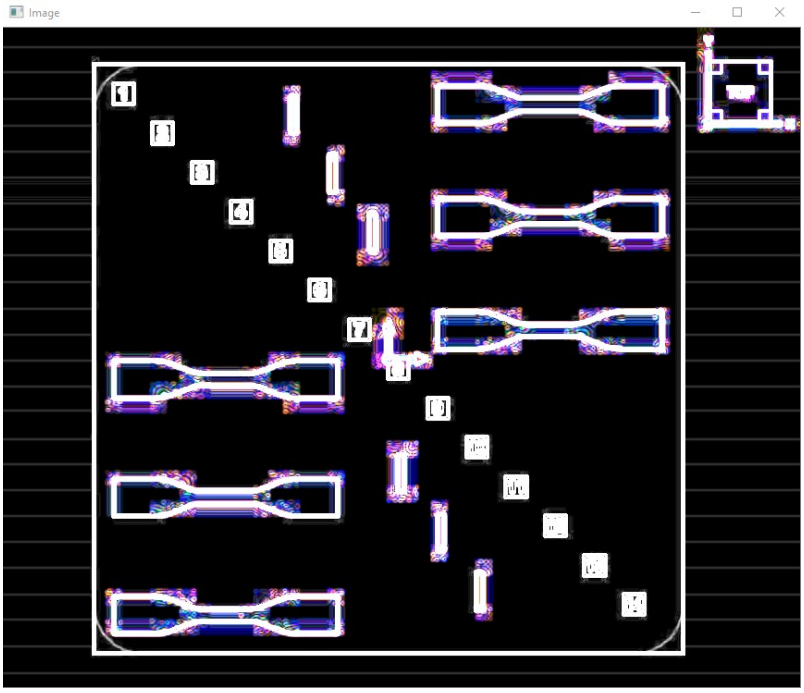
X derivative:



Y derivative:



Magnitude of gradients:



Problem 3:

Statement:

Rotating the image controlled with a trackbar

Solution:

Rotation of the image using getRotationMatrix2D and warpAffine has a problem which is image will be cropped from sides to fit in the fixed size window. But with my algorithm we can change the size of the window to show the complete rotated image.

Details:

I have defined a function to rotate the image based on the position of the trackbar produced after taking the input "r" from the user.
This algorithm will compute the cosine and sine of the image sides after rotation and changes the size of the window based on these values.
For static image:

```
def rotate(n):
    global image
    height, width = image.shape[:2]
    image_center = (width/2, height/2)
```

```
    rotation_mat = cv2.getRotationMatrix2D(image_center, n, 1.)
    abs_cos = abs(rotation_mat[0,0])
    abs_sin = abs(rotation_mat[0,1])
    bound_w = int(height * abs_sin + width * abs_cos)
    bound_h = int(height * abs_cos + width * abs_sin)
    rotation_mat[0, 2] += bound_w/2 - image_center[0]
    rotation_mat[1, 2] += bound_h/2 - image_center[1]
    rotated_mat = cv2.warpAffine(image, rotation_mat, (bound_w, bound_h))
    cv2.imshow(winName, rotated_mat)

if key == ord('r'):
    cv2.createTrackbar("r", winName, 0, 360, rotate)
```

For live images:

```
while s_key == ord('r'):
    retval, frame = cam.read()
    n = cv2.getTrackbarPos("r", winName)
    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    height, width = frame.shape[:2]
    image_center = (width/2, height/2)
    rotation_mat = cv2.getRotationMatrix2D(image_center, n, 1.)
    abs_cos = abs(rotation_mat[0,0])
    abs_sin = abs(rotation_mat[0,1])
    bound_w = int(height * abs_sin + width * abs_cos)
    bound_h = int(height * abs_cos + width * abs_sin)
    rotation_mat[0, 2] += bound_w/2 - image_center[0]
    rotation_mat[1, 2] += bound_h/2 - image_center[1]
    frame = cv2.warpAffine(frame, rotation_mat, (bound_w, bound_h))
    cv2.imshow(winName, frame)
    keyy = cv2.waitKey(1)
    if keyy == ord('i'):
        break
    if keyy == ord('q'):
        quit = ord('q')
        break
```

Results and discussion:

Original image with a trackbar to controll the rotation:

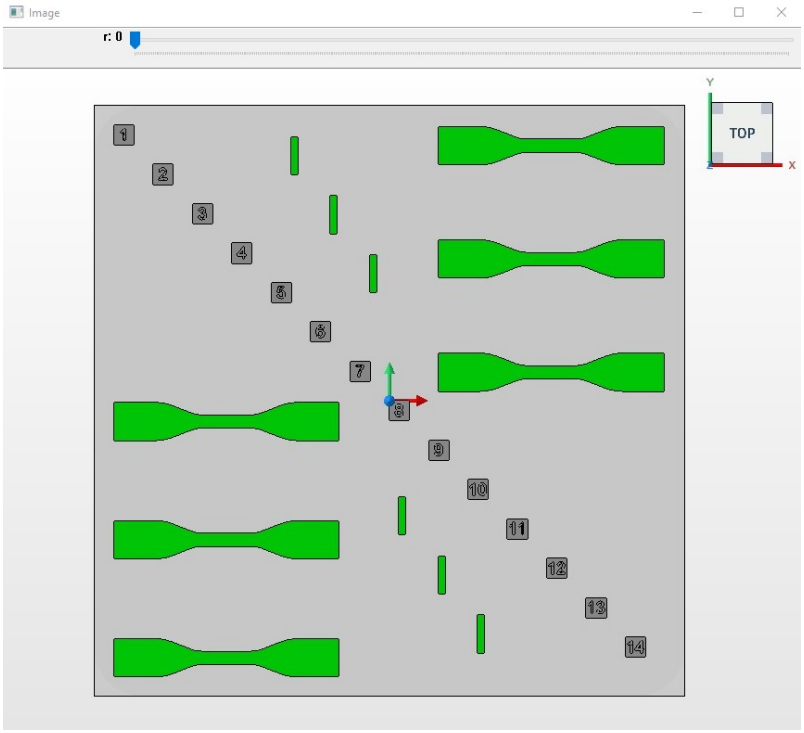


Image after 15 degree rotation:

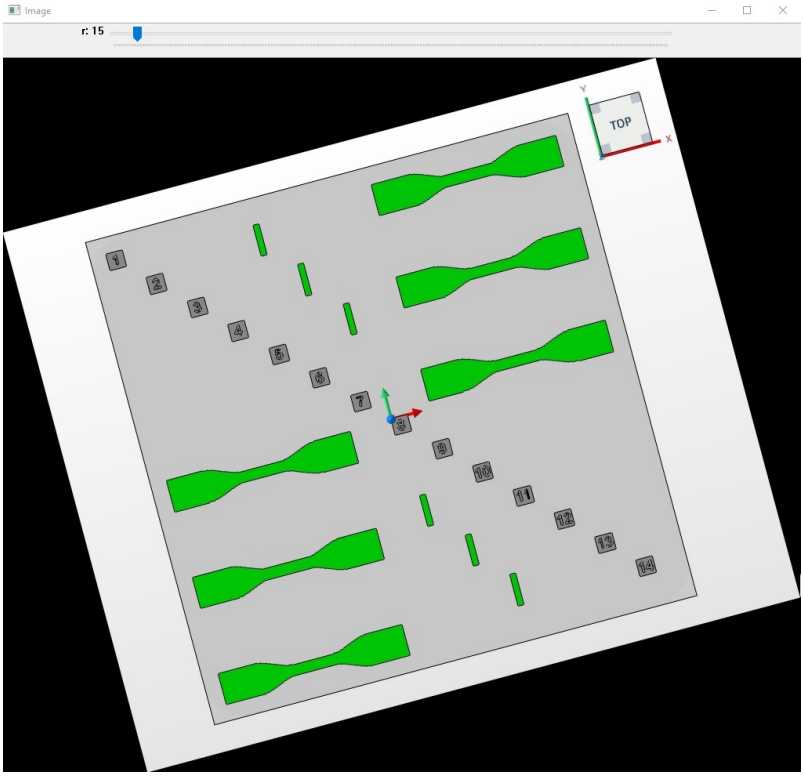
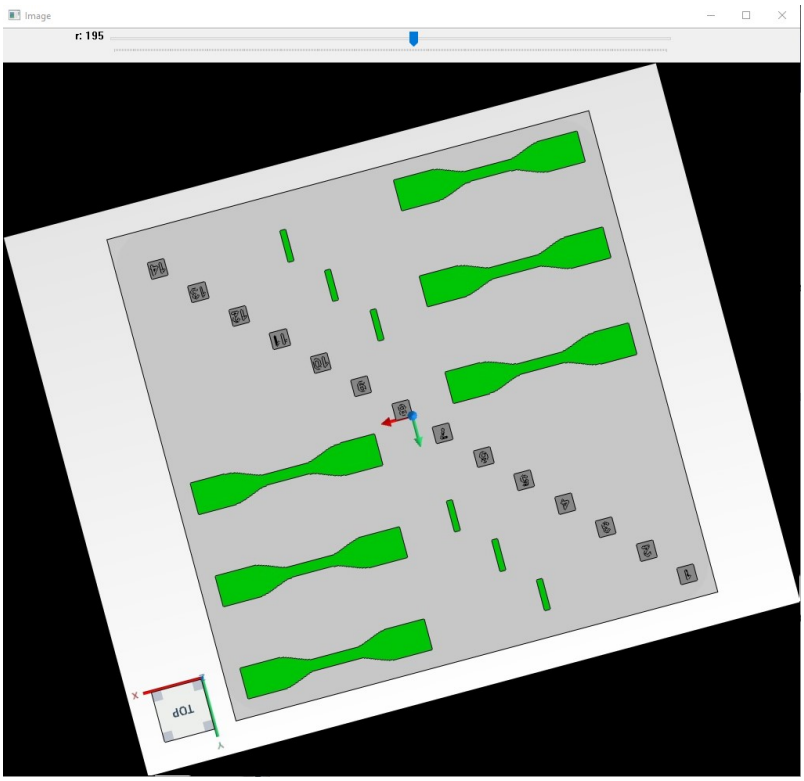


Image after 195 degree rotation:



The only weakness that I could observe was the change in the size of trackbar when the window size changes based on the image.

Problem 4:

Statement:

Converting image to grayscale using OpenCV functions and using my own function

Solution:

For converting the image to grayscale we can multiply the image color channels with the following matrix:
$\begin{bmatrix} 0.2989 & 0.5870 & 0.1140 \end{bmatrix}$

Details:

Function for converting to grayscale:

```
def rgb2gray(rgb):
    gray = np.dot(rgb[...,:3], [0.2989, 0.5870, 0.1140])
    return cv2.normalize(gray, None, alpha=0, beta=1, norm_type=cv2.NORM_MINMAX)
```

For live image:

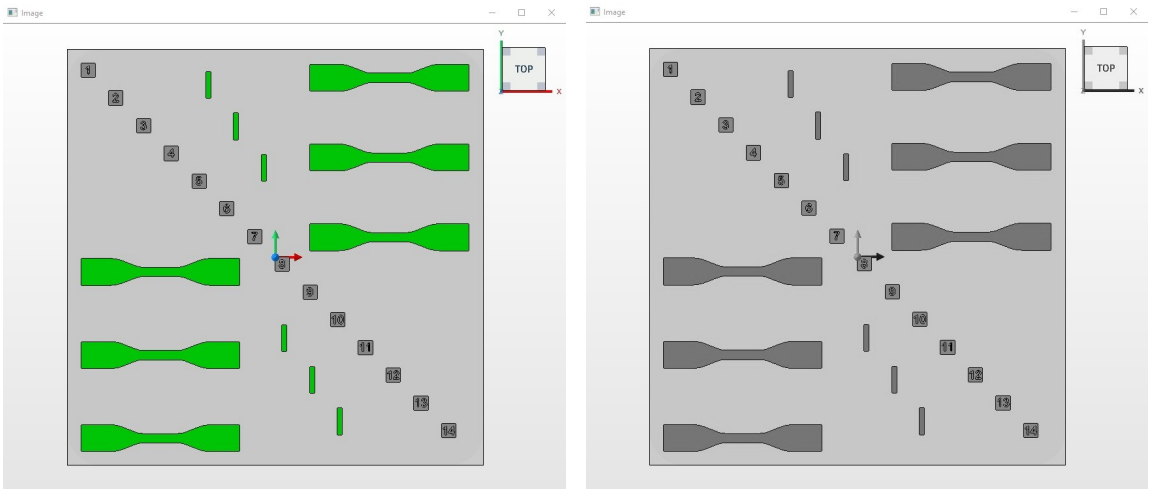
```
while s_key == ord('G'):
    retval, frame = cam.read()
    frame = rgb2gray(frame)
    cv2.imshow(winName, frame)
    keyy = cv2.waitKey(1)
    if keyy == ord('i'):
        break
    if keyy == ord('q'):
        quit = ord('q')
        break
```

For static image:

```
if key == ord('G'):
    image_gray = rgb2gray(image)
    image = image_gray
    cv2.imshow(winName, image_gray)
```

Results and discussion:

Original image vs Gray scale version:



Problem 5:

Statement:

Smoothing the image controlled with trackbar

Solution:

After any change in trackbar's position, we will make a smoothing kernel. For example, if the trackbar is on 5, we will have a 5 by 5 smoothing kernel.
After making the kernel, we will compute the convolution of image with that kernel to smooth the image.
We can either use Cython to make our own convolution layer or we can use the OpenCV function to compute the results.

Details:

For static image:

```
def slideHandler(n):
    global image
    img = cv2.imread("1.jpg")
    image_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    cv2.imshow(winName, image_gray)
    if n==0:
        cv2.imshow(winName, image_gray)
        image = image_gray
    else:
        kernel = np.ones((n,n), np.float32)/(n*n)
        dst = cv2.filter2D(image_gray, -1, kernel)
        cv2.imshow(winName, dst)
        image = dst

def slideHandlerOwnConvolution(n):
    global image
    img = cv2.imread("1.jpg")
    image_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    cv2.imshow(winName, image_gray)
    if n==0:
        cv2.imshow(winName, image_gray)
        image = image_gray
    else:
        kernel = np.ones((n,n), np.float32)/(n*n)
        dst = convolve2d(image_gray, kernel)
        cv2.imshow(winName, dst)
        image = dst

if key == ord('s'):
    cv2.createTrackbar("s", winName, 0, 255, slideHandler)

if key == ord('S'):
    cv2.createTrackbar("s", winName, 0, 255, slideHandlerOwnConvolution)
```

For live images we can't use the callback feature of trackbars in OpenCV. We need to create

a trackbar once and read it's position for every frame. In addition, we need a function that does nothing to pass it to the callback of trackbar:

```
def nothing(x):
    pass

if s_key == ord('s') or s_key == ord('S'):
    cv2.createTrackbar("s", winName, 0, 255, nothing)

while s_key == ord('s'):
    retval, frame = cam.read()
    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    n = cv2.getTrackbarPos("s", winName)
    if n==0:
        pass
    if n!=0:
        kernel = np.ones((n,n), np.float32)/(n*n)
        frame = cv2.filter2D(frame, -1, kernel)
    cv2.imshow(winName, frame)
    keyy = cv2.waitKey(1)
    if keyy == ord('i'):
        break
    if keyy == ord('q'):
        quit = ord('q')
        break
```

Results and discussion:

Our own convolution is significantly slower than the OpenCV version even when it is implemented by cython.

Original image:

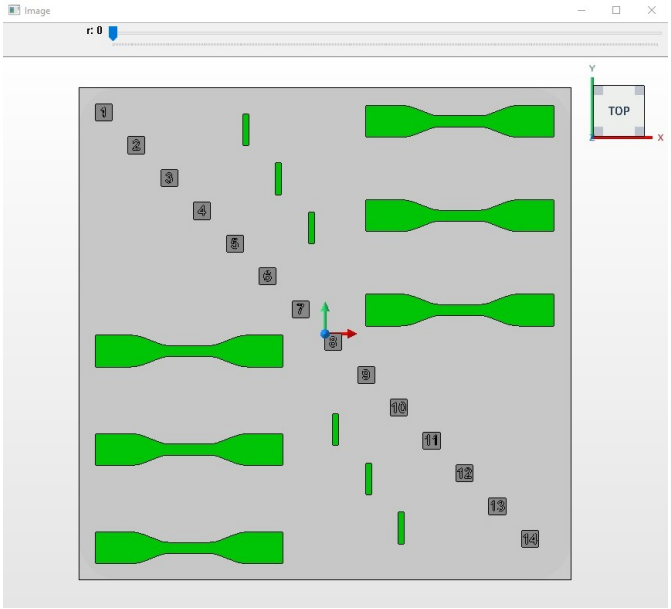


Image after smoothing with 5 by 5 kernel:

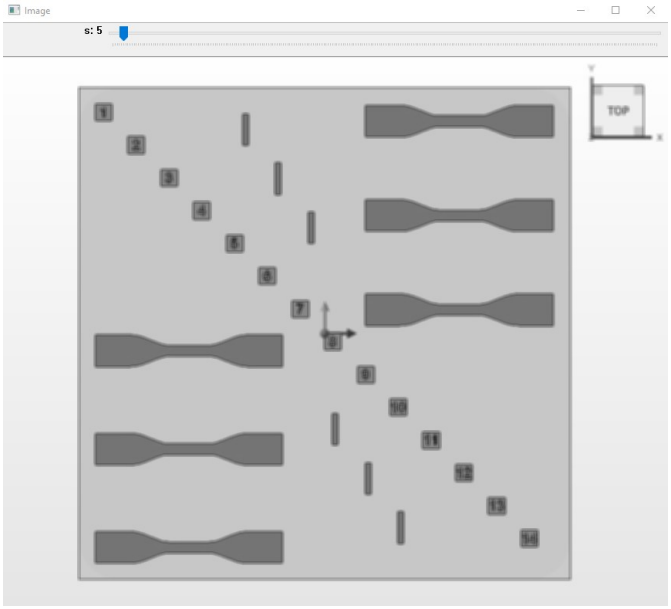


Image after smoothing with 50 by 50 kernel:

Problem 6:

Statement:

Changing the color channel of image Solution:

We can do this by removing the desired color channel from the image every time "c" is pressed.
We keep track of how many times "c" is being pressed by defining variable i Details:

For static image:

```
i = 0

if key == ord('c'):
    image_oc = np.zeros(shape=image.shape, dtype=np.uint8)
    image_oc[:,:,i] = image[:,:,i]
    cv2.imshow(winName, image_oc)
    if i==2:
        i = 0
    else:
        i = i+1
```

For live image:

```
i = 0

while s_key == ord('c'):
    retval, frame = cam.read()
    frame_oc = np.zeros(shape=frame.shape, dtype=np.uint8)
    frame_oc[:,:,i] = frame[:,:,i]
    cv2.imshow(winName, frame_oc)
    keyy = cv2.waitKey(1)
    if keyy == ord('c'):
        if i==2:
            i = 0
        else:
            i = i+1

    if keyy == ord('i'):
                break
    if keyy == ord('q'):
        quit = ord('q')
        break
```

Results and discussion:

Original image, blue channel only, green channel only and red channel only: