COMPUTER VISION

PRACTICAL CODES

KISHORE KUMAR.S

192224044

Q1.

AIM :

To Perform basic Image Handling and processing operations on the image. Read an image in python and Convert an Image to Grayscalej

SOURCE CODE :

```
import cv2
image path = r""E:\kishore\image(1).jpg""
image = cv2.imread(image_path)
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
cv2.imshow('Original Image', image)
cv2.imshow('Grayscale Image', gray_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

INPUT :

OUTPUT :

Q2.

AIM :

To perform basic Image Handling and processing operations on the image.Read an image in python and Convert an Image to Blur using GaussianBlur.

SOURCE CODE :

```
import cv2
image_path = " E:\kishore\image(1).jpg "
image = cv2.imread(image_path)
blurred_image = cv2.GaussianBlur(image, (5, 5), 0)
cv2.imshow('Original Image', image)
cv2.imshow('Blurred Image', blurred_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

INPUT:

OUTPUT:

Q3.

AIM :

To perform Basic Operations to Convert image to show outline Canny function.

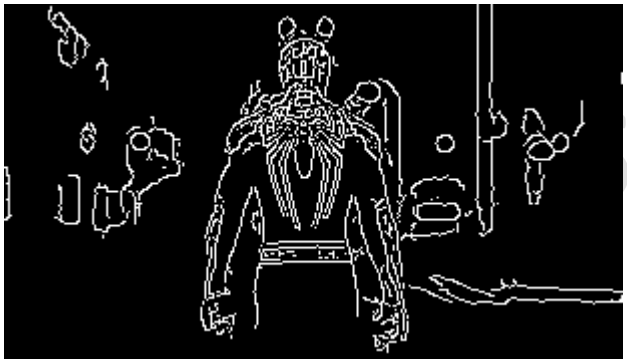
SOURCE CODE :

```
import cv2
import numpy as np
kernel = np.ones((5,5),np.uint8)
print(kernel)
path = " E:\kishore\image(1).jpg "
img =cv2.imread(path)
imgGray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
imgBlur = cv2.GaussianBlur(imgGray,(7,7),0)
imgCanny = cv2.Canny(imgBlur,100,200)
desired_width = 800
desired_height = 600
```

```
img_resized = cv2.resize(imgCanny, (desired_width, desired_height))
cv2.imshow("Img Canny",img_resized)
cv2.waitKey(0)
```

INPUT:

OUTPUT:



Q4.

AIM :

To perform basic Image Handling and processing operations on the image Read an image in python and Dilate an Image using Dilate function

SOURCE CODE :

```
import cv2
import numpy as np
```

```
kernel = np.ones((5,5),np.uint8)
print(kernel)
path = " E:\kishore\image(1).jpg "
img =cv2.imread(path)
imgGray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
imgBlur = cv2.GaussianBlur(imgGray,(7,7),0)
imgCanny = cv2.Canny(imgBlur,100,200)
imgDilation = cv2.dilate(imgCanny,kernel , iterations = 10)
imgEroded = cv2.erode(imgDilation,kernel,iterations=2)
desired_width = 800
desired_height = 600
img_resized = cv2.resize(imgEroded, (desired_width, desired_height))
cv2.imshow("Img Erosion",img_resized)
cv2.waitKey(0)
```

INPUT:

OUTPUT:

Q5.

AIM :

To Perform basic Image Handling and processing operations on the image Read an image in python and Erode an Image using erode function

SOURCE CODE :

```
import cv2
import numpy as np
kernel = np.ones((5,5),np.uint8)
print(kernel)
path = " E:\kishore\image(1).jpg "
img =cv2.imread(path)
imgGray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
imgBlur = cv2.GaussianBlur(imgGray,(7,7),0)
imgCanny = cv2.Canny(imgBlur,100,200)
imgDilation = cv2.dilate(imgCanny,kernel , iterations = 10)
imgEroded = cv2.erode(imgDilation,kernel,iterations=2)
desired_width = 800
desired_height = 600
img_resized = cv2.resize(imgEroded, (desired_width, desired_height))
cv2.imshow("Img Erosion",img_resized)
cv2.waitKey(0)
```

INPUT:

OUTPUT:

Q6.

AIM :

Perform basic video processing operations on the captured video• Read captured video in python and display the video, in slow motion and in fast motion.

SOURCE CODE :

```
import cv2
import numpy as np
cap = cv2.VideoCapture("E:\kishore\video(1).mp4 ")
if (cap.isOpened()== False):
   print("Error opening video file")
while(cap.isOpened()):
   ret, frame = cap.read()
   if ret == True:
      cv2.imshow('Frame', frame)
      if cv2.waitKey(250) & 0xFF == ord('q'):
         break
   else:
      break
cap.release()
cv2.destroyAllWindows()
```

OUTPUT:

Q7.

AIM :

Capture video from web Camera and Display the video, in slow motion and in fast motion operations on the captured video.

SOURCE CODE :

```
import cv2
cap = cv2.VideoCapture(0)
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
fps = cap.get(cv2.CAP_PROP_FPS)
path = "0"
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
output = cv2.VideoWriter(path, fourcc, 2,(width, height))
while True:
   ret, frame = cap.read()
   cv2.imshow("frame", frame)
   output.write(frame)
   k = cv2.waitKey(24)
   if k == ord("q"):
```

```
    break
cap.release()
output.release()
cv2.destroyAllWindows()
```

OUTPUT:

Q8.

AIM :

To perform Scaling an image to its Bigger and Smaller sizes

SOURCE CODE :

```
import cv2
import numpy as np
kernel = np.ones((5,5),np.uint8)
img = cv2.imread("E:\kishore\image(1).jpg ",cv2.IMREAD_COLOR)
img = cv2.resize(img,(600,600))
cv2.imshow("image",img)
cv2.waitKey(0)
```

OUTPUT:

Q9.

AIM :

Perform Rotation of an image to clockwise and counter clockwise direction.

SOURCE CODE :

```
import cv2
path =" E:\kishore\image(1).jpg "
src = cv2.imread(path)
window_name = 'Image'
image = cv2.rotate(src, cv2.ROTATE_180)
img = cv2.resize(image,(600,600))
cv2.imshow(window_name, img)
cv2.waitKey(0)
```

OUTPUT:

Q10.

AIM :

The Aim of the Experiment is to perform Rotation of an image import cv2 along 90 degree.

SOURCE CODE :

```
path = " E:\kishore\image(1).jpg "
src = cv2.imread(path)
window_name = 'Image'
image = cv2.rotate(src, cv2.ROTATE_90_COUNTERCLOCKWISE)
img = cv2.resize(image,(600,600))
cv2.imshow(window_name, img)
cv2.waitKey(0)
```

OUTPUT:

Q11.

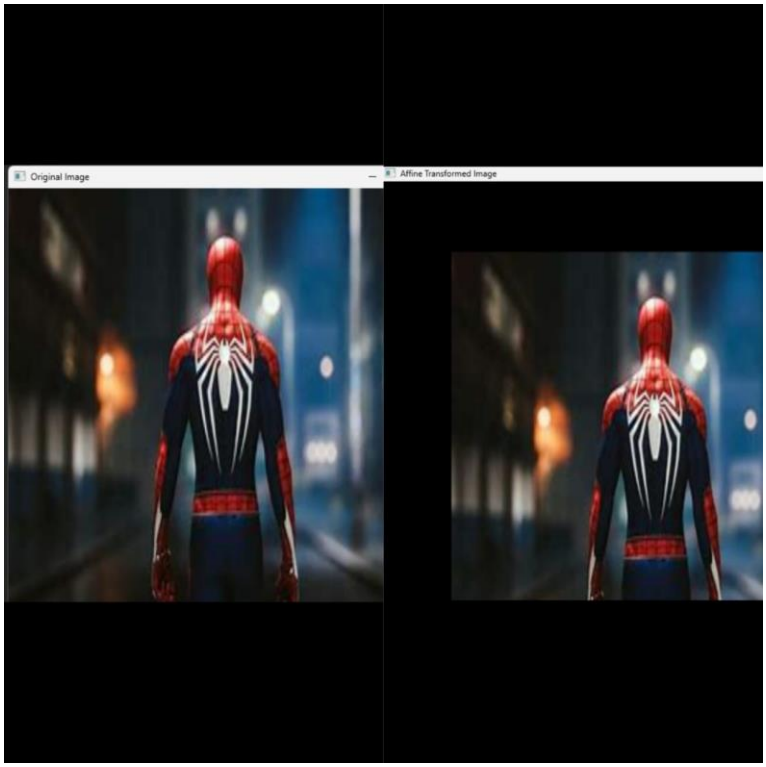
AIM :

Perform Affine Transformation on the image.

SOURCE CODE :

```
import cv2
import numpy as np
image_path = " E:\kishore\image(1).jpg "
image = cv2.imread(image_path)
rows, cols, _ = image.shape
transformation_matrix = np.float32([[1, 0, 50], [0, 1, 30]])
affine_image = cv2.warpAffine(image, transformation_matrix, (cols, rows))
img = cv2.resize(image,(600,600))
img2 = cv2.resize(affine_image,(600,600))
cv2.imshow('Original Image', img)
cv2.imshow('Affine Transformed Image', img2)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:



Q12.

AIM :

Perform Perspective Transformation on the image.

SOURCE CODE :

```
import cv2
import numpy as np
img = cv2.imread("E:\kishore\image(1).jpg”)
rows,cols,ch = img.shape
pts1 = np.float32([[56,65],[368,52],[28,387],[389,390]])
pts2 = np.float32([[100,50],[300,0],[0,300],[300,300]])
M = cv2.getPerspectiveTransform(pts1,pts2)
dst = cv2.warpPerspective(img,M,(cols, rows))
cv2.imshow('Transformed Image', dst)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:

Q13.

AIM :

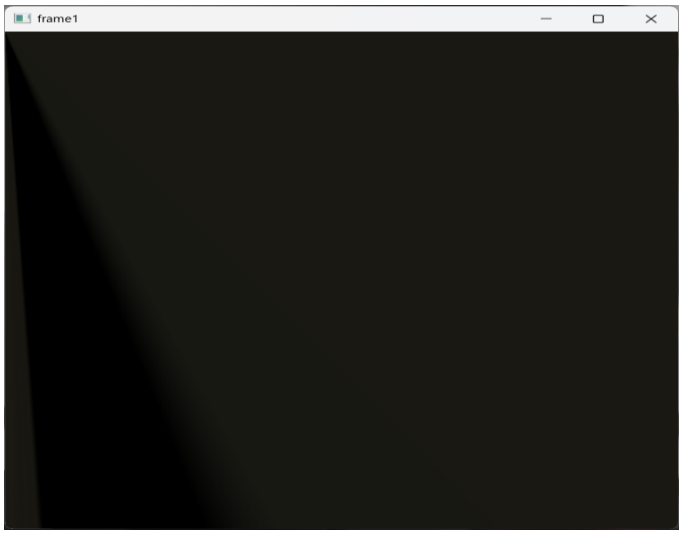
Perform Perspective Transformation on the Video.

SOURCE CODE :

```
import cv2
import numpy as np

cap = cv2.VideoCapture(0)

while True:

        ret, frame = cap.read()

        pts1 = np.float32([[0, 260], [640, 260],
                                        [0, 400], [640, 400]])
        pts2 = np.float32([[0, 0], [400, 0],
                                        [0, 640], [400, 640]])

        matrix = cv2.getPerspectiveTransform(pts1, pts2)
        result = cv2.warpPerspective(frame, matrix, (500, 600))

        cv2.imshow('frame', frame)
        cv2.imshow('frame1', result)

        if cv2.waitKey(24) == 27:
                break
```

```
cap.release()
cv2.destroyAllWindows()
```

OUTPUT:



Q14.

AIM :

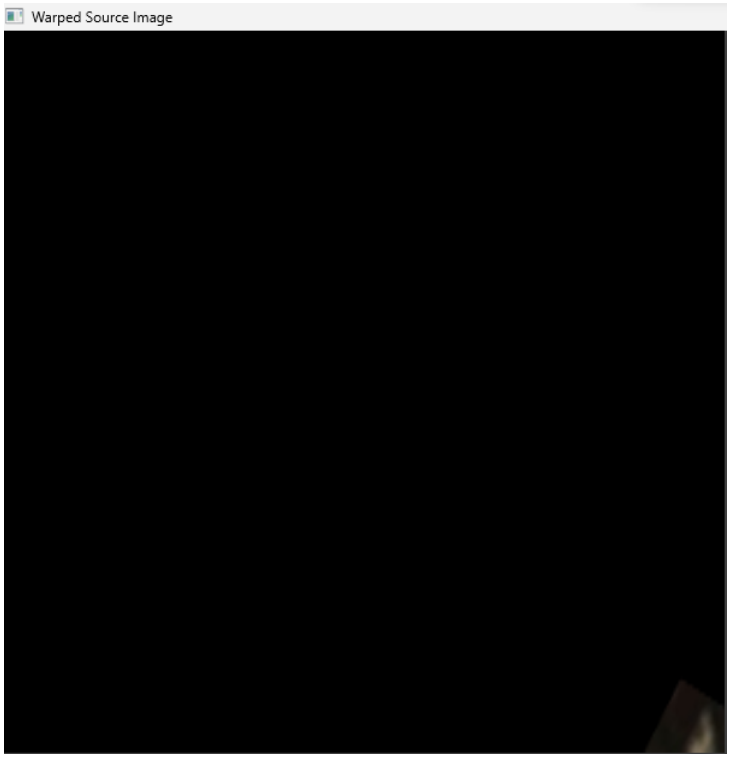
Perform transformation using Homography matrix.

SOURCE CODE

```
import cv2
import numpy as np
im_src = cv2.imread(r"E:\kishore\image(1).jpg)
pts_src = np.array([[141, 131], [480, 159], [493, 630],[64, 601]])
im_dst = cv2.imread(r"C:\Users\Vigneshwaran\Desktop\p.jpeg")
pts_dst = np.array([[318, 256],[534, 372],[316, 670],[73, 473]])
h, status = cv2.findHomography(pts_src, pts_dst)
im_out = cv2.warpPerspective(im_src, h, (im_dst.shape[1],im_dst.shape[0]))
img = cv2.resize(im_src,(600,600))
img1 = cv2.resize(im_dst,(600,600))
img2 = cv2.resize(im_out,(600,600))
cv2.imshow("Source Image", img)
cv2.imshow("Destination Image", img1)
```

```
cv2.imshow("Warped Source Image", img2)
cv2.waitKey(0)
```

OUTPUT:



Q15.

AIM :

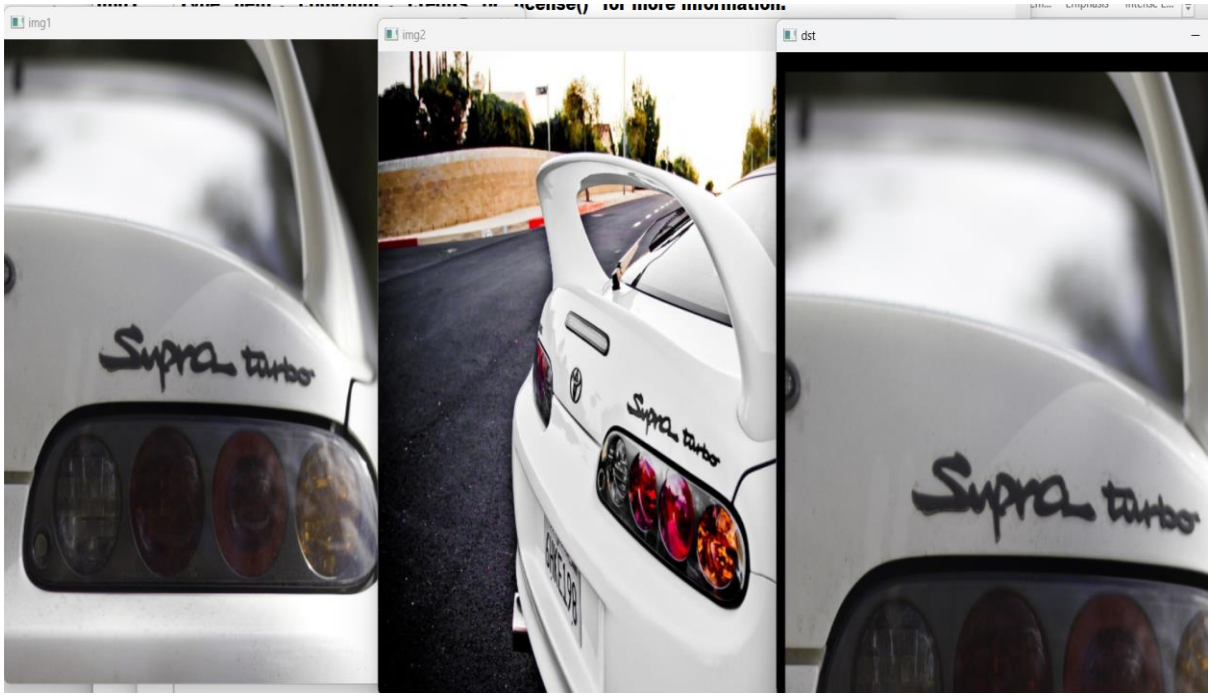
Perform transformation using Direct Linear Transformation.

SOURCE CODE :

```
import cv2
import numpy as np
img1 = cv2.imread(r" E:\kishore\image(0).jpg ")
img2 = cv2.imread(r" E:\kishore\image(1).jpg ")
pts1 = np.array([[50, 50], [200, 50], [50, 200], [200, 200]])
pts2 = np.array([[100, 100], [300, 100], [100, 300], [300, 300]])
H, _ = cv2.findHomography(pts1, pts2)
dst = cv2.warpPerspective(img1, H, (img2.shape[1], img2.shape[0]))
img = cv2.resize(img1,(600,600))
```

```
img3 = cv2.resize(img2,(600,600))
img4 = cv2.resize(dst,(600,600))
cv2.imshow('img1', img)
cv2.imshow('img2', img3)
cv2.imshow('dst', img4)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:



Q16.

AIM :

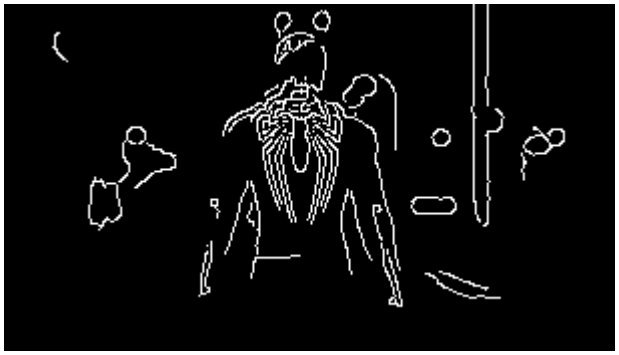
Perform Edge detection using canny method

SOURCE CODE :

```
import cv2
img = cv2.imread(r “E:\kishore\image(1).jpg ")
cv2.imshow('Original', img)
cv2.waitKey(0)
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
img_blur = cv2.GaussianBlur(img_gray, (3,3), 0)
edges = cv2.Canny(image=img_blur, threshold1=100, threshold2=200)
cv2.imshow('Canny Edge Detection', edges)
```

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

OUTPUT:



Q17.

AIM :

Perform Edge detection using Sobel Matrix along X axis

SOURCE CODE :

```
import cv2
img = cv2.imread(r" E:\kishore\image(1).jpg ")
cv2.imshow('Original', img)
cv2.waitKey(0)
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
img_blur = cv2.GaussianBlur(img_gray, (3,3), 0)
sobelx = cv2.Sobel(src=img_blur, ddepth=cv2.CV_64F, dx=1, dy=0, ksize=5)
cv2.imshow('Sobel X', sobelx)
cv2.waitKey(0)
```

OUTPUT:

Q18.

AIM :

Perform Edge detection using Sobel Matrix along Y axis

SOURCE CODE :

```
import cv2
img = cv2.imread(r" E:\kishore\image(1).jpg ")
cv2.imshow('Original', img)
cv2.waitKey(0)
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
img_blur = cv2.GaussianBlur(img_gray, (3,3), 0)
sobely = cv2.Sobel(src=img_blur, ddepth=cv2.CV_64F, dx=0, dy=1, ksize=5)
cv2.imshow('Sobel Y', sobely)
cv2.waitKey(0)
```

OUTPUT:

Q19.

AIM :

Perform Edge detection using Sobel Matrix along XY axis

SOURCE CODE :

```
import cv2
img = cv2.imread (r" E:\kishore\image(1).jpg ")
cv2.imshow('Original', img)
cv2.waitKey(0)
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
img_blur = cv2.GaussianBlur(img_gray, (3,3), 0)
sobelxy = cv2.Sobel(src=img_blur, ddepth=cv2.CV_64F, dx=1, dy=1, ksize=5)
cv2.imshow('Sobel X Y using Sobel() function', sobelxy)
cv2.waitKey(0)
```

OUTPUT:

Q20.

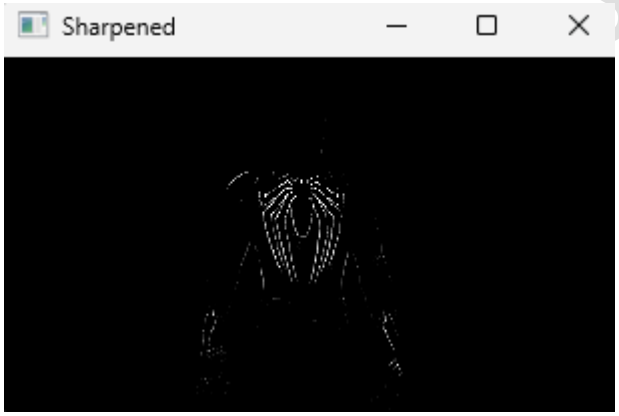
AIM :

Perform Sharpening of Image using Laplacian mask with negative center coefficient.

SOURCE CODE :

```
import cv2
import numpy as np
img = cv2.imread (r" E:\kishore\image(1).jpg ")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
kernel = np.array([[0,1,0], [1,-8,1], [0,1,0]])
sharpened = cv2.filter2D(gray, -1, kernel)
cv2.imshow('Original', gray)
cv2.imshow('Sharpened', sharpened)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:



Q21.

AIM :

Perform Sharpening of Image using Laplacian mask implemented with an extension of diagonal neighbors

SOURCE CODE :

```
import cv2
import numpy as np
img = cv2.imread (r" E:\kishore\image(1).jpg ")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
kernel = np.array([[0,1,0], [1,-4,1], [0,1,0]])
sharpened = cv2.filter2D(gray, -1, kernel)
cv2.imshow('Original', gray)
cv2.imshow('Sharpened', sharpened)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:

Q22.

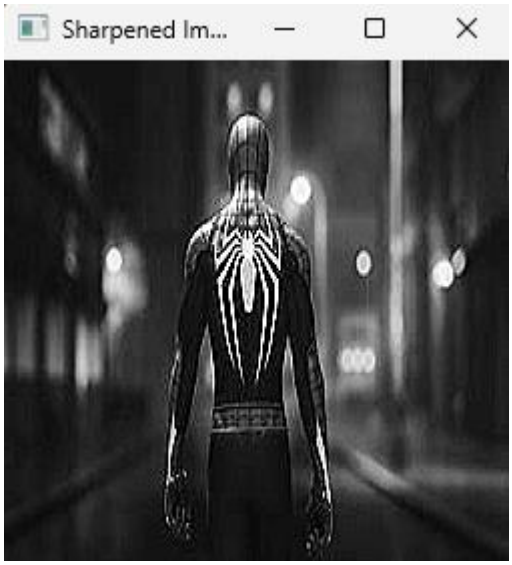
AIM :

Perform Sharpening of Image using Laplacian mask with positive center coefficient.

SOURCE CODE :

```
import cv2
import numpy as np
img = cv2.imread( r" E:\kishore\image(1).jpg ")
img = cv2.resize(img,(255, 255))
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
laplacian_kernel = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]])
sharpened_img = cv2.filter2D(gray_img, -1, laplacian_kernel)
sharpened_img = cv2.cvtColor(sharpened_img, cv2.COLOR_GRAY2BGR)
cv2.imshow('Original Image', img)
cv2.imshow('Sharpened Image', sharpened_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:



Q23.

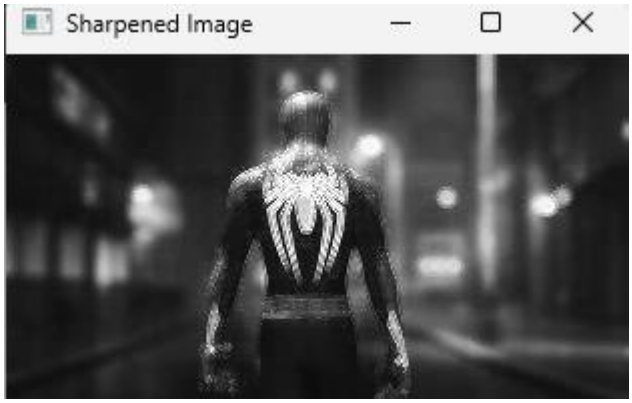
AIM :

Perform Sharpening of Image using unsharp masking.

SOURCE CODE :

```
import cv2
import numpy as np
img = cv2.imread (r" E:\kishore\image(1).jpg ")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
laplacian_kernel = np.array([[0, 1, 0],[1, -4, 1],[0, 1, 0]])
laplacian = cv2.filter2D(gray, -1, laplacian_kernel)
sharpened = cv2.add(gray, laplacian)
cv2.imshow('Original Image', gray)
cv2.imshow('Sharpened Image', sharpened)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:



Q24.

AIM :

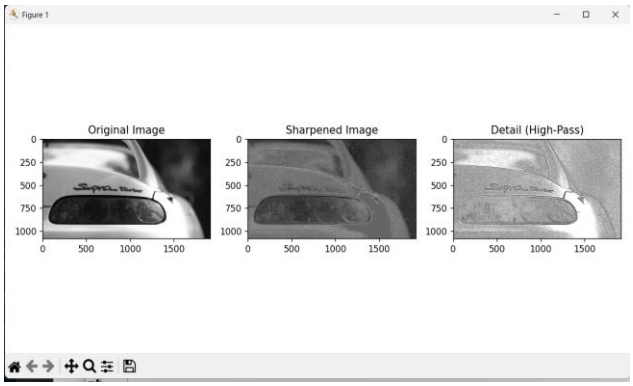
Perform Sharpening of Image using High-Boost Masks.

SOURCE CODE :

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
def high_boost_sharpening(image_path, k=1.5):
    original_image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
```

```
    blurred_image = cv2.GaussianBlur(original_image, (5, 5), 0)
    high_pass = original_image - blurred_image
    sharpened_image = original_image + k * high_pass
    return original_image, sharpened_image
image_path =r"E:\kishore\image(1).jpg”
original_image, sharpened_image = high_boost_sharpening(image_path, k=1.5)
plt.figure(figsize=(10, 5))
plt.subplot(1, 3, 1)
plt.imshow(original_image, cmap='gray')
plt.title('Original Image')
plt.subplot(1, 3, 2)
plt.imshow(sharpened_image, cmap='gray')
plt.title('Sharpened Image')
plt.subplot(1, 3, 3)
plt.imshow(original_image - sharpened_image, cmap='gray')
plt.title('Detail (High-Pass)')
plt.tight_layout()
plt.show()
```

OUTPUT:



Q25.

AIM :

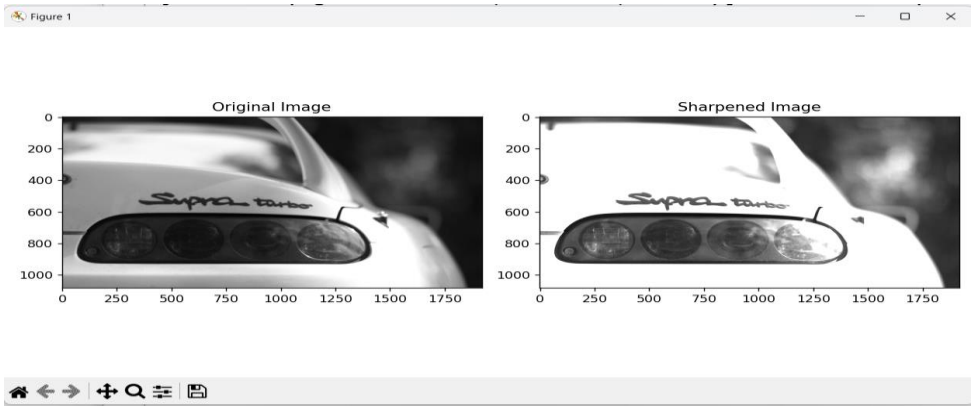
Perform Sharpening of Image using Gradient masking

SOURCE CODE :

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
def sharpen_image_with_gradient(image_path, alpha=1.5):
```

```
    original_image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    gradient_x = cv2.Sobel(original_image, cv2.CV_64F, 1, 0, ksize=3)
    gradient_y = cv2.Sobel(original_image, cv2.CV_64F, 0, 1, ksize=3)
    gradient_magnitude = np.sqrt(gradient_x**2 + gradient_y**2)
    gradient_magnitude = cv2.normalize(gradient_magnitude, None, 0, 255,
cv2.NORM_MINMAX, cv2.CV_8U)
    sharpened_image = cv2.addWeighted(original_image, 1 + alpha, gradient_magnitude, -
alpha, 0)
    return original_image, sharpened_image
image_path = r" E:\kishore\image(1).jpg "
original_image, sharpened_image = sharpen_image_with_gradient(image_path, alpha=1.5)
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(original_image, cmap='gray')
plt.title('Original Image')
plt.subplot(1, 2, 2)
plt.imshow(sharpened_image, cmap='gray')
plt.title('Sharpened Image')
plt.tight_layout()
plt.show()
```


OUTPUT:



Q26.

AIM :

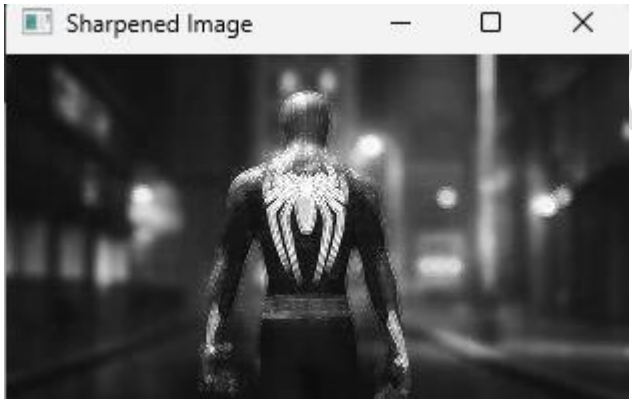
Insert water marking to the image using OpenCV.

SOURCE CODE :

```
import cv2
img = cv2.imread (r" E:\kishore\image(1).jpg ")
wm = cv2.imread (r" E:\kishore\image(1).jpg ")
h_wm, w_wm = wm.shape[:2]
```

```
h_img, w_img = img.shape[:2]
center_x = int(w_img/2)
center_y = int(h_img/2)
top_y = center_y - int(h_wm/2)
left_x = center_x - int(w_wm/2)
bottom_y = top_y + h_wm
right_x = left_x + w_wm
roi = img[top_y:bottom_y, left_x:right_x]
result = cv2.addWeighted(roi, 1, wm, 0.3, 0)
img[top_y:bottom_y, left_x:right_x] = result
cv2.imshow("Watermarked Image", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:



Q27.

AIM :

Do Cropping, Copying and pasting image inside another image using OpenCV

SOURCE CODE :

```
import cv2
import numpy as np
image = cv2.imread (r" E:\kishore\image(1).jpg ")
img2 = cv2.imread (r" E:\kishore\image(1).jpg ")
print(image.shape) # Print image shape
cv2.imshow("original", image)
imageCopy = image.copy()
cv2.circle(imageCopy, (100, 100), 30, (255, 0, 0), -1)
cv2.imshow('image', image)
```

```
cv2.imshow('image copy', imageCopy)
cropped_image = image[80:280, 150:330]
cv2.imshow("cropped", cropped_image)
cv2.imwrite("Cropped Image.jpg", cropped_image)
dst = cv2.addWeighted(image, 0.5, img2, 0.7, 0)
img_arr = np.hstack((image, img2))
cv2.imshow('Input Images',img_arr)
cv2.imshow('Blended Image',dst)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:

Q28.

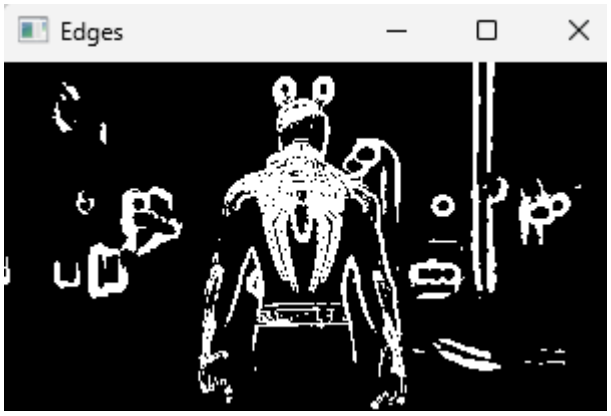
AIM :

Find the boundary of the image using Convolution kernel for the given image

SOURCE CODE :

```
import cv2
import numpy as np
img = cv2.imread(r" E:\kishore\image(1).jpg ",cv2.IMREAD_GRAYSCALE)
dx = cv2.Sobel(img, cv2.CV_64F, 1, 0)
dy = cv2.Sobel(img, cv2.CV_64F, 0, 1)
edges = cv2.magnitude(dx, dy)
thresh = 100
edges[edges < thresh] = 0
edges[edges >= thresh] = 255
cv2.imshow("Edges", edges)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:



Q29.

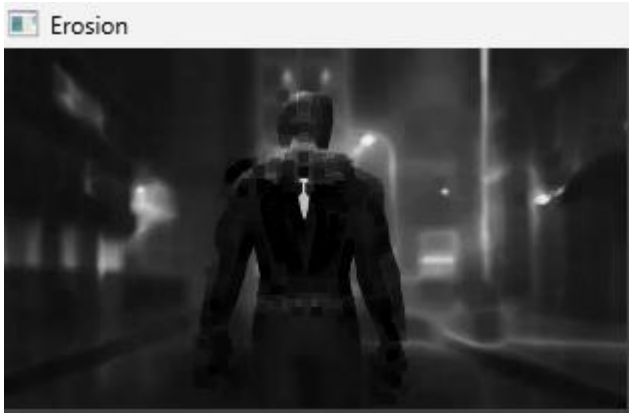
AIM :

Morphological operations based on OpenCV using Erosion technique

SOURCE CODE :

```
import cv2
import numpy as np
img = cv2.imread(r" E:\kishore\image(1).jpg ",cv2.IMREAD_GRAYSCALE)
kernel = np.ones((5,5), np.uint8)
erosion = cv2.erode(img, kernel, iterations=1)
cv2.imshow("Original", img)
cv2.imshow("Erosion", erosion)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:



Q30.

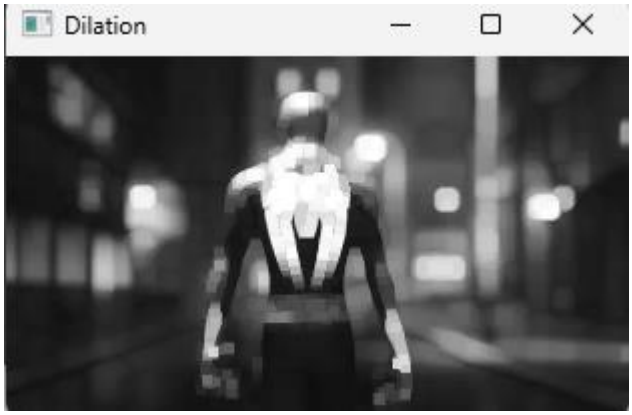
AIM :

Morphological operations based on OpenCV using Dilation technique

SOURCE CODE :

```
import cv2
import numpy as np
img = cv2.imread(r" E:\kishore\image(1).jpg ",cv2.IMREAD_GRAYSCALE)
kernel = np.ones((5,5), np.uint8)
dilation = cv2.dilate(img, kernel, iterations=1)
cv2.imshow("Original", img)
cv2.imshow("Dilation", dilation)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:



Q31.

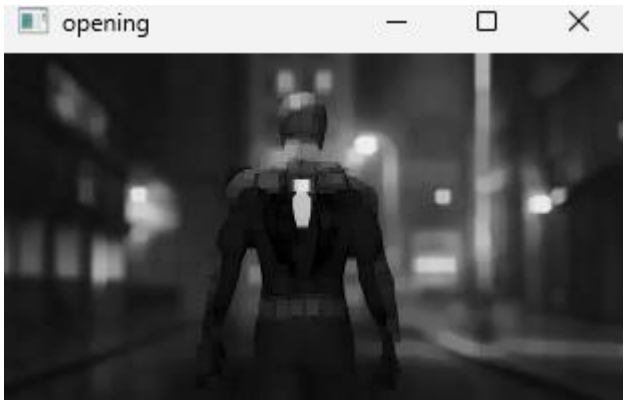
AIM :

Morphological operations based on OpenCV using Opening technique.

SOURCE CODE :

```
import cv2
import numpy as np
img = cv2.imread(r" E:\kishore\image(1).jpg ",cv2.IMREAD_GRAYSCALE)
kernel = np.ones((5,5), np.uint8)
opening = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)
cv2.imshow("Original", img)
cv2.imshow("opening", opening)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:



Q32.

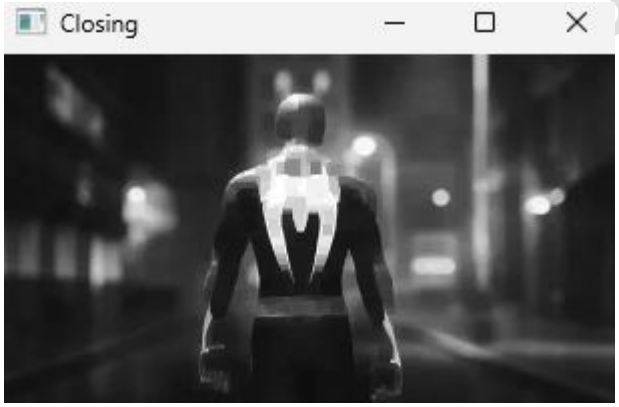
AIM :

Morphological operations based on OpenCV using Closing technique.

SOURCE CODE :

```
import cv2
import numpy as np
img = cv2.imread(cv2.IMREAD_GRAYSCALE)
kernel = np.ones((5,5), np.uint8)
closing = cv2.morphologyEx(r" E:\kishore\image(1).jpg ",img, cv2.MORPH_CLOSE, kernel)
cv2.imshow("Original", img)
cv2.imshow("Closing", closing)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:



Q33.

AIM :

Morphological operations based on OpenCV using Morphological Gradient technique

SOURCE CODE :

```
import cv2
import numpy as np
img = cv2.imread(r"E:\kishore\image(1).jpg", cv2.IMREAD_GRAYSCALE)
kernel = np.ones((5,5), np.uint8)
grad = cv2.morphologyEx(img, cv2.MORPH_GRADIENT, kernel)
cv2.imshow("Original", img)
cv2.imshow("Gradient", grad)
cv2.waitKey
```

OUTPUT:

Q34.

AIM :

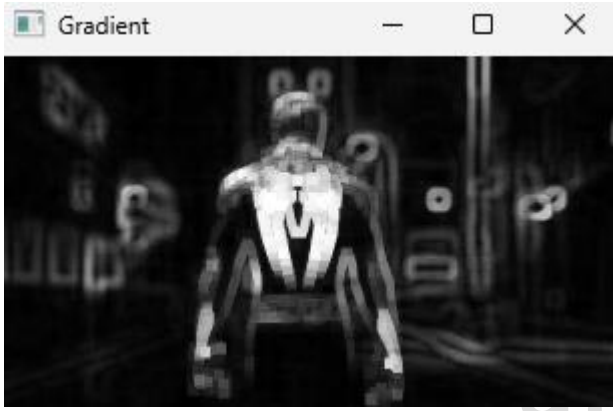
Morphological operations based on OpenCV using Top hat technique.

SOURCE CODE :

```
import cv2
import numpy as np
img = cv2.imread(r" E:\kishore\image(1).jpg ",cv2.IMREAD_GRAYSCALE)
kernel = np.ones((5,5), np.uint8)
tophat = cv2.morphologyEx(img, cv2.MORPH_TOPHAT, kernel)
```

```
cv2.imshow("Original", img)
cv2.imshow("Top Hat", tophat)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:



Q35.

AIM :

Morphological operations based on OpenCV using Black hat technique.

SOURCE CODE :

```
import cv2
import numpy as np
img = cv2.imread(r" E:\kishore\image(1).jpg ",cv2.IMREAD_GRAYSCALE)
kernel = np.ones((5,5), np.uint8)
blackhat = cv2.morphologyEx(img, cv2.MORPH_BLACKHAT, kernel)
cv2.imshow("Original", img)
cv2.imshow("Black Hat", blackhat)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:

Q36.

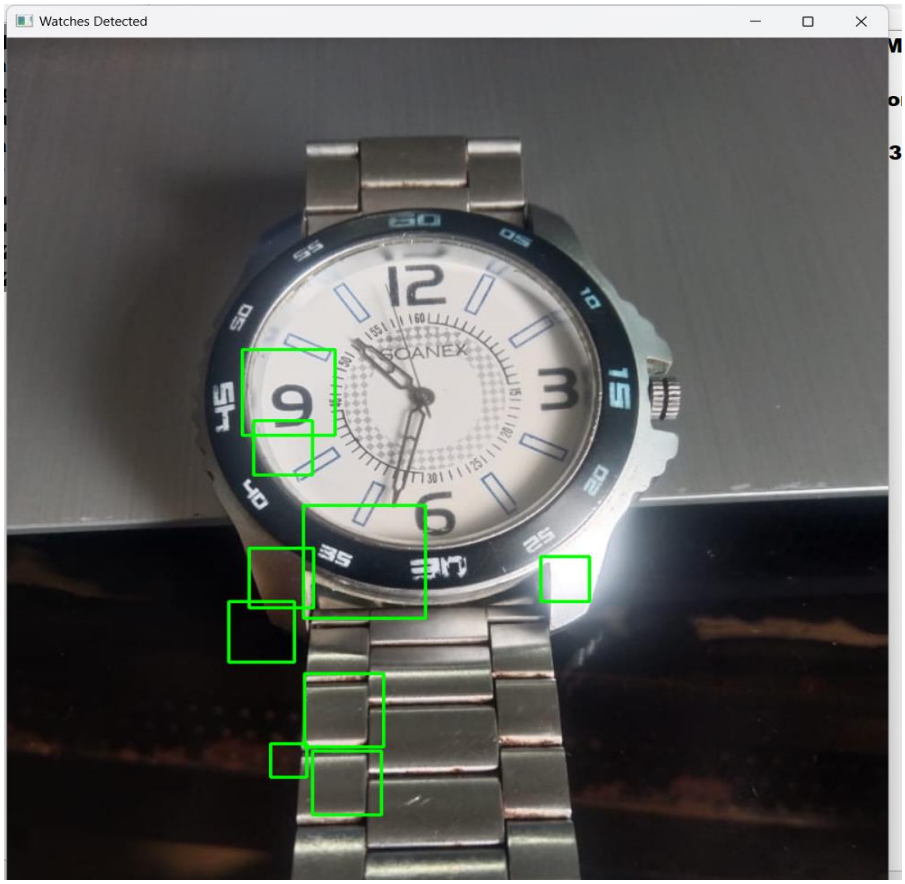
AIM :

Recognise watch from the given image by general Object recognition using OpenCV.

SOURCE CODE :

```
import cv2
watch_cascade = cv2.CascadeClassifier(“E:\kishore\image(1).jpg")//Object-Detection-using-OpenCV-master//watch-cascade.xml")
img = cv2.imread ("E:\kishore\image(1).jpg ")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
watches = watch_cascade.detectMultiScale(gray, scaleFactor=1.2, minNeighbors=5)
for (x, y, w, h) in watches:
   cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)
   cv2.imshow('Watches Detected', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:



Q37.

AIM :

Using Opencv play Video in Reverse mode.

SOURCE CODE :

```
import cv2
cap = cv2.VideoCapture(r" E:\kishore\countdown_-_2637 (360p) (1).mp4")
total_frames = cap.get(cv2.CAP_PROP_FRAME_COUNT)
current_frame = total_frames - 1
while current_frame >= 0:
    cap.set(cv2.CAP_PROP_POS_FRAMES, current_frame)
    ret, frame = cap.read()
    if not ret:
        break
    cv2.imshow('Video in Reverse', frame)
    if cv2.waitKey(25) & 0xFF == ord('q'):
        break
    current_frame -= 1
cap.release()
cv2.destroyAllWindows()
```

OUTPUT:

Q38.

AIM :

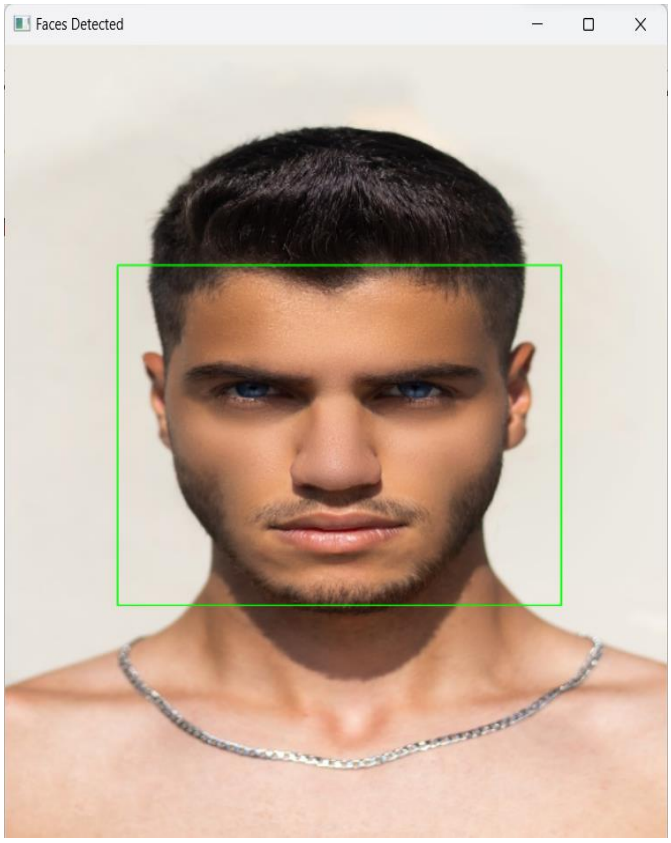
Face Detection using Opencv

SOURCE CODE :

```
import cv2
img = cv2.imread import cv2
img = cv2.imread (E:\kishore\image(1).jpg ")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
face_cascade =cv2.CascadeClassifier ("E:\kishore\image(1).jpg”)
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5)
for (x, y, w, h) in faces:
   cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)
   img1 = cv2.resize(img,(600,600))
   cv2.imshow('Faces Detected', img1)
cv2.waitKey(0)
cv2.destroyAllWindows()
)
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
face_cascade =cv2.CascadeClassifier("E:\kishore\haarcascade_frontalface_default.xml")
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5)
for (x, y, w, h) in faces:
   cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)
```

```
    img1 = cv2.resize(img,(600,600))
    cv2.imshow('Faces Detected', img1)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:



Q39.

AIM :

Vehicle Detection in a Video frame using OpenCV

SOURCE CODE :

```
import cv2
car_cascade = cv2.CascadeClassifier ("E:\kishore\video(1).mp4")
cap = cv2.VideoCapture ("E:\kishore\video(1).mp4")
while True:
    ret, frame = cap.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    cars = car_cascade.detectMultiScale(gray, 1.1, 1)
```

```
    for (x,y,w,h) in cars:
        cv2.rectangle(frame, (x,y), (x+w,y+h), (0,0,255), 2)
        cv2.imshow('frame', frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
cap.release()
cv2.destroyAllWindows()
```

OUTPUT:

Q10.

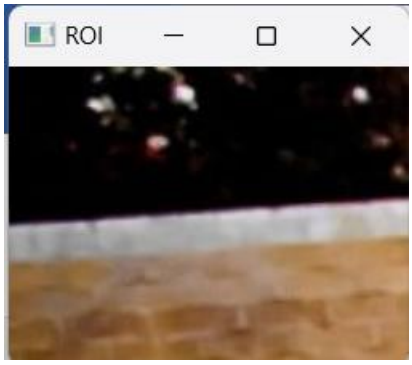
AIM :

Draw Rectangular shape and extract objects

SOURCE CODE :

```
import cv2
img = cv2.imread ("E:\kishore\image(1).jpg ")
x, y = 100, 100
width, height = 200, 150
roi = img[y:y+height, x:x+width]
cv2.imshow('ROI', roi)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:



##########################