ACDS Lecture Series

Lecture - 17

CSIR

OpenCV

G. N. Sastry and Team

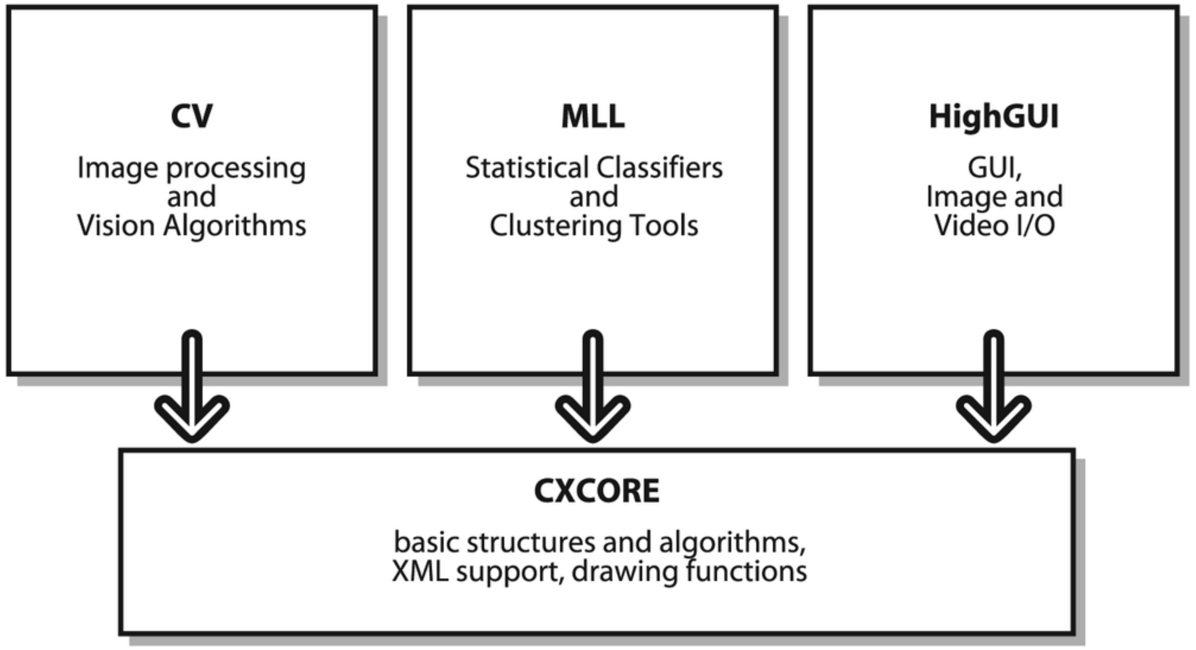
ADVANCED COMPUTATION AND DATA SCIENCES (ACDS) DIVISION

CSIR-North East Institute of Science and Technology, Jorhat, Assam, India

Contents

17.1 Introduction

- OpenCV is an Image Processing library created by Intel and maintained by Willow Garage.
- Available for C, C++, and Python
- Newest update is version 2.2
- Open Source and free
- Easy to use and install


CV
Image processing and Vision Algorithms
MLL
Statistical Classifiers and Clustering Tools
HighGUI
GUI, Image and Video I/O
CXCORE
basic structures and algorithms, XML support, drawing functions

OpenCV Modules

17.1.1 Installation

- For Mac OS X. Simply install Mac Ports then type

  sudo port install opencv

- Do not use synaptic on Linux to install OpenCV. It is version 1.2.

- For Linux and Windows, follow the installation guide at http://opencv.willowgarage.com/wiki/InstallGuide

- Linux users can come to me for help. I have built it on Ubuntu dozens of times. I have built it successfully on Windows once.

- Make sure to read the beginning as it gives you precise commands to install ffmpeg, libavformat-dev, libswscale-dev, and other required libraries.

17.2 Core Functionality

- Mat - The Basic Image Container
- How to scan images, lookup tables and time measurement with OpenCV
- Mask operations on matrices
- Operations with images
- Adding (blending) two images using OpenCV
- Changing the contrast and brightness of an image!
- Discrete Fourier Transform
- File Input and Output using XML and YAML files
- How to use the OpenCV parallel_for_ to parallelize your code

17.2 Core Functionality

- OpenCV uses the **cv** namespace.
- cv::Mat object replaces the original C standard **IplImage** and **CvMat** classes.
- All original functions and classes of the C standard OpenCV components in the Bradski book are still available and current. However you will need to read that book for it.
- **namedWindow** is used for viewing images. See my manual for instructions on calling it.
  - In general, default string as input with original image size set. Else, use string as input name and 0 for adjustable size.

17.3 Preprocessing

Preprocessing or namely image processing is a prior step in computer vision, where the goal is to convert an image into a form suitable for further analysis.

Examples of operations such as exposure correction, color balancing, image noise reduction, or increasing image sharpness are highly important and very care demanding to achieve acceptable results in most computer vision applications like computational photography or even face recognition.

- Steps in Preprocessing:
  - Images in Computer Vision
  - Color: RGB Representation
  - Pixel Transforms
  - Histogram Equalization

17.3 Preprocessing

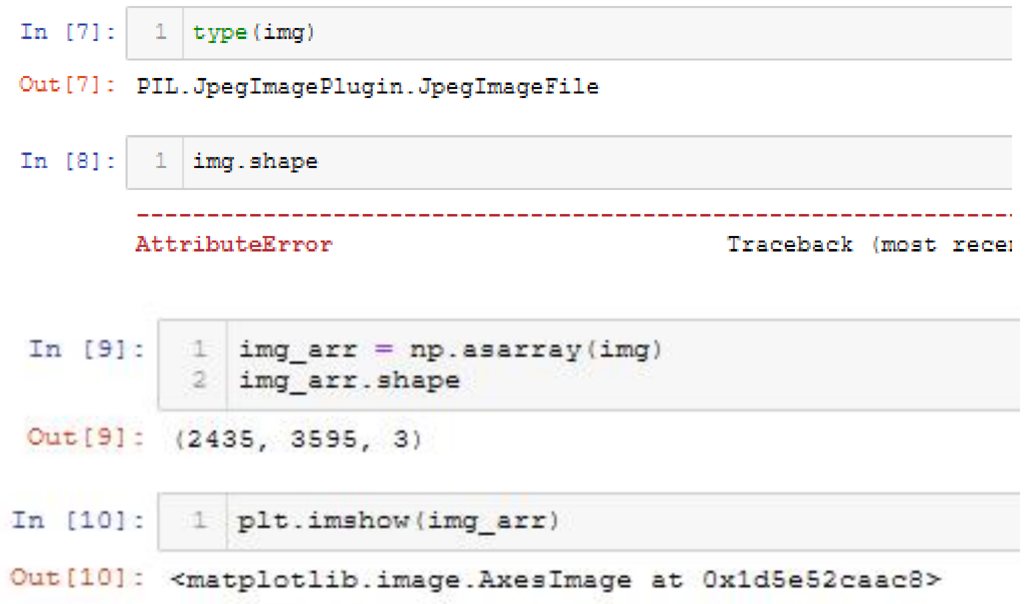
Getting Started with Images

```
In [5]:  import numpy as np
         import matplotlib.pyplot as plt
         %matplotlib inline
```

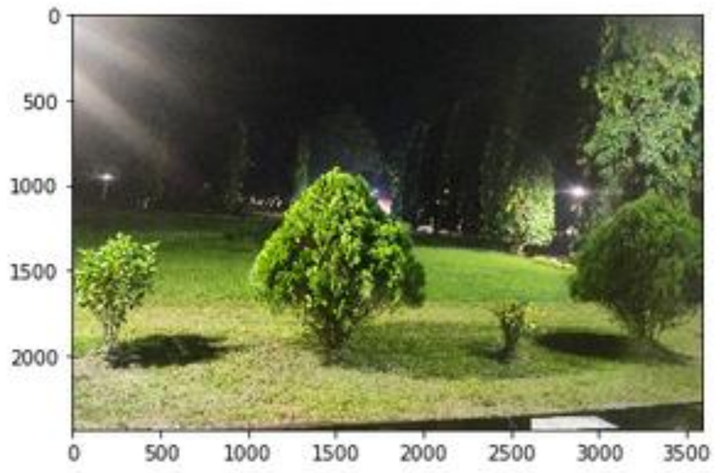
```
In [6]:  from PIL import Image
         img = Image.open('D:/Ranjan_Dataset/Images/green.jpg')
         img
```

Out[6]:

17.3 Preprocessing

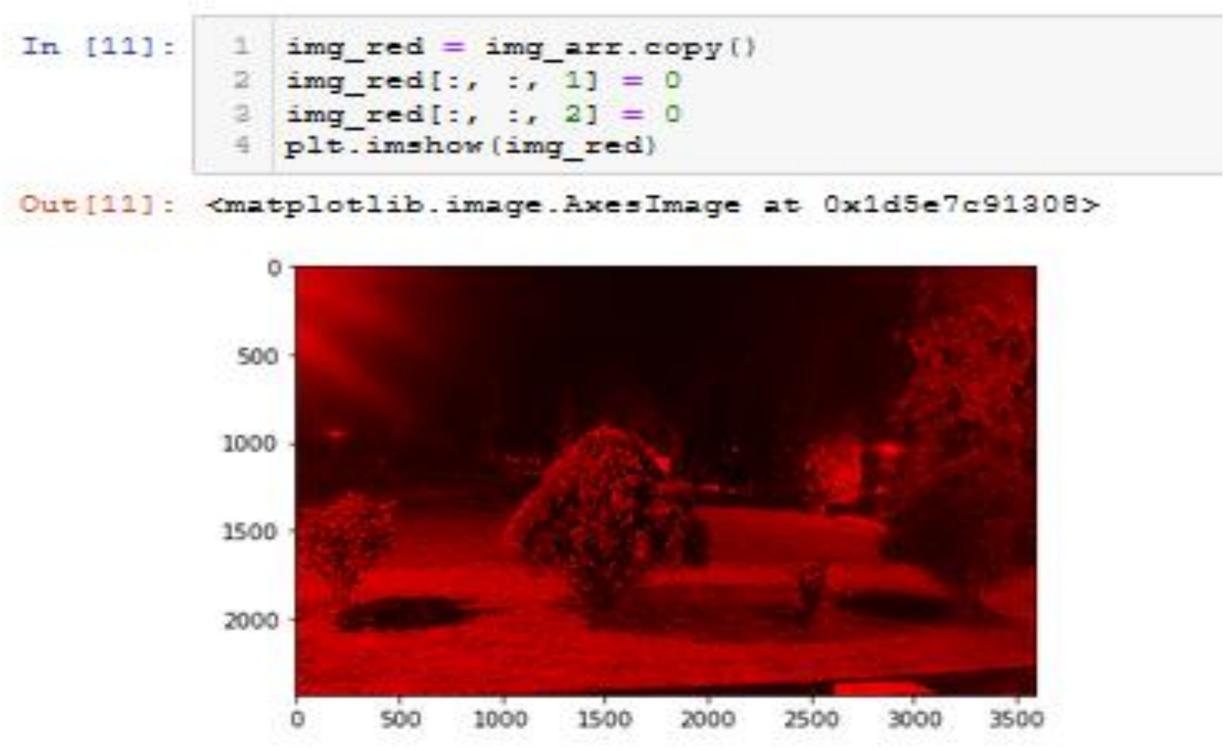


```
In [7]:  type(img)

Out[7]:  PIL.JpegImagePlugin.JpegImageFile

In [8]:  img.shape

         ---------------------------------------------------------------------------
         AttributeError                            Traceback (most rece

In [9]:  img_arr = np.asarray(img)
         img_arr.shape

Out[9]:  (2435, 3595, 3)

In [10]: plt.imshow(img_arr)

Out[10]: <matplotlib.image.AxesImage at 0x1d5e52caac8>
```



17.3 Preprocessing

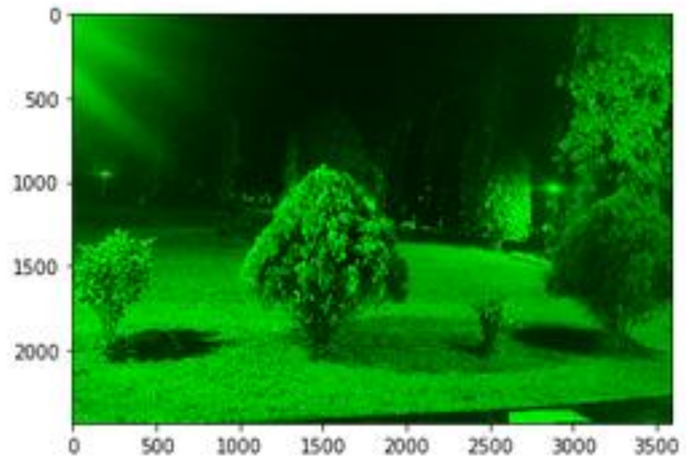
- Color space is represented by three different channels Red, Green, and Blue.
- Each channel stems from the so-called *trichromatic nature of human vision* since we have three separate photoreceptors each of which respond selectively to different portions of the color spectrum.
- The three primary colors are added to produce 16.777.216 distinct colors in an 8-bit per channel RGB system.



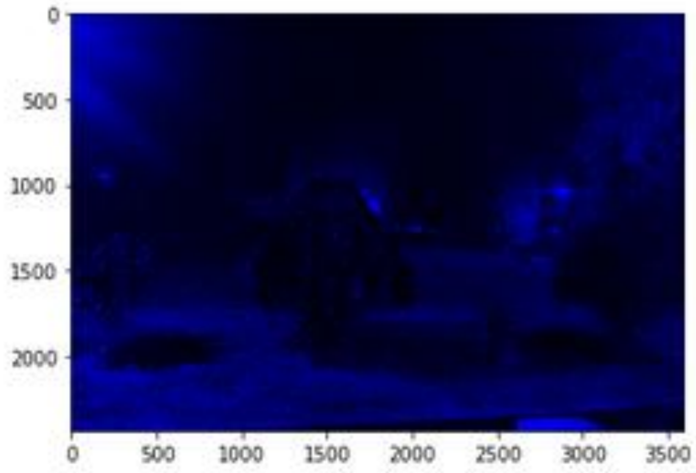
```
In [11]:  img_red = img_arr.copy()
          img_red[:, :, 1] = 0
          img_red[:, :, 2] = 0
          plt.imshow(img_red)

Out[11]: <matplotlib.image.AxesImage at 0x1d5e7c91308>
```

17.3 Preprocessing

```
In [12]:
img_green = img_arr.copy()
img_green[:, :, 0] = 0
img_green[:, :, 2] = 0
plt.imshow(img_green)

Out[12]: <matplotlib.image.AxesImage at 0x1d5e7cfe3c8>
```



```
In [13]:
img_blue = img_arr.copy()
img_blue[:, :, 0] = 0
img_blue[:, :, 1] = 0
plt.imshow(img_blue)

Out[13]: <matplotlib.image.AxesImage at 0x1d5f37bf908>
```

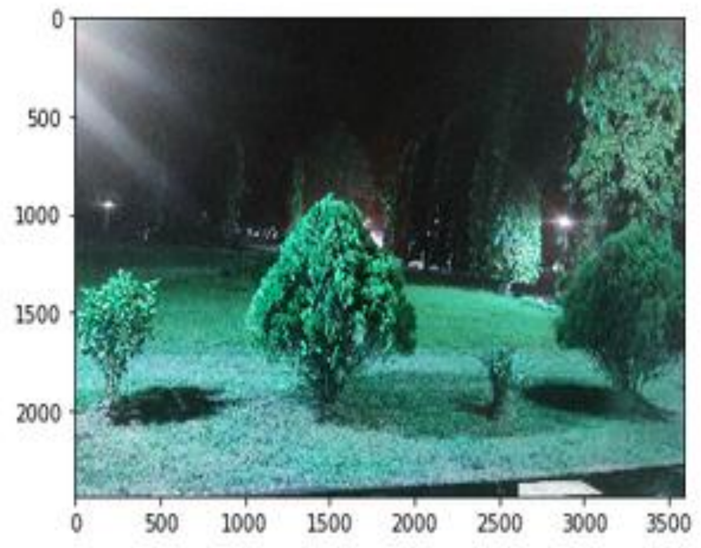


17.3 Preprocessing

- In OpenCV, images are converted into multi-dimensional arrays, which greatly simplifies their manipulation.

```
In [14]: img = cv2.imread("D:/Ranjan_Dataset/Images/green.jpg")
         img

Out[14]: array([[[210, 214, 219],
                 [213, 217, 222],
                 [212, 216, 221],
                 ...,
                 [117, 127, 121],
                 [116, 126, 120],
                 [104, 117, 109]],

                [[207, 211, 216],
                 [209, 213, 218],
                 [209, 213, 218],
                 ...,
                 [113, 123, 117],
                 [116, 126, 120],
                 [104, 117, 109]],

                [[209, 213, 218],
                 [211, 215, 220],
                 [210, 214, 219],
                 ...,
                 [111, 123, 117],
                 [116, 128, 122],
                 [113, 125, 119]],

                ...,

                [[143, 191, 179],
                 [151, 200, 186],
                 [149, 197, 185],
                 ...,
                 [ 11,  46,  42],
                 [  0,  27,  23],
                 [  0,  28,  24]],

                [[131, 180, 166],
                 [142, 191, 177],
                 [150, 199, 185],
                 ...,
                 [ 33,  68,  64],
                 [ 24,  59,  55],
                 [ 29,  62,  58]],

                [[125, 174, 160],
                 [134, 183, 169],
                 [151, 200, 186],
                 ...,
                 [ 39,  74,  70],
                 [ 40,  75,  71],
                 [ 46,  79,  75]]], dtype=uint8)
```

```
In [15]: plt.imshow(img)

Out[15]: <matplotlib.image.AxesImage at 0x1d5f6a695c8>
```

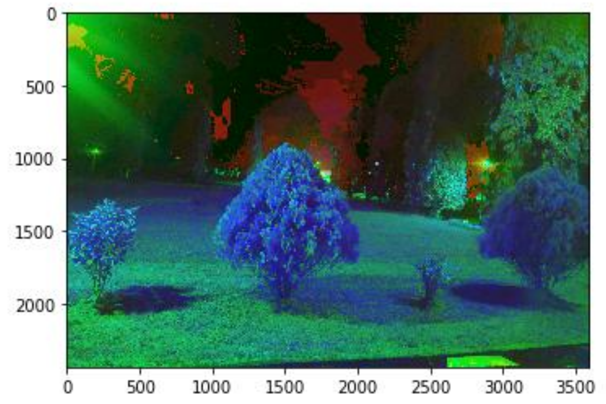


17.3 Preprocessing

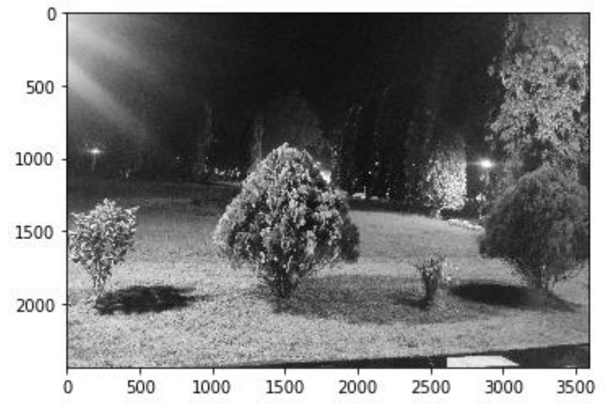
- Convert BGR and RGB and HSV and HLS

  The order of colors is BGR (blue, green, red) , RGB(red, green, blue), HSV(Hue Saturation Value).

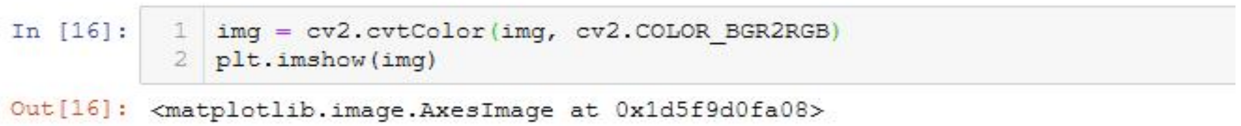
```
In [5]:  imgs = cv2.cvtColor(img, cv2.COLOR_BGR2HLS)
         plt.imshow(imgs)

Out[5]: <matplotlib.image.AxesImage at 0x1f910be2648>
```



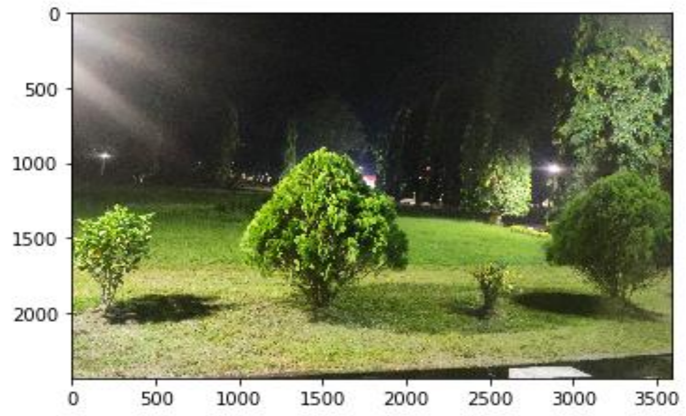
```
In [17]:  img = cv2.imread('D:/Ranjan_Dataset/Images/green.jpg', 0)
          plt.imshow(img, cmap='gray')

Out[17]: <matplotlib.image.AxesImage at 0x1d580024d08>
```

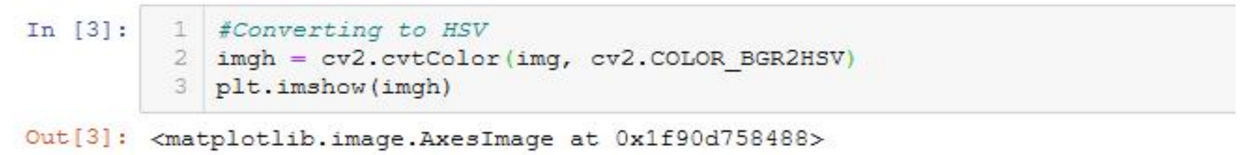


17.3 Preprocessing

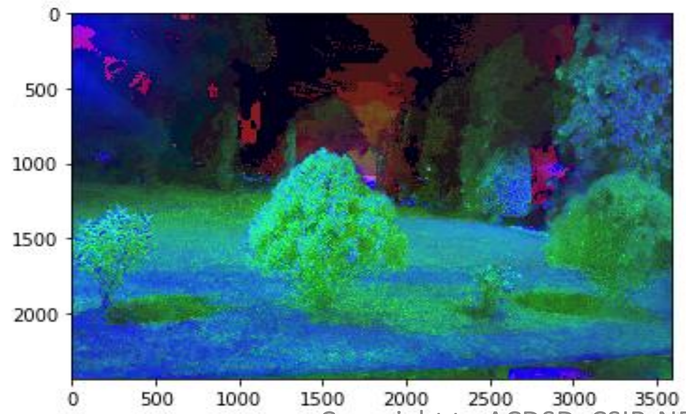


```
In [16]:  img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
          plt.imshow(img)

Out[16]: <matplotlib.image.AxesImage at 0x1d5f9d0fa08>
```



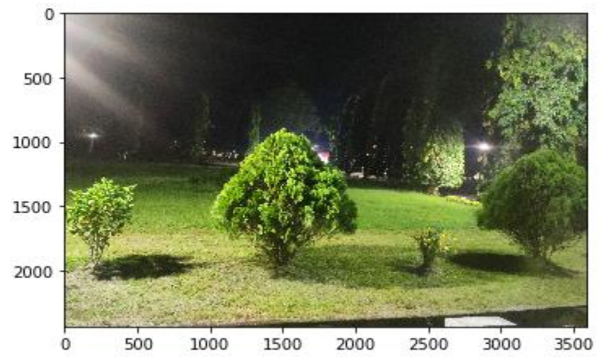


```
In [3]:  #Converting to HSV
         imgh = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
         plt.imshow(imgh)

Out[3]: <matplotlib.image.AxesImage at 0x1f90d758488>
```

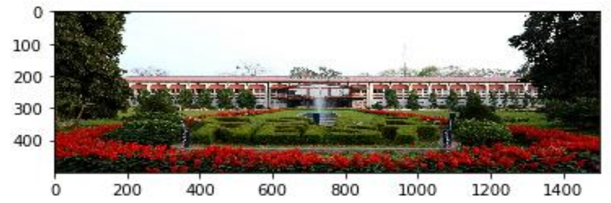


17.3 Preprocessing

```
In [29]: #Import and the two images
         img1 = cv2.imread('D:/Ranjan_Dataset/Images/green.jpg')
         img2 = cv2.imread('D:/Ranjan_Dataset/Images/admin.jpg')

         img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
         img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2RGB)

In [7]:  img1.shape

Out[7]:  (2435, 3595, 3)

In [8]:  img2.shape

Out[8]:  (500, 1500, 3)

In [9]:  plt.imshow(img1)

Out[9]:  <matplotlib.image.AxesImage at 0x1f9127af508>
```



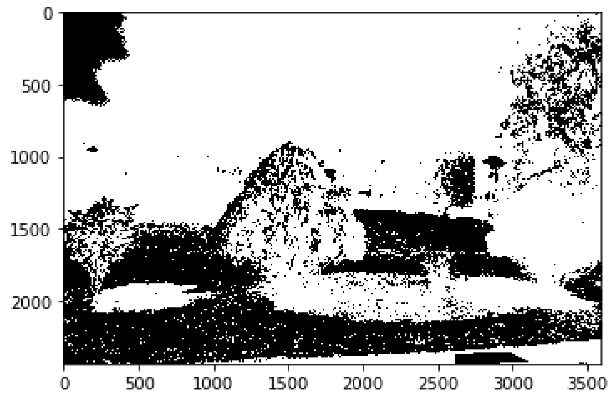
```
In [10]: plt.imshow(img2)

Out[10]: <matplotlib.image.AxesImage at 0x1f914136288>
```



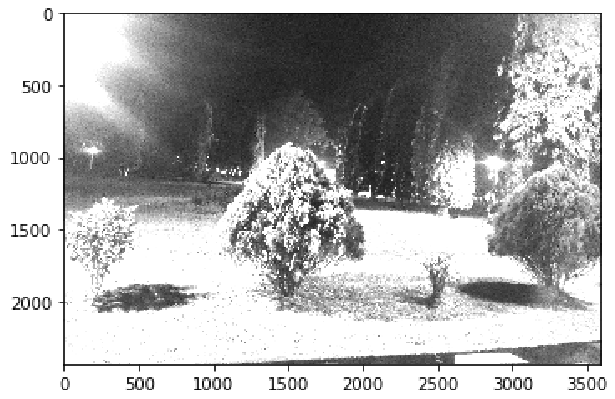
17.3 Preprocessing

- For instance, a grayscale image is interpreted as a 2D array with pixels varying from 0 to 255.

```
In [46]:  #Binary Inverse
          ret,thresh2 = cv2.threshold(img,127,255,cv2.THRESH_BINARY_INV)
          plt.imshow(thresh2,cmap='gray')

Out[46]: <matplotlib.image.AxesImage at 0x1f90d0a0f88>
```

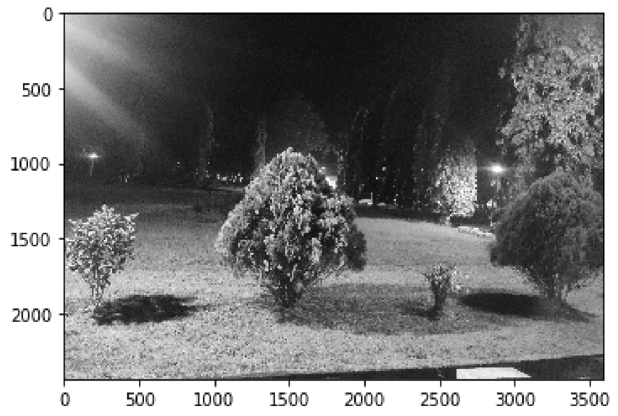


```
In [47]:  #Threshold Truncation
          ret,thresh3 = cv2.threshold(img,127,255,cv2.THRESH_TRUNC)
          plt.imshow(thresh3,cmap='gray')

Out[47]: <matplotlib.image.AxesImage at 0x1f90d109608>
```

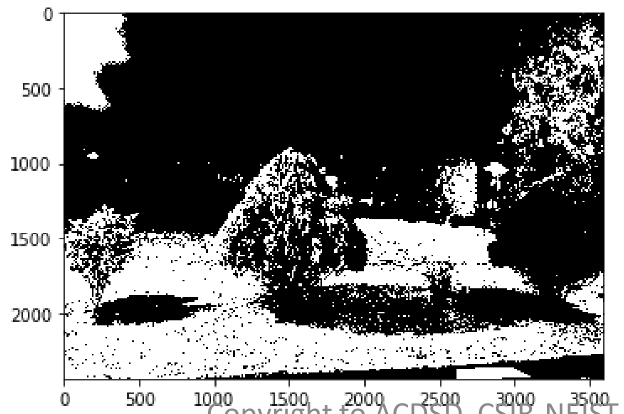


17.3 Preprocessing

```
In [41]:  img = cv2.imread('D:/Ranjan_Dataset/Images/green.jpg',0)
          plt.imshow(img,cmap='gray')

Out[41]: <matplotlib.image.AxesImage at 0x1f90bd6ad88>
```



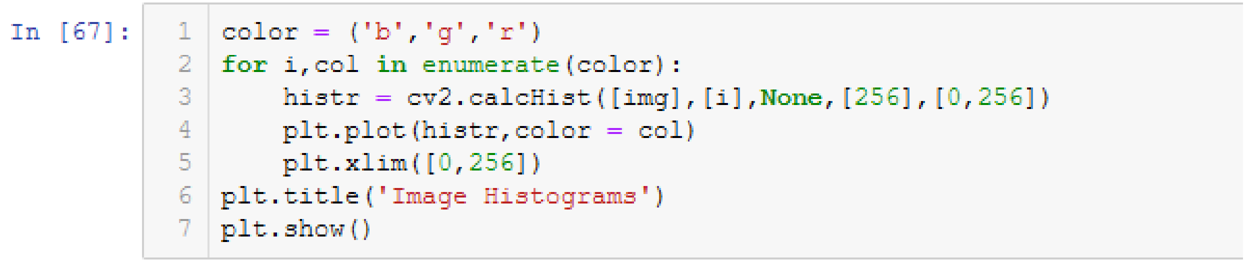
```
In [45]:  #Simple Binary
          ret,thresh1 = cv2.threshold(img,127,127,cv2.THRESH_BINARY)
          ret
          plt.imshow(thresh1,cmap='gray')

Out[45]: <matplotlib.image.AxesImage at 0x1f90d03bf88>
```



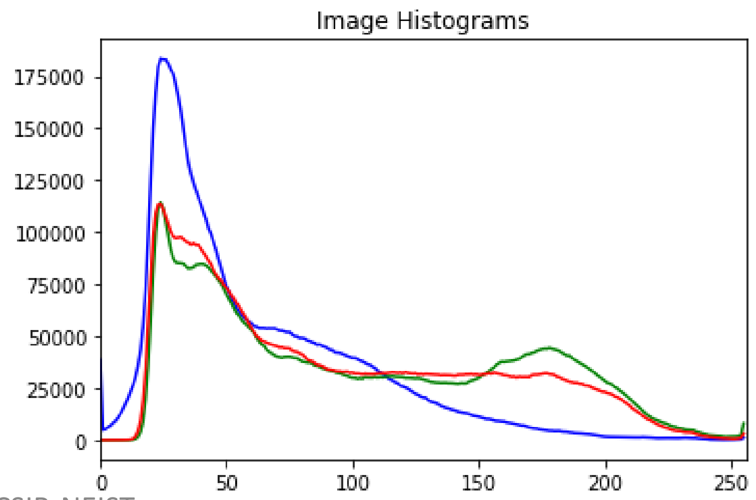
17.3 Preprocessing

Histogram Equalization:

- Plotting the pixel intensity distribution we can visualize the set of lightness values for an image.
- We can identify and correct darkest areas in the image to make it feel more appealing
- The real objective is to find an automated process to equally distribute light across the entire image without having to check area by area, pixel by pixel
- We apply a transformation function to spread out the most frequent intensity values uniformly across the image

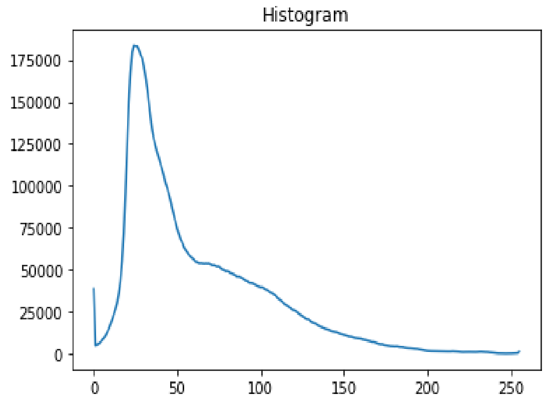


In [67]:
```
color = ('b','g','r')
for i,col in enumerate(color):
    histr = cv2.calcHist([img],[i],None,[256],[0,256])
    plt.plot(histr,color = col)
    plt.xlim([0,256])
plt.title('Image Histograms')
plt.show()
```

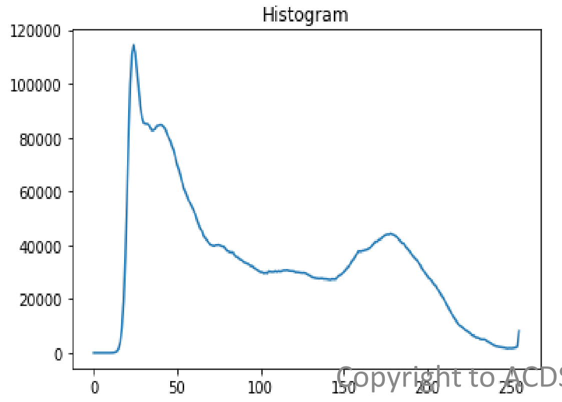


17.3 Preprocessing

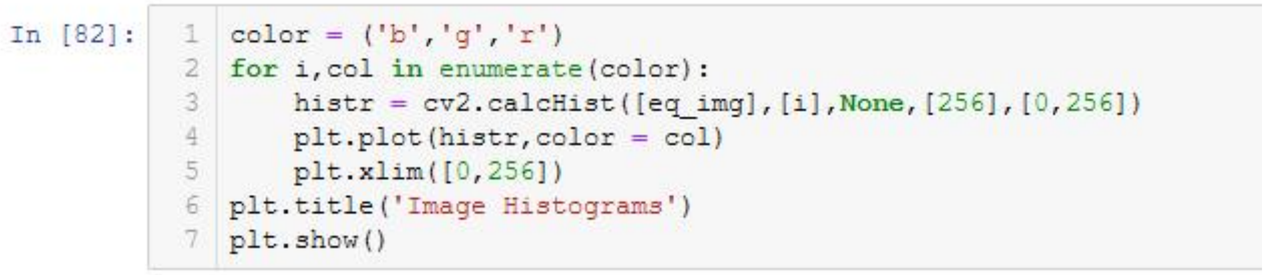
```
In [73]: hist_values = cv2.calcHist([img], channels=[0], mask=None, histSize=[256], ranges=[0,256])
         hist_values.shape
         plt.plot(hist_values)
         plt.title('Histogram')

Out[73]: Text(0.5, 1.0, 'Histogram')
```



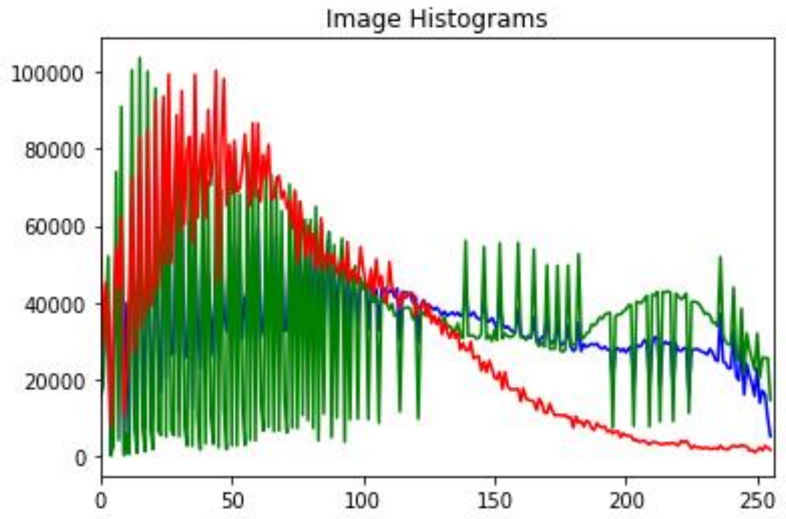
```
In [72]: hist_values = cv2.calcHist([img], channels=[1], mask=None, histSize=[256], ranges=[0,256])
         hist_values.shape
         plt.plot(hist_values)
         plt.title('Histogram')

Out[72]: Text(0.5, 1.0, 'Histogram')
```



17.3 Preprocessing



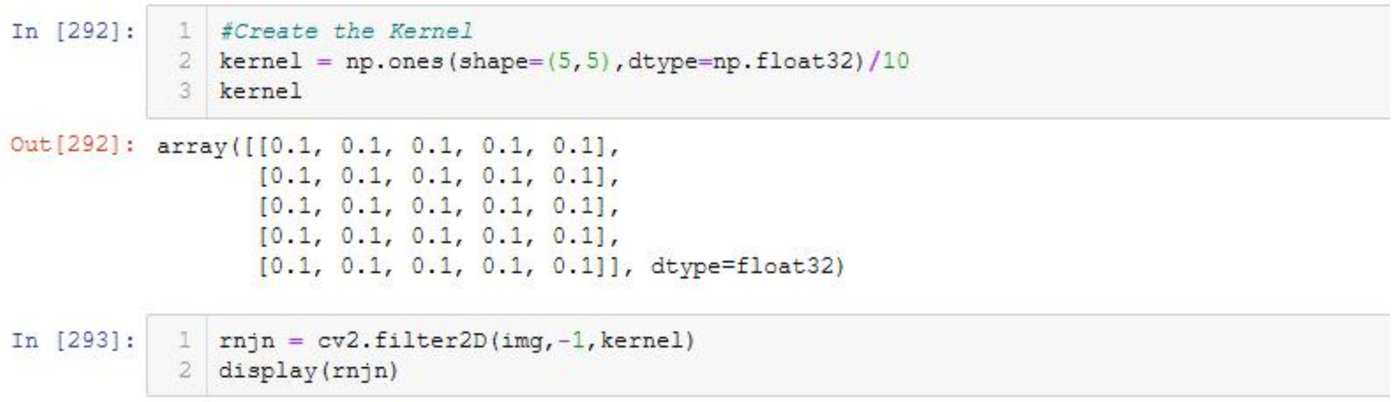
```
In [82]:
color = ('b','g','r')
for i,col in enumerate(color):
    histr = cv2.calcHist([eq_img],[i],None,[256],[0,256])
    plt.plot(histr,color = col)
    plt.xlim([0,256])
plt.title('Image Histograms')
plt.show()
```



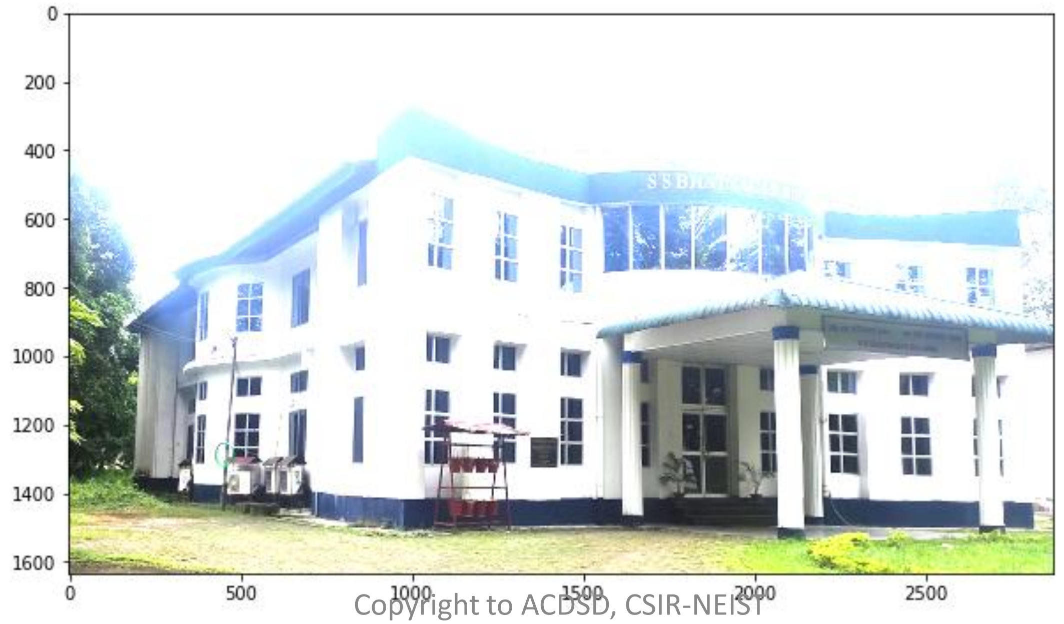
17.3 Preprocessing

- **Image Filtering**

  Applies 2D Convolutions employing various low and high pass filters that help in removing noise, blurring images, etc.



```
In [292]:  #Create the Kernel
            kernel = np.ones(shape=(5,5),dtype=np.float32)/10
            kernel

Out[292]: array([[0.1, 0.1, 0.1, 0.1, 0.1],
                 [0.1, 0.1, 0.1, 0.1, 0.1],
                 [0.1, 0.1, 0.1, 0.1, 0.1],
                 [0.1, 0.1, 0.1, 0.1, 0.1],
                 [0.1, 0.1, 0.1, 0.1, 0.1]], dtype=float32)

In [293]:  rnjn = cv2.filter2D(img,-1,kernel)
            display(rnjn)
```



17.3 Preprocessing

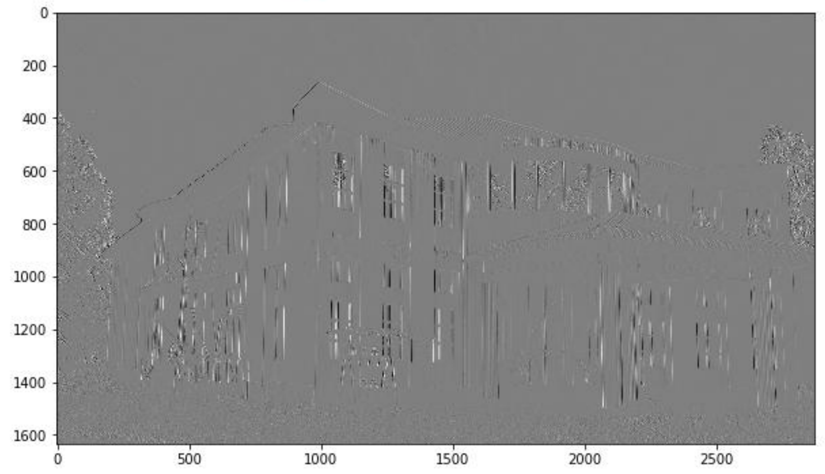
In [291]:
```
img =cv2.imread('D:/Ranjan_Dataset/Images/dept.jpg')
img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
display(img)
```

In [294]:
```
blurred_img = cv2.blur(img,ksize=(5,5))
display(blurred_img)
```

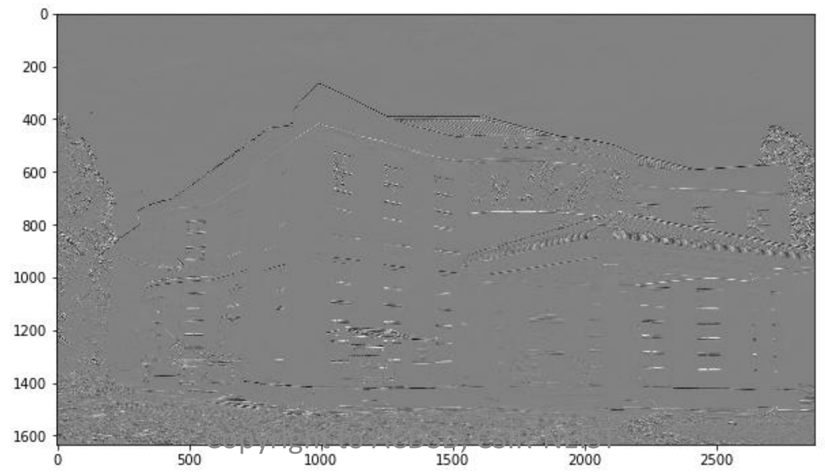
17.3 Preprocessing

- The operator normally takes a single grayscale image as input and generates another grayscale image as output.

In [210]:
```
sobelx = cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=5)
display(sobelx,cmap='gray')
```



In [211]:
```
sobely = cv2.Sobel(img, cv2.CV_64F, 0, 1, ksize=5)
display(sobely,cmap='gray')
```



17.3 Preprocessing

In [207]:
```
img =cv2.imread('D:/Ranjan_Dataset/Images/dept.jpg', 0)
#img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
display(img,cmap='gray')
```

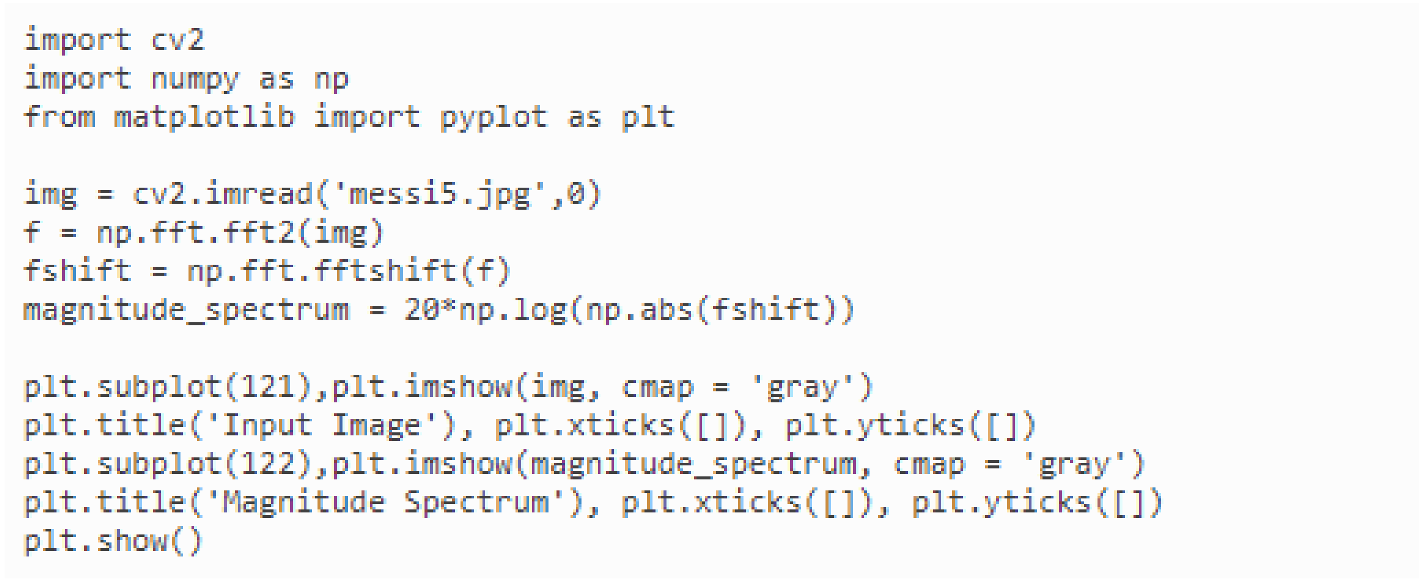
In [209]:
```
blurred_img = cv2.blur(img,ksize=(5,5))
display(blurred_img, cmap='gray')
```

17.4 High level GUI and media

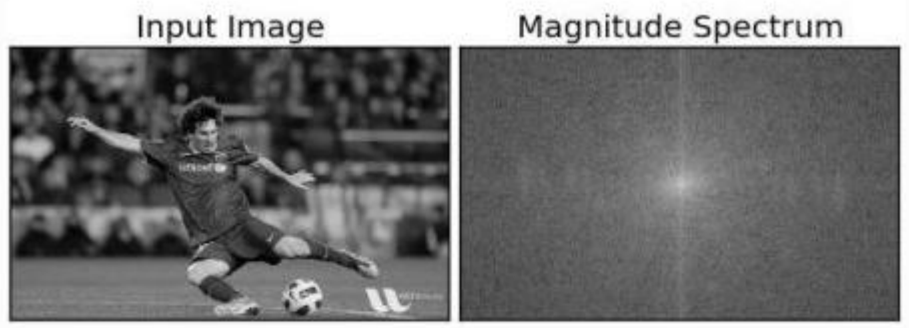
- While OpenCV was designed for use in full-scale applications and can be used within functionally rich UI frameworks (such as Qt, WinForms or Cocoa) or without any UI at all, sometimes there is a need to try some functionality quickly and visualize the results. This is what the High GUI module has been designed for.
- It provides easy interface to:
  - create and manipulate windows that can display images and “remember” their content (no need to handle repaint events from OS)
  - add track bars to the windows, handle simple mouse events as well as keyboard commands
  - read and write images to/from disk or memory.
  - read video from camera or file and write video to a file.

17.5 Image Transforms

- Fourier Transform is used to analyze the frequency characteristics of various filters.
- For images, **2D Discrete Fourier Transform (DFT)** is used to find the frequency domain.
- A fast algorithm called **Fast Fourier Transform (FFT)** is used for calculation of DFT.



```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('messi5.jpg',0)
f = np.fft.fft2(img)
fshift = np.fft.fftshift(f)
magnitude_spectrum = 20*np.log(np.abs(fshift))

plt.subplot(121),plt.imshow(img, cmap = 'gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(magnitude_spectrum, cmap = 'gray')
plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
plt.show()
```



17.6 Image Parts and Segmentation

- **Image Gradients:**

These techniques provide very useful information about the composition of the image.

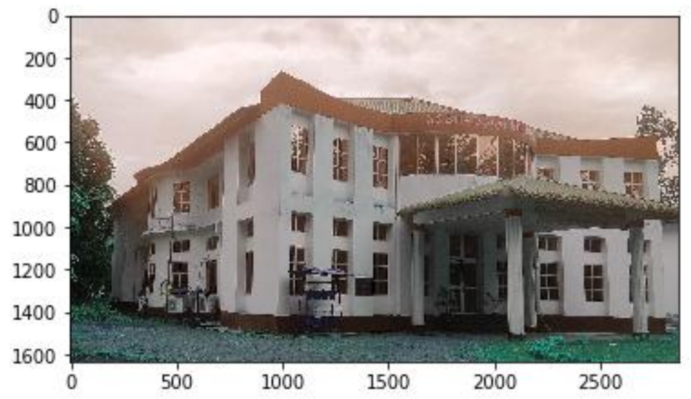
Each pixel of a gradient image measures the change in intensity of that same pixel in the original image, in a given direction.

Having the pixel information we observe that the pixels with the large gradient values become possible edges.

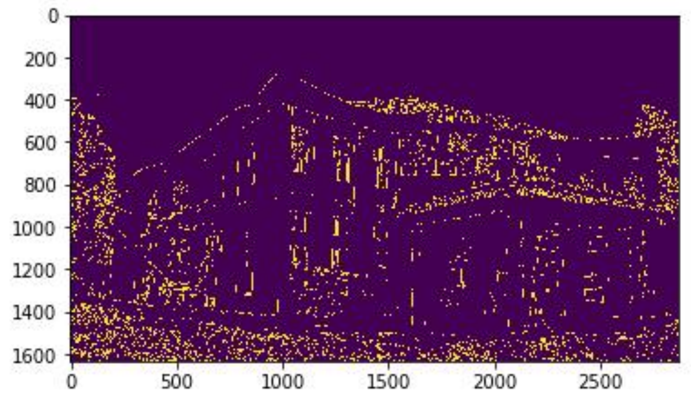
Some well-known edge detection algorithms like the Canny Edge Detector extensively use the gradient image to extract the contours.

For example: Canny Edge Detection

```
In [16]:  img = cv2.imread('D:/Ranjan_Dataset/Images/dept.jpg')
          plt.imshow(img)

Out[16]:  <matplotlib.image.AxesImage at 0x289dc1334c8>
```



```
In [17]:  edges = cv2.Canny(image=img, threshold1=127, threshold2=127)
          plt.imshow(edges)

Out[17]:  <matplotlib.image.AxesImage at 0x289dc1a14c8>
```



17.6 Image Parts and Segmentation

- **Canny Edge Detection:**

OpenCV puts all the above in single function, **cv2.Canny()**.

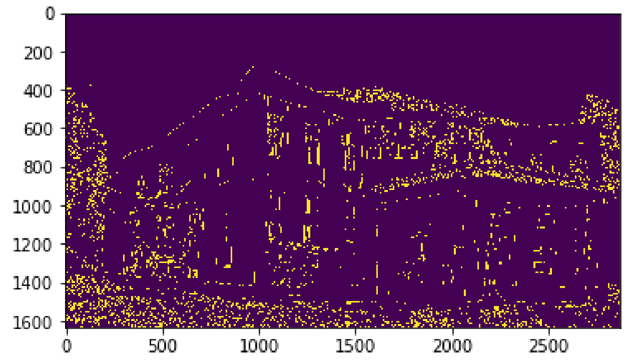
First argument is the input image.

Second and third arguments are the *minVal* and *maxVal* respectively.

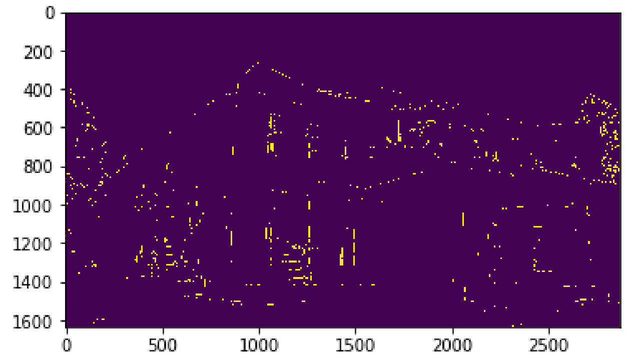
Third argument is *aperture_size*. It is the size of Sobel kernel used for find image gradients. By default it is 3.

Last argument is *L2gradient* which specifies the equation for finding gradient magnitude.

```
In [23]:  edges = cv2.Canny(image=img, threshold1=minm , threshold2=maxm)
          plt.imshow(edges)

Out[23]: <matplotlib.image.AxesImage at 0x289de2ca788>
```



```
In [24]:  blurred_img = cv2.blur(img,ksize=(5,5))

          edges = cv2.Canny(image=blurred_img, threshold1=minm , threshold2=maxm)
          plt.imshow(edges)

Out[24]: <matplotlib.image.AxesImage at 0x289de331c08>
```



17.6 Image Parts and Segmentation

- **Template Matching:**

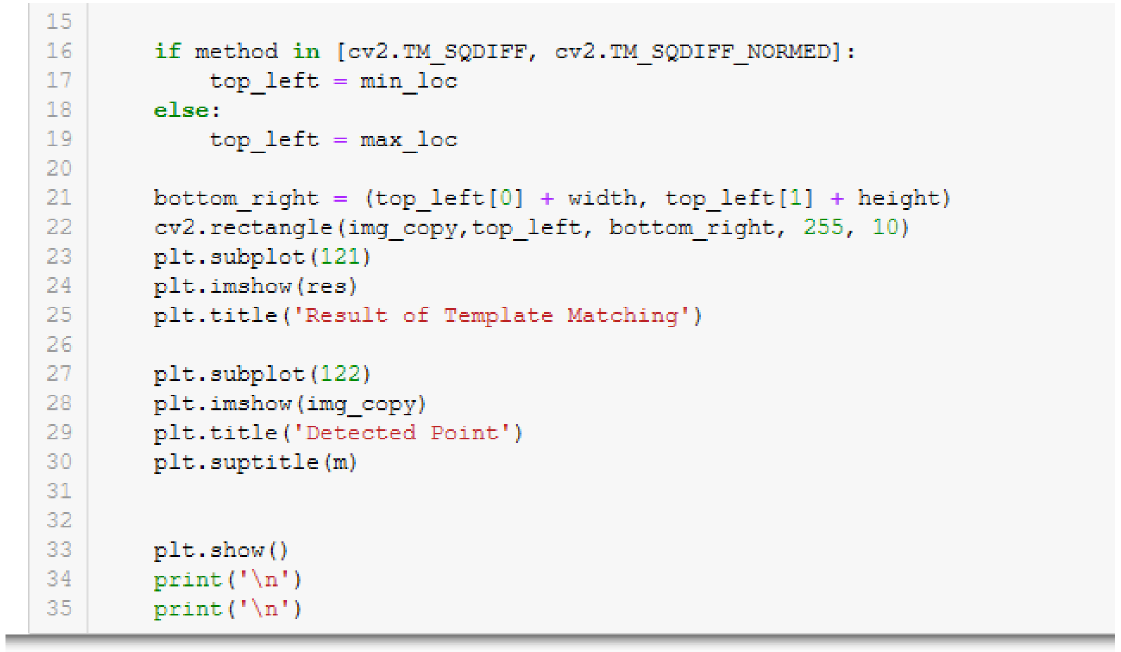
Template Matching is a method for searching and finding the location of a template image in a larger image.

OpenCV comes with a function **cv2.matchTemplate()** for this purpose.

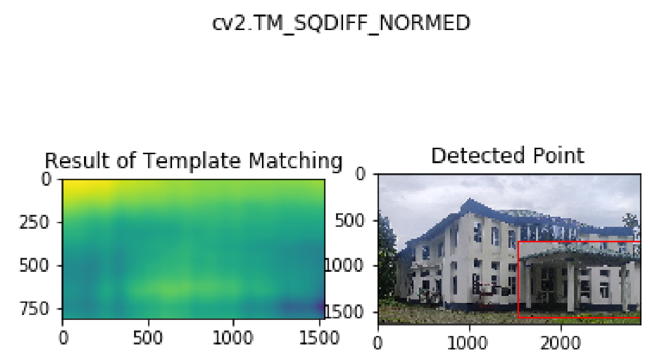
It simply slides the template image over the input image (as in 2D convolution) and compares the template and patch of input image under the template image.

Several comparison methods are implemented in OpenCV.

It returns a grayscale image, where each pixel denotes how much does the neighborhood of that pixel match with template.



```
    if method in [cv2.TM_SQDIFF, cv2.TM_SQDIFF_NORMED]:
        top_left = min_loc
    else:
        top_left = max_loc

    bottom_right = (top_left[0] + width, top_left[1] + height)
    cv2.rectangle(img_copy,top_left, bottom_right, 255, 10)
    plt.subplot(121)
    plt.imshow(res)
    plt.title('Result of Template Matching')

    plt.subplot(122)
    plt.imshow(img_copy)
    plt.title('Detected Point')
    plt.suptitle(m)


    plt.show()
    print('\n')
    print('\n')
```

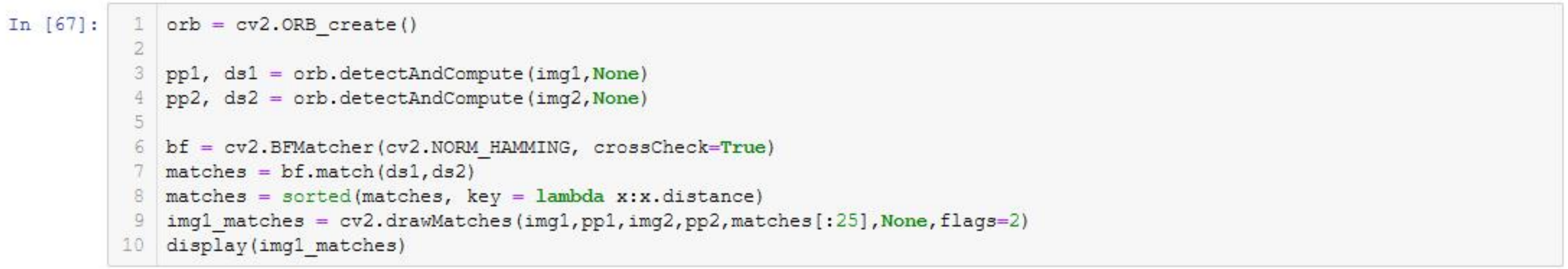


17.6 Image Parts and Segmentation

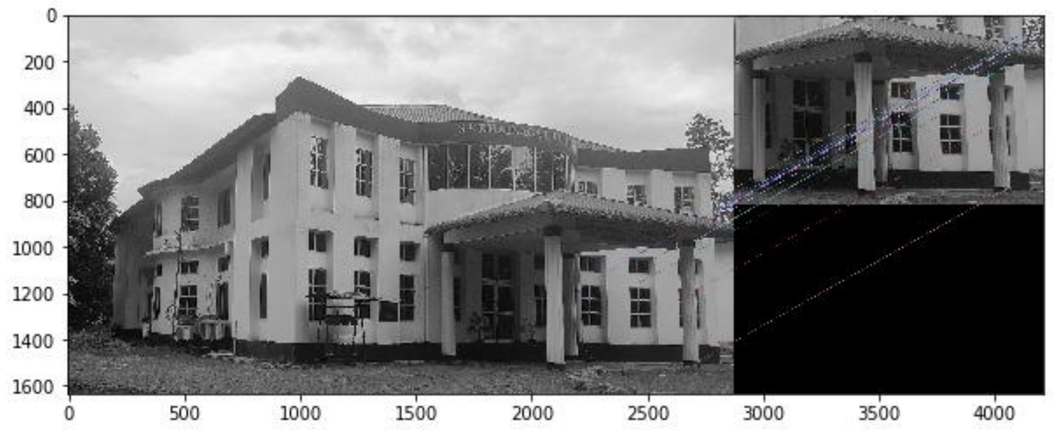
- **Feature detection:**

It includes methods for computing abstractions of image information and making local decisions at every image point whether there is an image feature of a given type at that point or not.

The resulting features will be subsets of the image domain, often in the form of isolated points, continuous curves or connected regions.



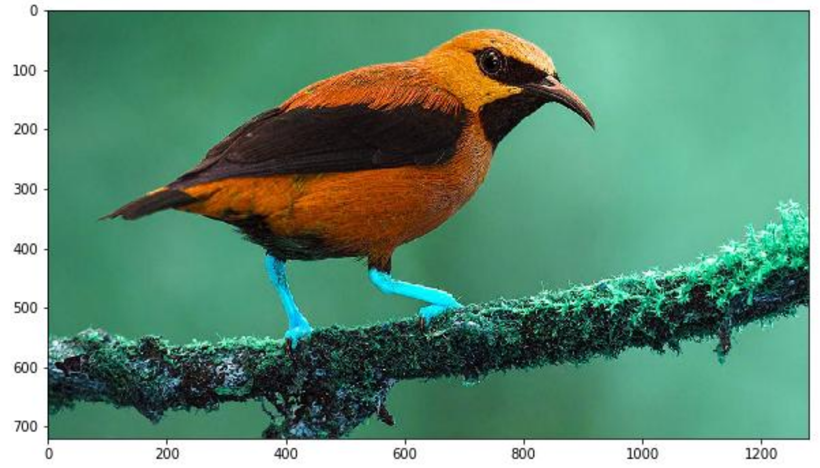
```
In [67]:
orb = cv2.ORB_create()

pp1, ds1 = orb.detectAndCompute(img1,None)
pp2, ds2 = orb.detectAndCompute(img2,None)

bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
matches = bf.match(ds1,ds2)
matches = sorted(matches, key = lambda x:x.distance)
img1_matches = cv2.drawMatches(img1,pp1,img2,pp2,matches[:25],None,flags=2)
display(img1_matches)
```



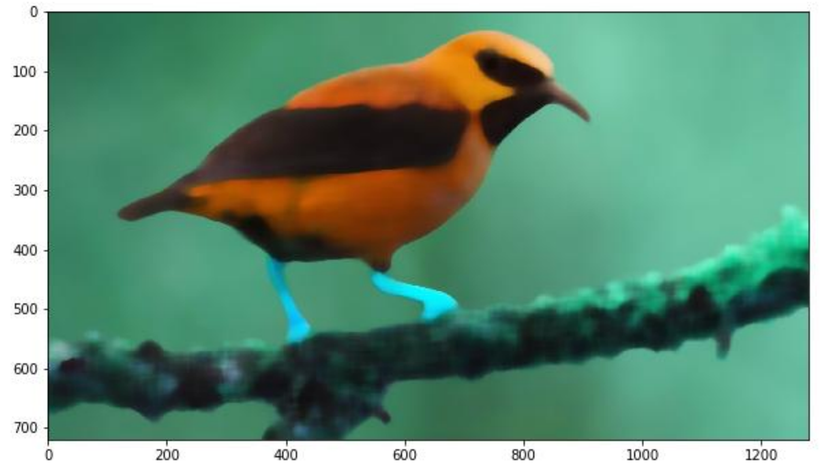
17.6 Image Parts and Segmentation

- In computer vision the term “image segmentation” or simply “segmentation” refers to dividing the image into groups of pixels based on some criteria.

```
In [258]:
img = cv2.imread('D:/Ranjan_Dataset/Images/bird.jpg')
display(img)
```

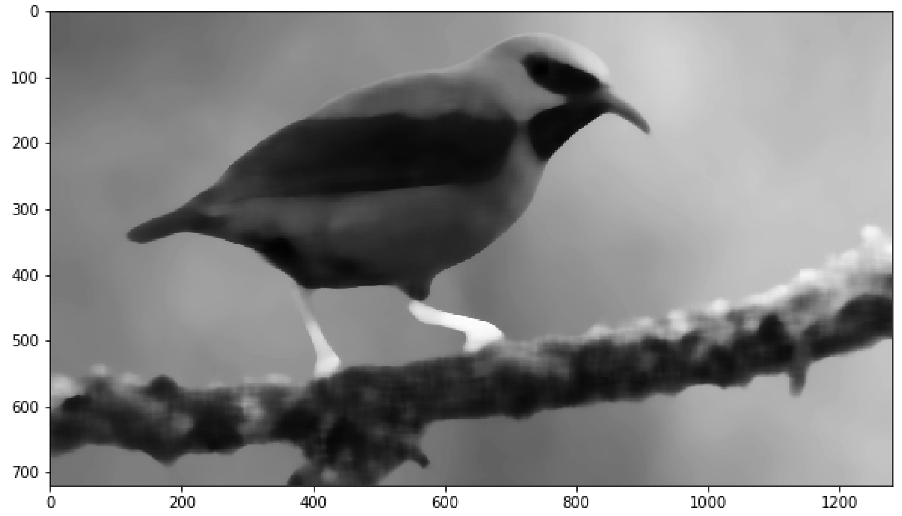


```
In [259]:
img_blur = cv2.medianBlur(img,25)
display(img_blur)
```

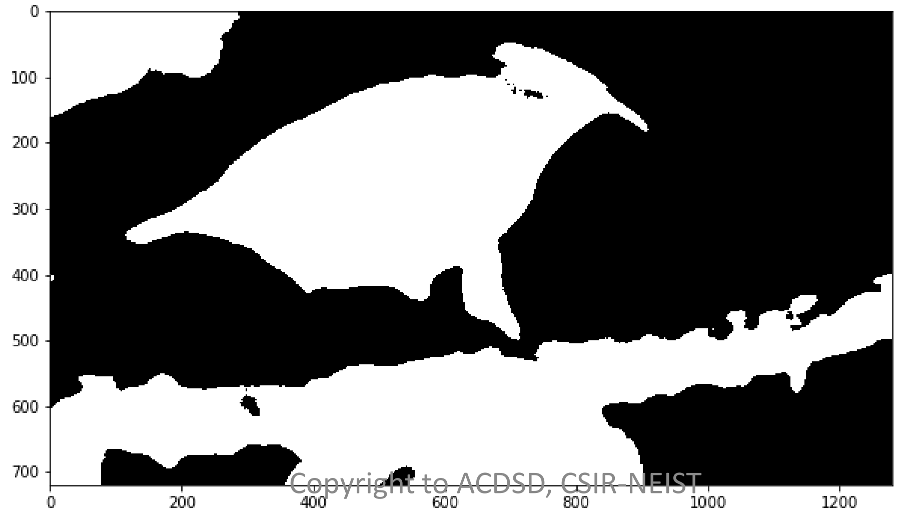


17.6 Image Parts and Segmentation

In [260]:
```
gray_img = cv2.cvtColor(img_blur,cv2.COLOR_BGR2GRAY)
display(gray_img,cmap='gray')
```

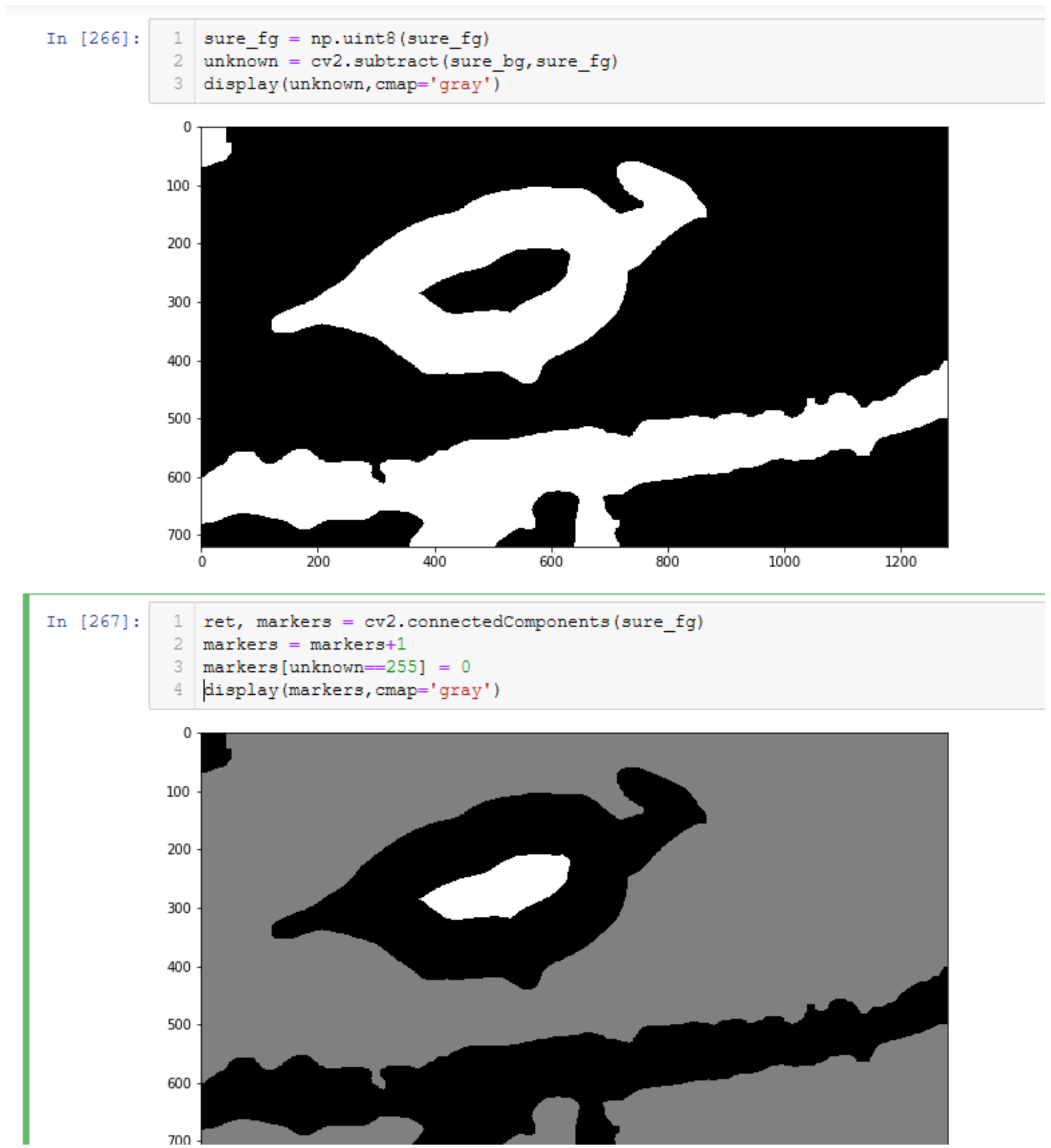


In [261]:
```
ret, img_thresh = cv2.threshold(gray_img,110,55,cv2.THRESH_BINARY_INV)
display(img_thresh,cmap='gray')
```



17.6 Image Parts and Segmentation

- Different output of segmentation:



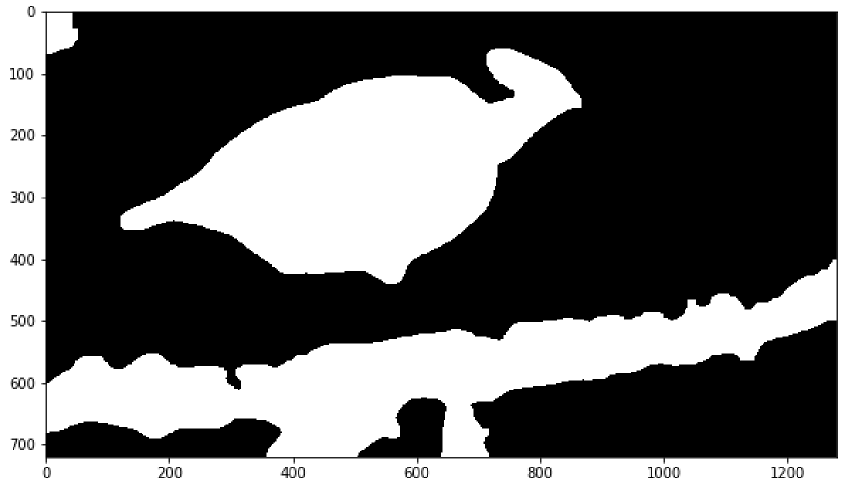
```
In [266]:
sure_fg = np.uint8(sure_fg)
unknown = cv2.subtract(sure_bg,sure_fg)
display(unknown,cmap='gray')
```

```
In [267]:
ret, markers = cv2.connectedComponents(sure_fg)
markers = markers+1
markers[unknown==255] = 0
display(markers,cmap='gray')
```

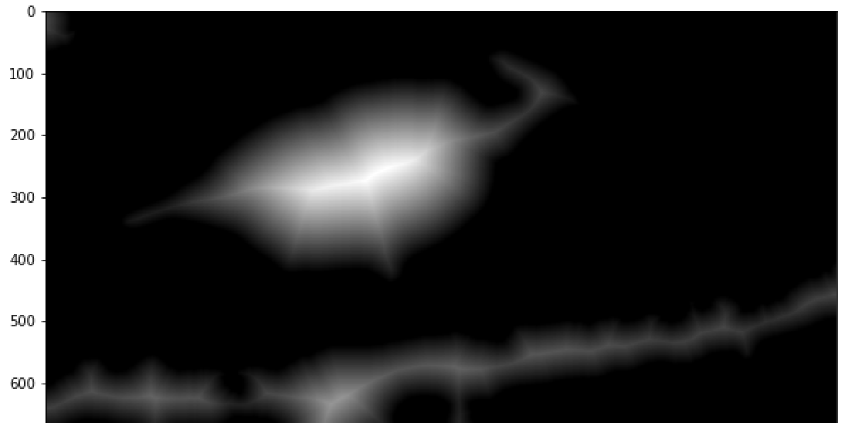
17.6 Image Parts and Segmentation

- Different output of segmentation:

In [264]:
```
#Sure background area
sure_bg = cv2.dilate(opening,kernel,iterations=3)
display(sure_bg,cmap='gray')
```



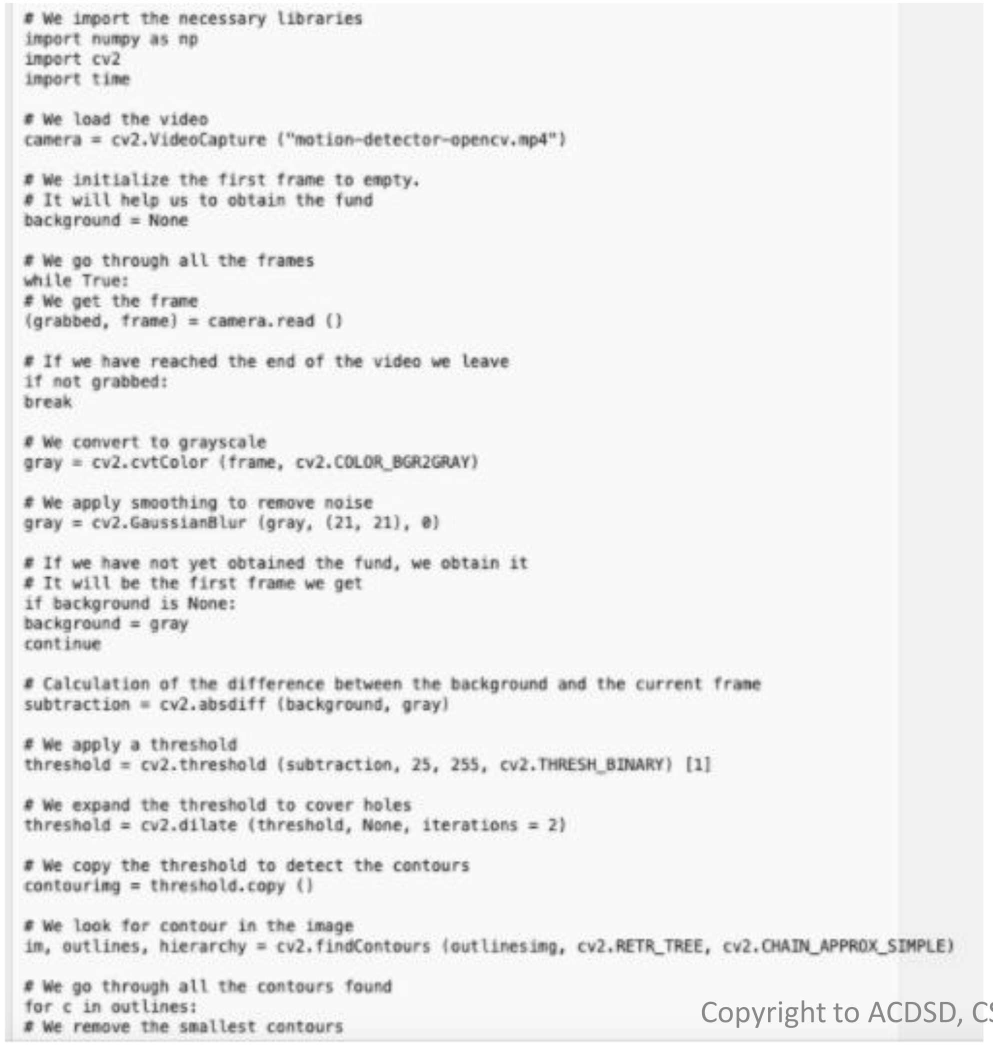
In [265]:
```
#Finding sure foreground area
dist_transform = cv2.distanceTransform(opening,cv2.DIST_L2,5)
ret, sure_fg = cv2.threshold(dist_transform,0.7*dist_transform.max(),255,0)
display(dist_transform,cmap='gray')
```



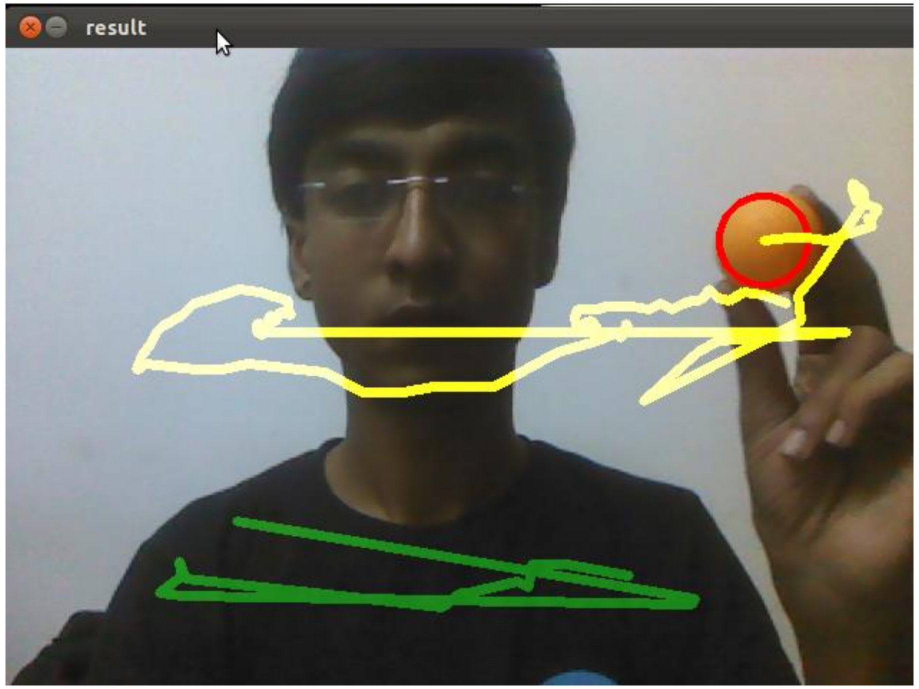
17.7 Tracking and Motion

In many applications based on computer vision, motion detection is used.

For example, when we want to count the people who pass by a certain place or how many cars have passed through a toll. In all these cases, the first thing we have to do is extract the people or vehicles that are at the scene.

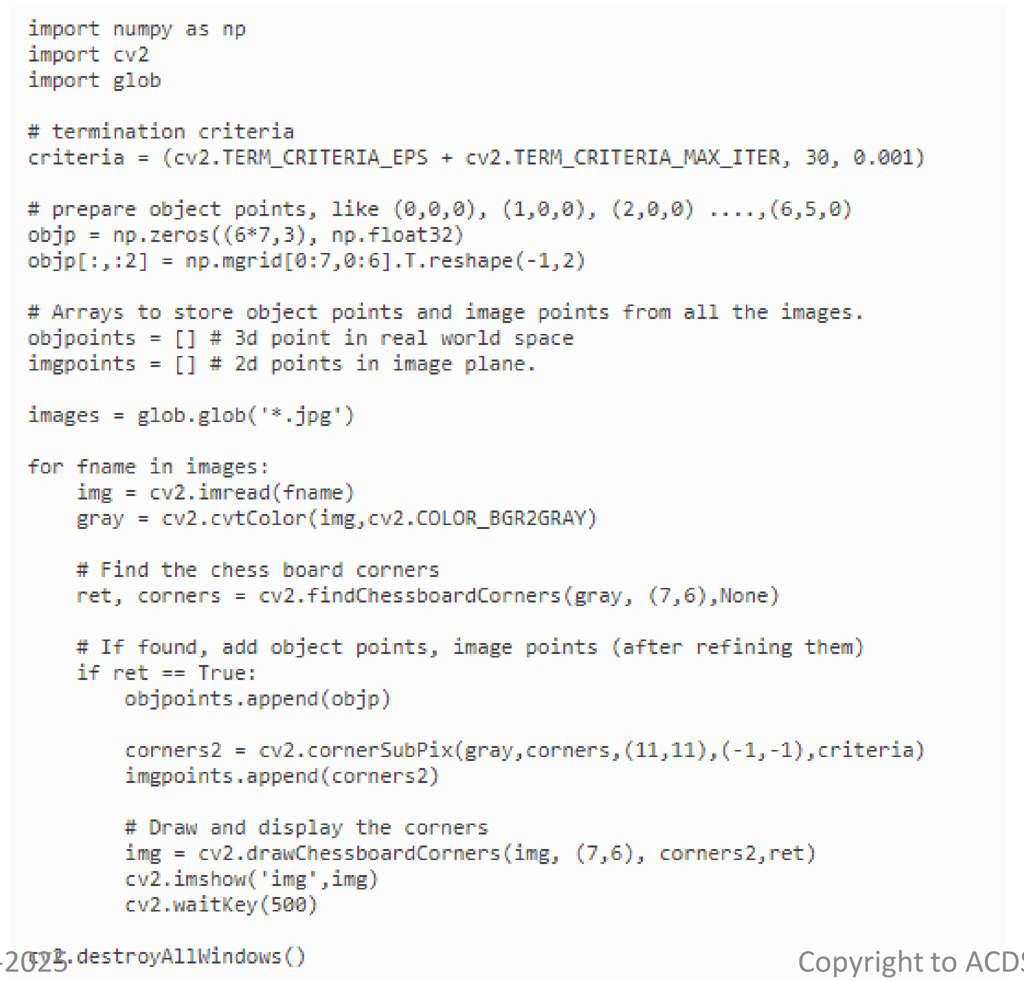


```
# We import the necessary libraries
import numpy as np
import cv2
import time

# We load the video
camera = cv2.VideoCapture ("motion-detector-opencv.mp4")

# We initialize the first frame to empty.
# It will help us to obtain the fund
background = None

# We go through all the frames
while True:
# We get the frame
(grabbed, frame) = camera.read ()

# If we have reached the end of the video we leave
if not grabbed:
break

# We convert to grayscale
gray = cv2.cvtColor (frame, cv2.COLOR_BGR2GRAY)

# We apply smoothing to remove noise
gray = cv2.GaussianBlur (gray, (21, 21), 0)

# If we have not yet obtained the fund, we obtain it
# It will be the first frame we get
if background is None:
background = gray
continue

# Calculation of the difference between the background and the current frame
subtraction = cv2.absdiff (background, gray)

# We apply a threshold
threshold = cv2.threshold (subtraction, 25, 255, cv2.THRESH_BINARY) [1]

# We expand the threshold to cover holes
threshold = cv2.dilate (threshold, None, iterations = 2)

# We copy the threshold to detect the contours
contourimg = threshold.copy ()

# We look for contour in the image
im, outlines, hierarchy = cv2.findContours (outlinesimg, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

# We go through all the contours found
for c in outlines:
# We remove the smallest contours
```

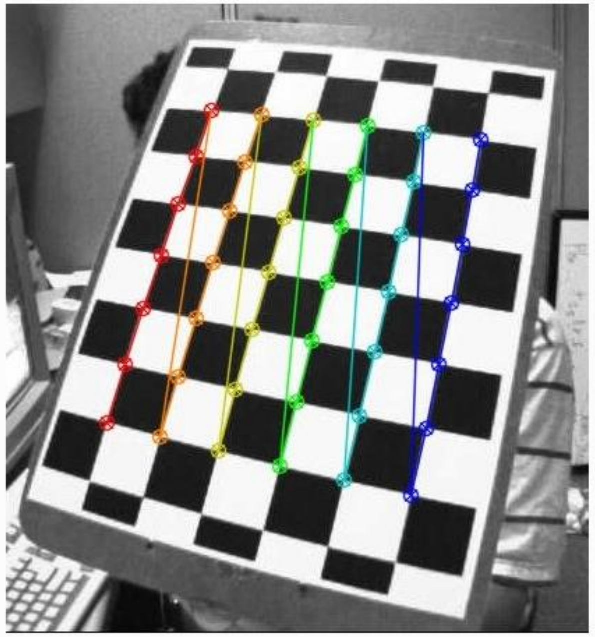


17.8 Projection and 3D vision

A **3D projection** (or **graphical projection**) is a design technique used to display a three-dimensional (3D) object on a two-dimensional (2D) surface.
These projections rely on visual perspective and aspect analysis to project a complex object for viewing capability on a simpler plane.

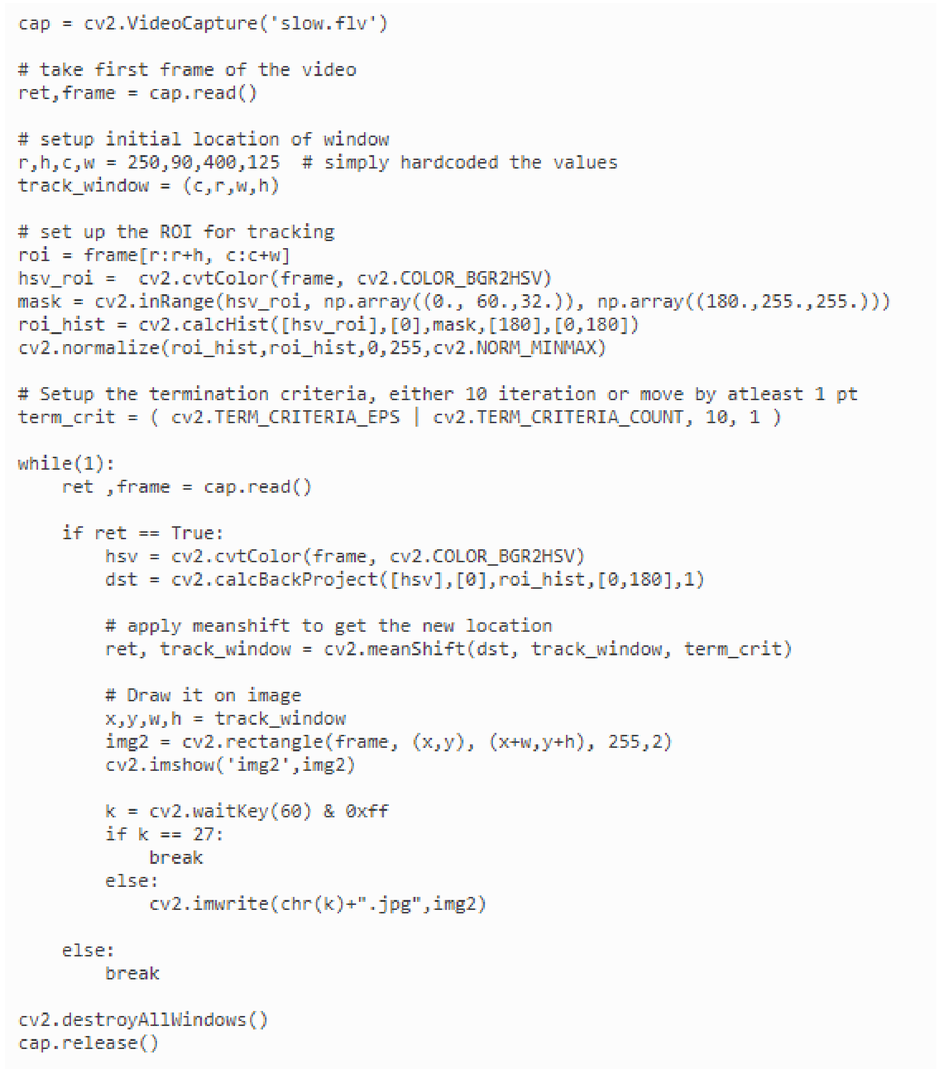


```
import numpy as np
import cv2
import glob

# termination criteria
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)

# prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)
objp = np.zeros((6*7,3), np.float32)
objp[:,:2] = np.mgrid[0:7,0:6].T.reshape(-1,2)

# Arrays to store object points and image points from all the images.
objpoints = [] # 3d point in real world space
imgpoints = [] # 2d points in image plane.

images = glob.glob('*.jpg')

for fname in images:
    img = cv2.imread(fname)
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

    # Find the chess board corners
    ret, corners = cv2.findChessboardCorners(gray, (7,6),None)

    # If found, add object points, image points (after refining them)
    if ret == True:
        objpoints.append(objp)

        corners2 = cv2.cornerSubPix(gray,corners,(11,11),(-1,-1),criteria)
        imgpoints.append(corners2)

        # Draw and display the corners
        img = cv2.drawChessboardCorners(img, (7,6), corners2,ret)
        cv2.imshow('img',img)
        cv2.waitKey(500)

cv2.destroyAllWindows()
```

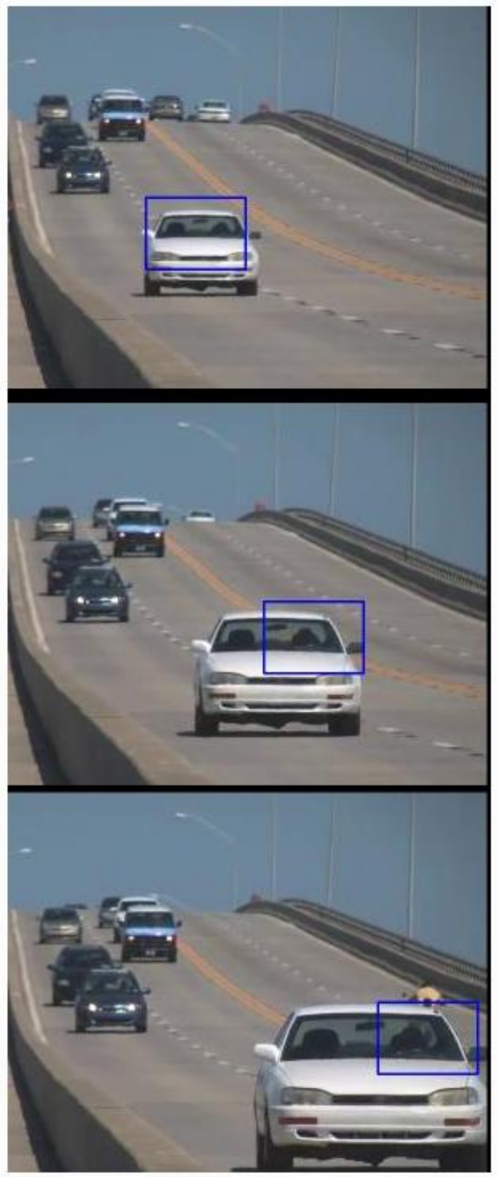


17.9 Video Processing

Computer vision is a huge part of the data science/AI domain. Sometimes, computer vision engineers have to deal with videos.



```
cap = cv2.VideoCapture('slow.flv')

# take first frame of the video
ret,frame = cap.read()

# setup initial location of window
r,h,c,w = 250,90,400,125  # simply hardcoded the values
track_window = (c,r,w,h)

# set up the ROI for tracking
roi = frame[r:r+h, c:c+w]
hsv_roi =  cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
mask = cv2.inRange(hsv_roi, np.array((0., 60.,32.)), np.array((180.,255.,255.)))
roi_hist = cv2.calcHist([hsv_roi],[0],mask,[180],[0,180])
cv2.normalize(roi_hist,roi_hist,0,255,cv2.NORM_MINMAX)

# Setup the termination criteria, either 10 iteration or move by atleast 1 pt
term_crit = ( cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 1 )

while(1):
    ret ,frame = cap.read()

    if ret == True:
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
        dst = cv2.calcBackProject([hsv],[0],roi_hist,[0,180],1)

        # apply meanshift to get the new location
        ret, track_window = cv2.meanShift(dst, track_window, term_crit)

        # Draw it on image
        x,y,w,h = track_window
        img2 = cv2.rectangle(frame, (x,y), (x+w,y+h), 255,2)
        cv2.imshow('img2',img2)

        k = cv2.waitKey(60) & 0xff
        if k == 27:
            break
        else:
            cv2.imwrite(chr(k)+".jpg",img2)

    else:
        break

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

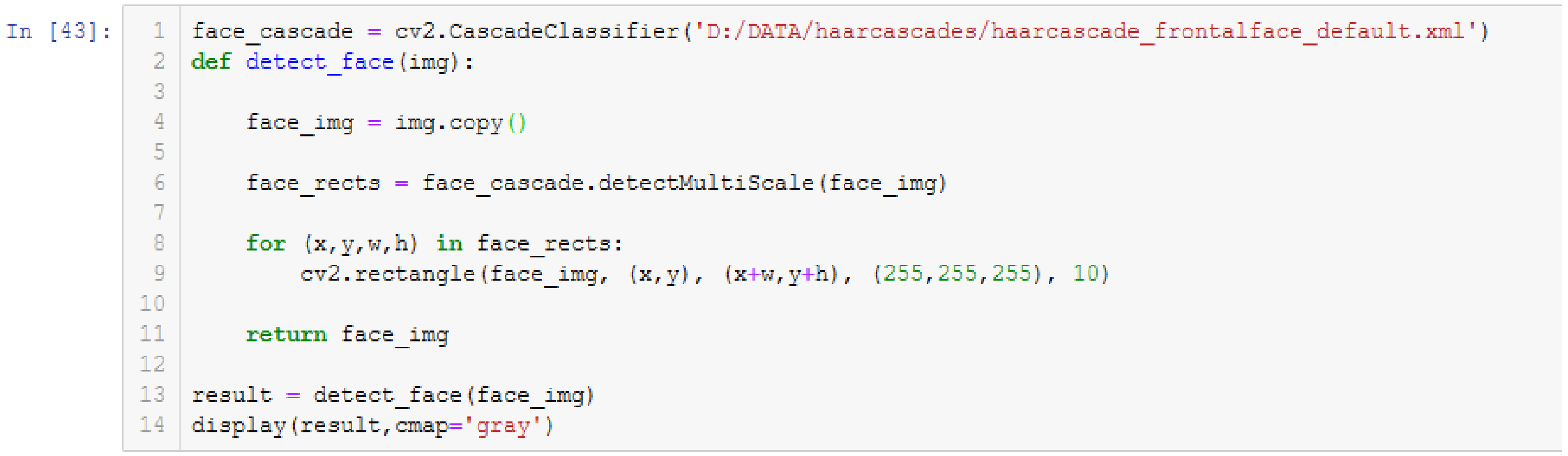


17.10 Object Detection & Recognition

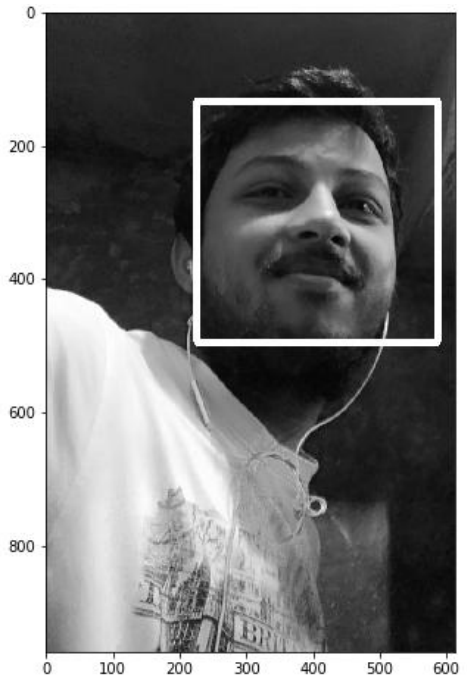
- **Face Detection:**

This is a computer technology being used in a variety of applications that identifies human faces in digital images.

Face detection also refers to the psychological process by which humans locate and attend to faces in a visual scene.



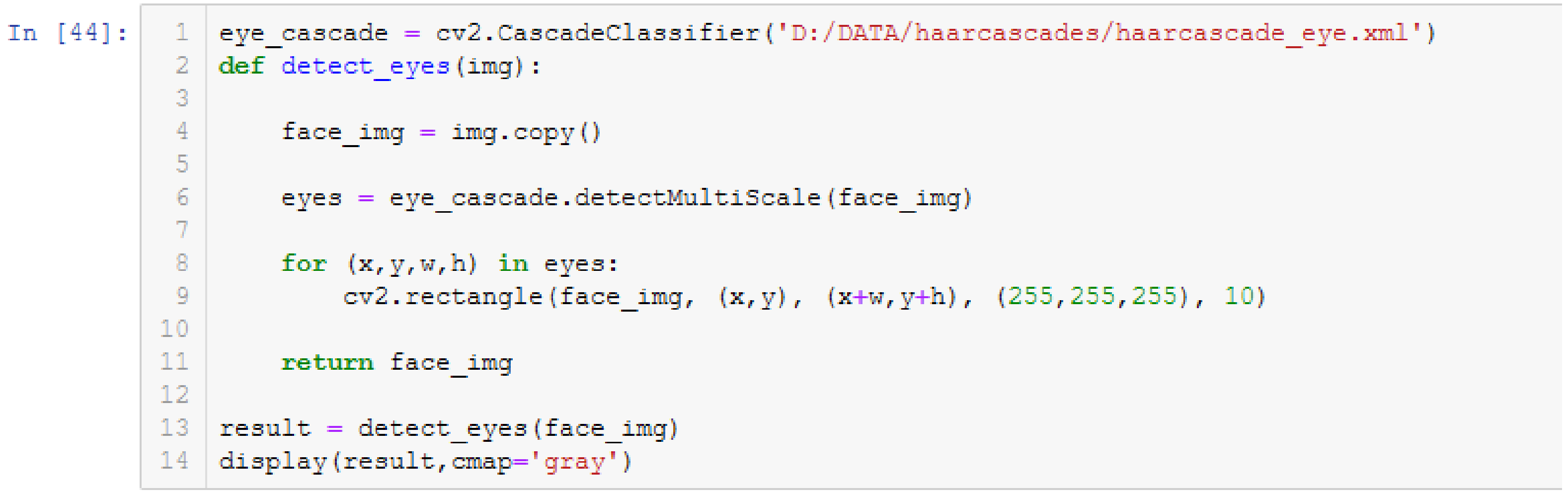
```
In [43]:
face_cascade = cv2.CascadeClassifier('D:/DATA/haarcascades/haarcascade_frontalface_default.xml')
def detect_face(img):

    face_img = img.copy()

    face_rects = face_cascade.detectMultiScale(face_img)

    for (x,y,w,h) in face_rects:
        cv2.rectangle(face_img, (x,y), (x+w,y+h), (255,255,255), 10)

    return face_img

result = detect_face(face_img)
display(result,cmap='gray')
```



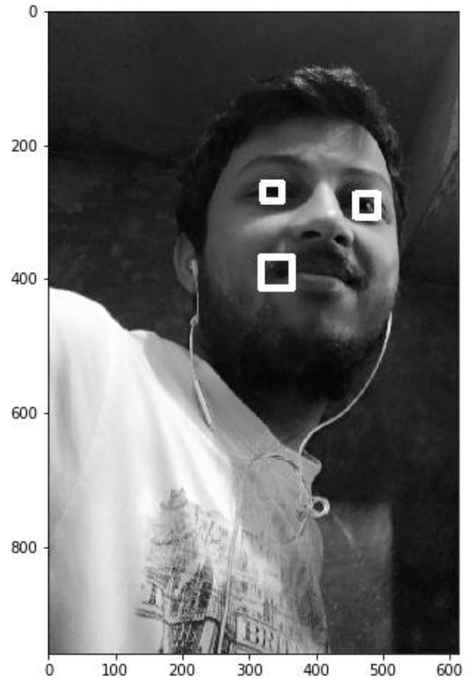
17.10 Object Detection & Recognition

- **Eye Detection:**

Eye tracking refers to the process of measuring where we look, also known as our point of gaze. These measurements are carried out by an eye tracker, that records the position of the eyes and the movements they make.

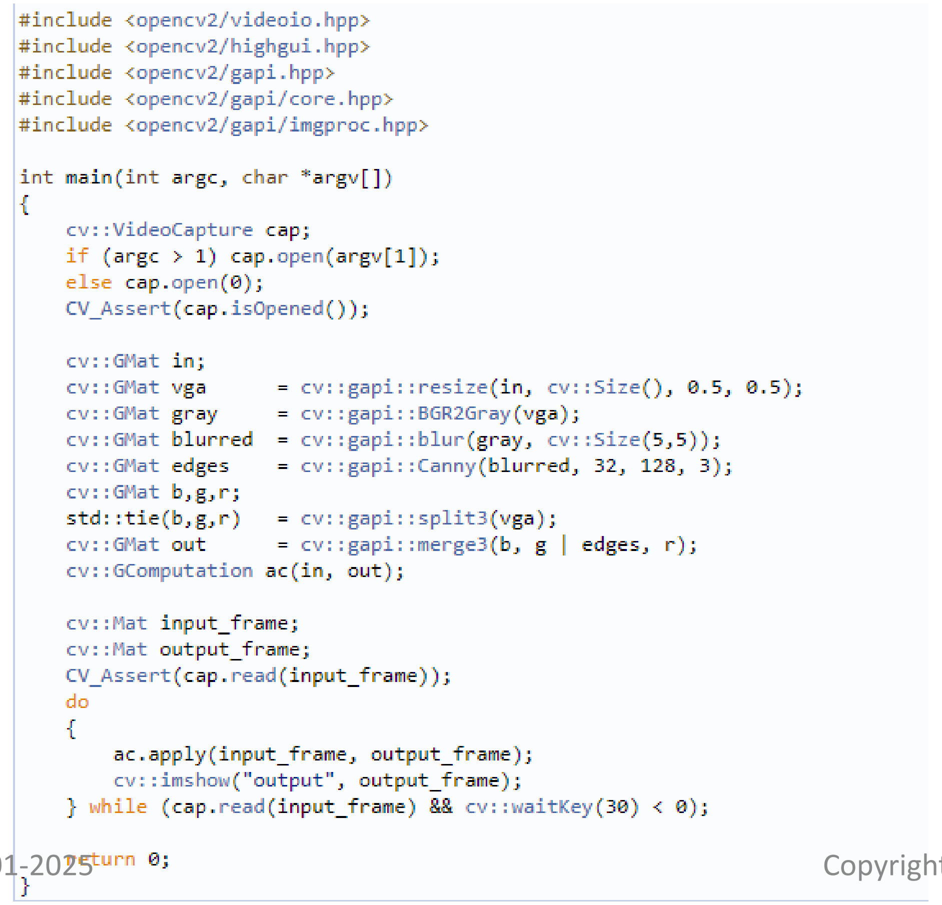


```
In [44]:
eye_cascade = cv2.CascadeClassifier('D:/DATA/haarcascades/haarcascade_eye.xml')
def detect_eyes(img):

    face_img = img.copy()

    eyes = eye_cascade.detectMultiScale(face_img)

    for (x,y,w,h) in eyes:
        cv2.rectangle(face_img, (x,y), (x+w,y+h), (255,255,255), 10)

    return face_img

result = detect_eyes(face_img)
display(result,cmap='gray')
```

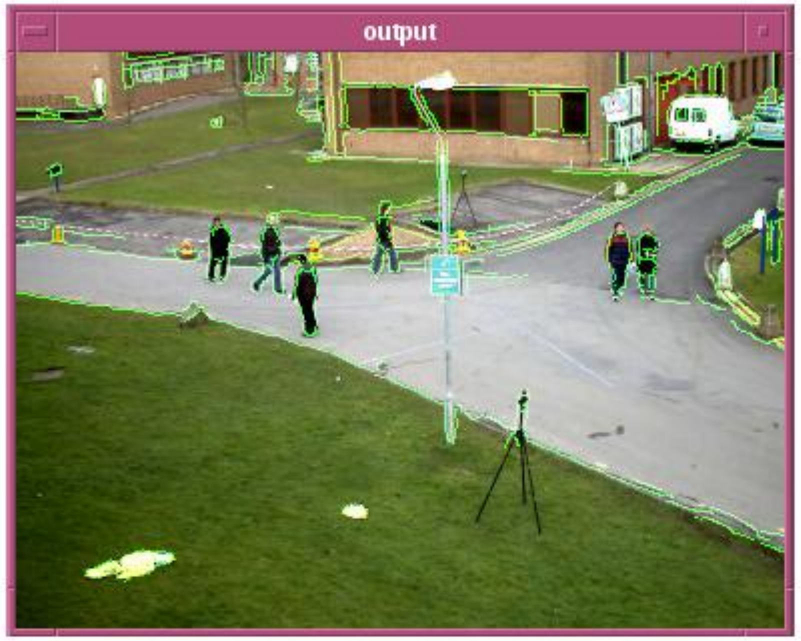


17.11 Graph API

- OpenCV Graph API (or G-API) is a new OpenCV module targeted to make regular image processing fast and portable. These two goals are achieved by introducing a new graph-based model of execution.
- G-API is a special module in OpenCV – in contrast with the majority of other main modules, this one acts as a framework rather than some specific CV algorithm. G-API provides means to define CV operations, construct graphs (in form of expressions) using it, and finally implement and run the operations for a particular backend.



```
#include <opencv2/videoio.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/gapi.hpp>
#include <opencv2/gapi/core.hpp>
#include <opencv2/gapi/imgproc.hpp>

int main(int argc, char *argv[])
{
    cv::VideoCapture cap;
    if (argc > 1) cap.open(argv[1]);
    else cap.open(0);
    CV_Assert(cap.isOpened());

    cv::GMat in;
    cv::GMat vga      = cv::gapi::resize(in, cv::Size(), 0.5, 0.5);
    cv::GMat gray     = cv::gapi::BGR2Gray(vga);
    cv::GMat blurred  = cv::gapi::blur(gray, cv::Size(5,5));
    cv::GMat edges    = cv::gapi::Canny(blurred, 32, 128, 3);
    cv::GMat b,g,r;
    std::tie(b,g,r)   = cv::gapi::split3(vga);
    cv::GMat out      = cv::gapi::merge3(b, g | edges, r);
    cv::GComputation ac(in, out);

    cv::Mat input_frame;
    cv::Mat output_frame;
    CV_Assert(cap.read(input_frame));
    do
    {
        ac.apply(input_frame, output_frame);
        cv::imshow("output", output_frame);
    } while (cap.read(input_frame) && cv::waitKey(30) < 0);

    return 0;
}
```



17.12 Computational Photography

Image Denoising

See a good technique to remove noises in images called Non-Local Means Denoising

Image Impainting

Do you have a old degraded photo with many black spots and strokes on it? Take it. Let's try to restore them with a technique called image inpainting.

High Dynamic Range (HRD)

Learn how to merge exposure sequence and process high dynamic range images.

17.13 GPU- Accelerated computer vision

- OpenCV is a free library for research and commercial purposes that includes hundreds of optimized computer vision and image processing algorithms.
- NVIDIA and Itseez have optimized many OpenCV functions using CUDA on desktop machines equipped with NVIDIA GPUs. These functions are 5 to 100 times faster in wall-clock time compared to their CPU counterparts.
- Anatoly Baksheev, OpenCV GPU Module Team Leader at Itseez, demonstrates how to obtain and build OpenCV, its GPU module, and the sample programs.

17.14 OpenCV iOS

- OpenCV for iOS is a practical guide that walks one through every important step for building a computer vision application for the iOS platform.
- It will help to port OpenCV code, profile and optimize it, and wrap it into a GUI application.
- Each recipe is accompanied by a sample project or an example that helps focus on a particular aspect of the technology.
- OpenCV for iOS starts by creating a simple iOS application and linking OpenCV before moving on to processing images and videos in real-time.
- It covers the major ways to retrieve images, process them, and view or export results.
- Special attention is also given to performance issues, as they greatly affect the user experience.

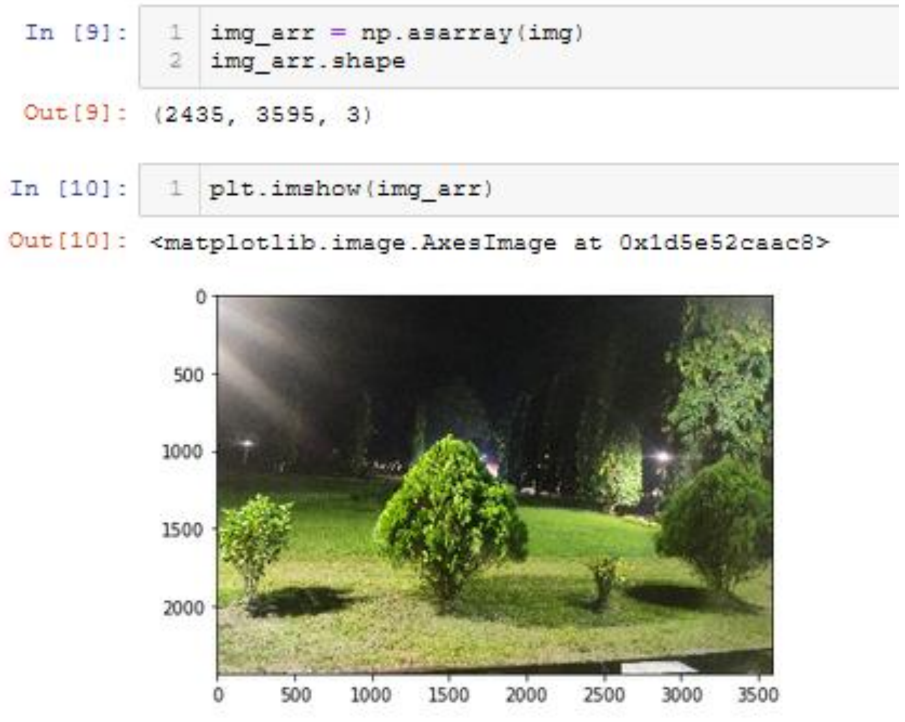
Worked out Examples

Question 1: Write codes in OpenCV to perform the following –

a) Reading and displaying an image

b) Type and shape of image

Answer:

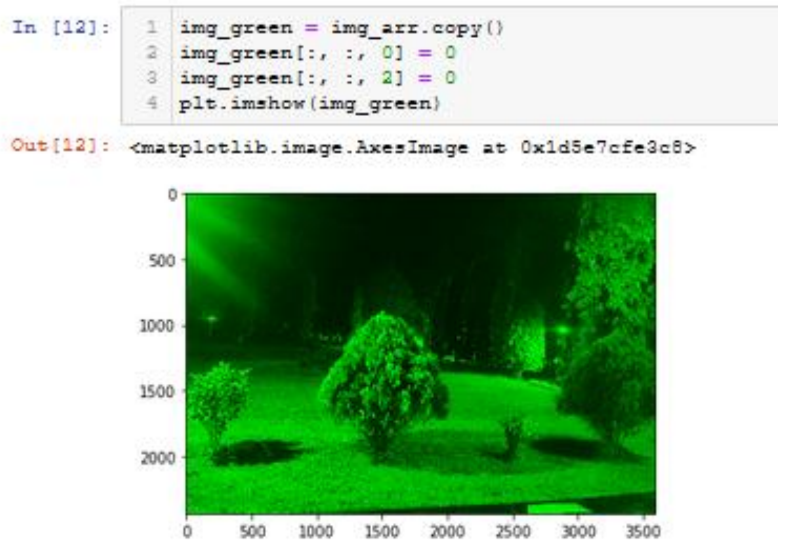


```
In [9]:  img_arr = np.asarray(img)
         img_arr.shape

Out[9]:  (2435, 3595, 3)

In [10]: plt.imshow(img_arr)

Out[10]: <matplotlib.image.AxesImage at 0x1d5e52caac8>
```

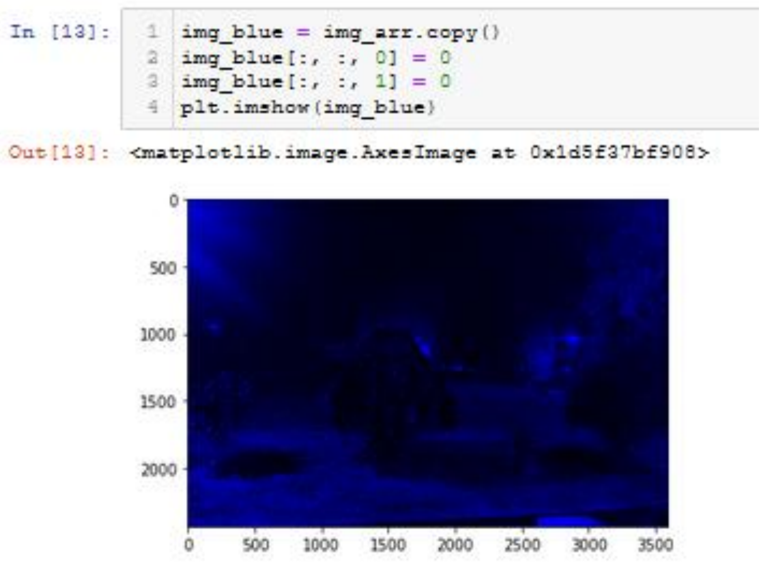
Worked out Examples

Question 2: Write codes in OpenCV to perform color transformation in an image?

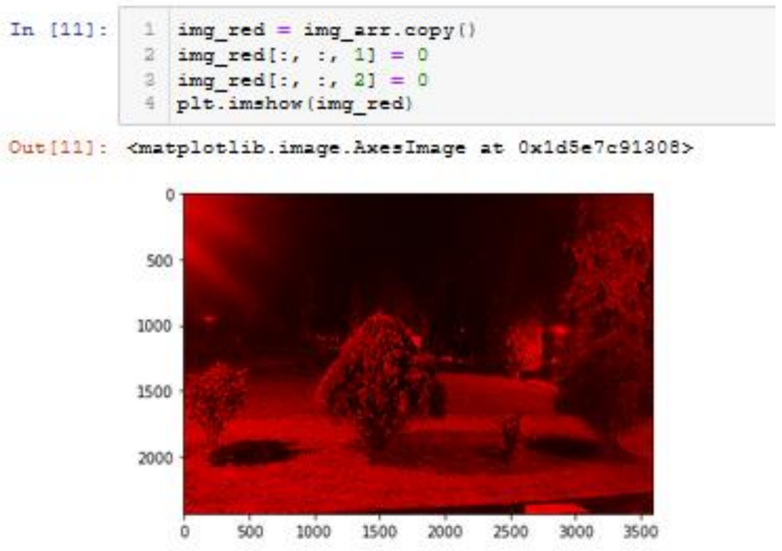
Answer:



```
In [12]:  img_green = img_arr.copy()
          img_green[:, :, 0] = 0
          img_green[:, :, 2] = 0
          plt.imshow(img_green)

Out[12]:  <matplotlib.image.AxesImage at 0x1d5e7cfe3c8>
```



```
In [13]:  img_blue = img_arr.copy()
          img_blue[:, :, 0] = 0
          img_blue[:, :, 1] = 0
          plt.imshow(img_blue)

Out[13]:  <matplotlib.image.AxesImage at 0x1d5f37bf908>
```

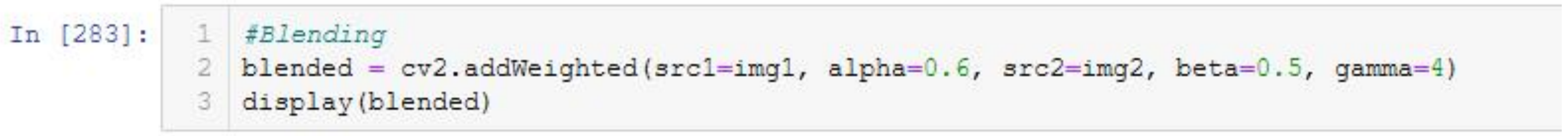


```
In [11]:  img_red = img_arr.copy()
          img_red[:, :, 1] = 0
          img_red[:, :, 2] = 0
          plt.imshow(img_red)

Out[11]:  <matplotlib.image.AxesImage at 0x1d5e7c91308>
```

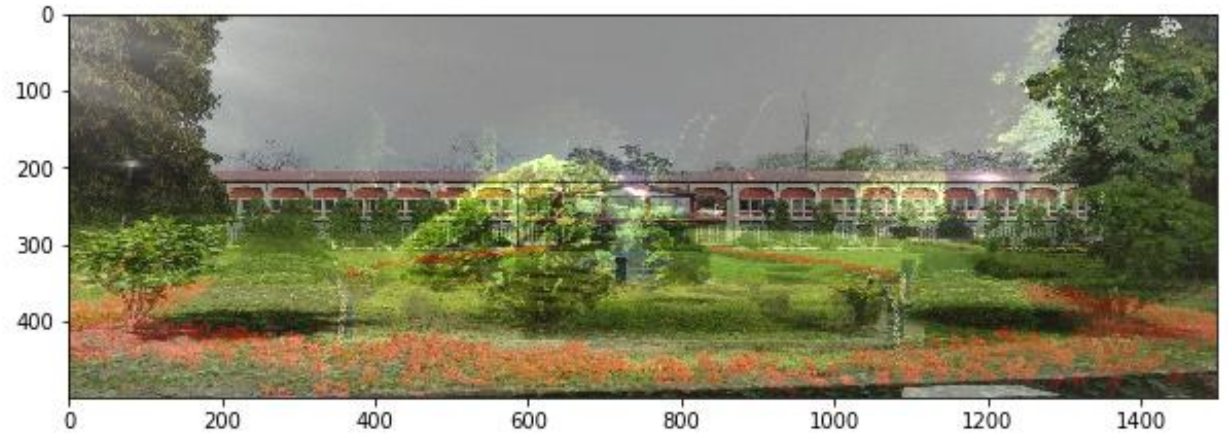
Worked out Examples

Question 3: How will you perform the blending operation in OpenCV?

Answer:



```
In [283]:
#Blending
blended = cv2.addWeighted(src1=img1, alpha=0.6, src2=img2, beta=0.5, gamma=4)
display(blended)
```

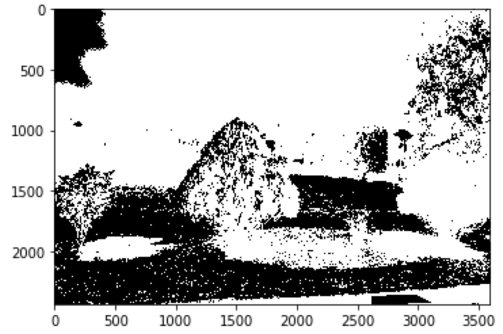


Worked out Examples

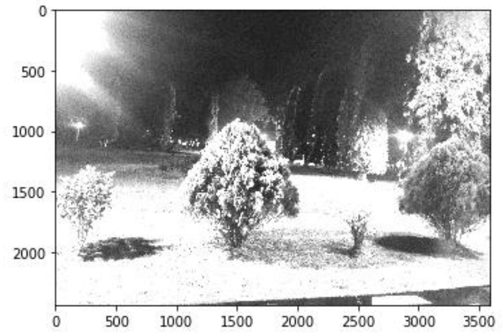
Question 4: Perform different thresolding operations available in OpenCV?

Answer:

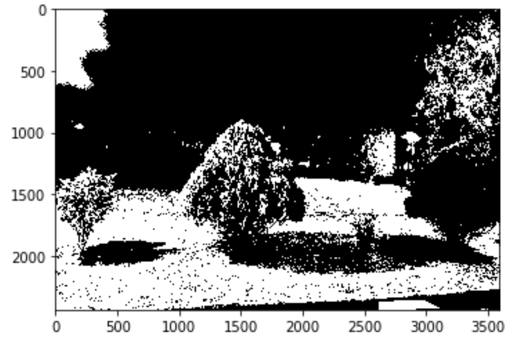
```
In [46]:  #Binary Inverse
          ret,thresh2 = cv2.threshold(img,127,255,cv2.THRESH_BINARY_INV)
          plt.imshow(thresh2,cmap='gray')

Out[46]: <matplotlib.image.AxesImage at 0x1f90d0a0f88>
```



```
In [47]:  #Threshold Truncation
          ret,thresh3 = cv2.threshold(img,127,255,cv2.THRESH_TRUNC)
          plt.imshow(thresh3,cmap='gray')

Out[47]: <matplotlib.image.AxesImage at 0x1f90d109608>
```



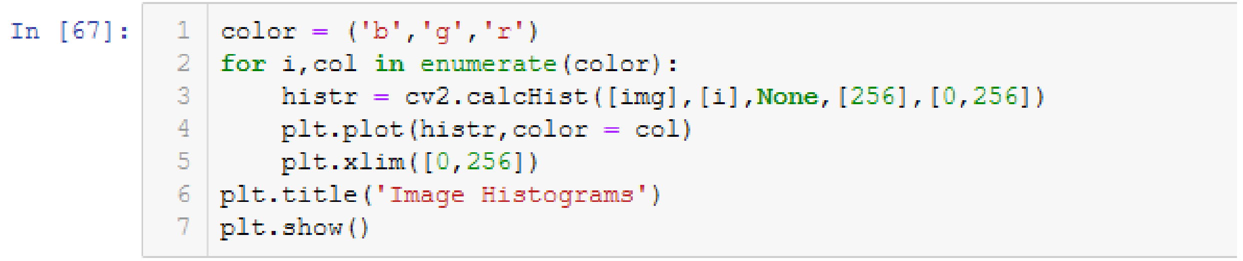
```
In [45]:  #Simple Binary
          ret,thresh1 = cv2.threshold(img,127,127,cv2.THRESH_BINARY)
          ret
          plt.imshow(thresh1,cmap='gray')

Out[45]: <matplotlib.image.AxesImage at 0x1f90d03bf88>
```



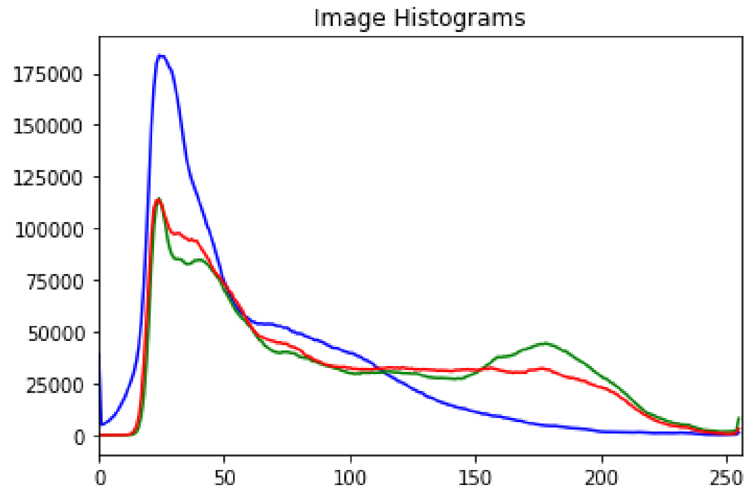
Worked out Examples

Question 5: How will you check the different color distributions in an image?

Answer:



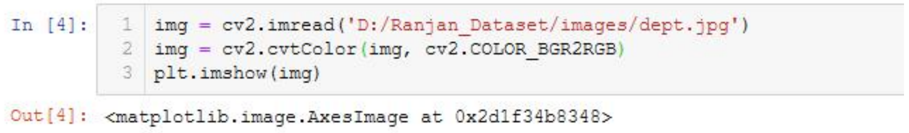
```
In [67]:
color = ('b','g','r')
for i,col in enumerate(color):
    histr = cv2.calcHist([img],[i],None,[256],[0,256])
    plt.plot(histr,color = col)
    plt.xlim([0,256])
plt.title('Image Histograms')
plt.show()
```



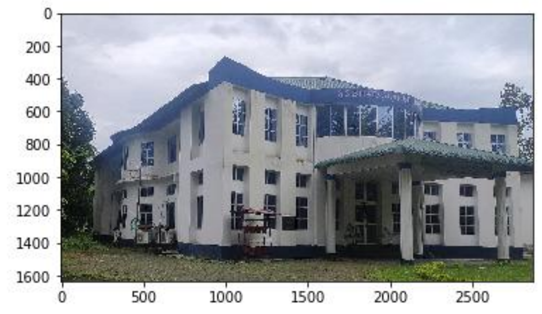
Worked out Examples

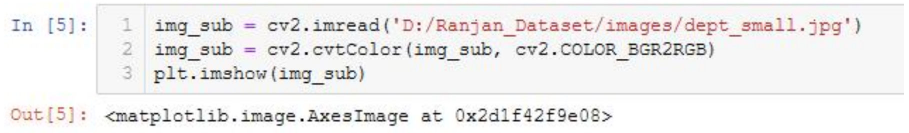
Question 6: Write a piece of code to implement template matching in OpenCV?

Answer:

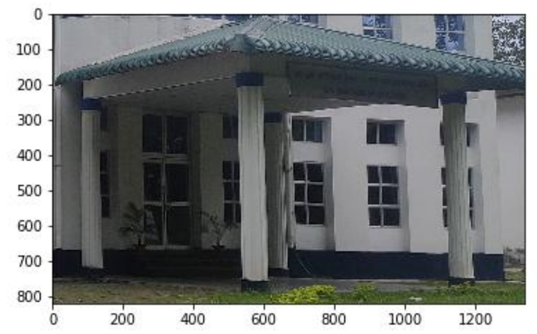


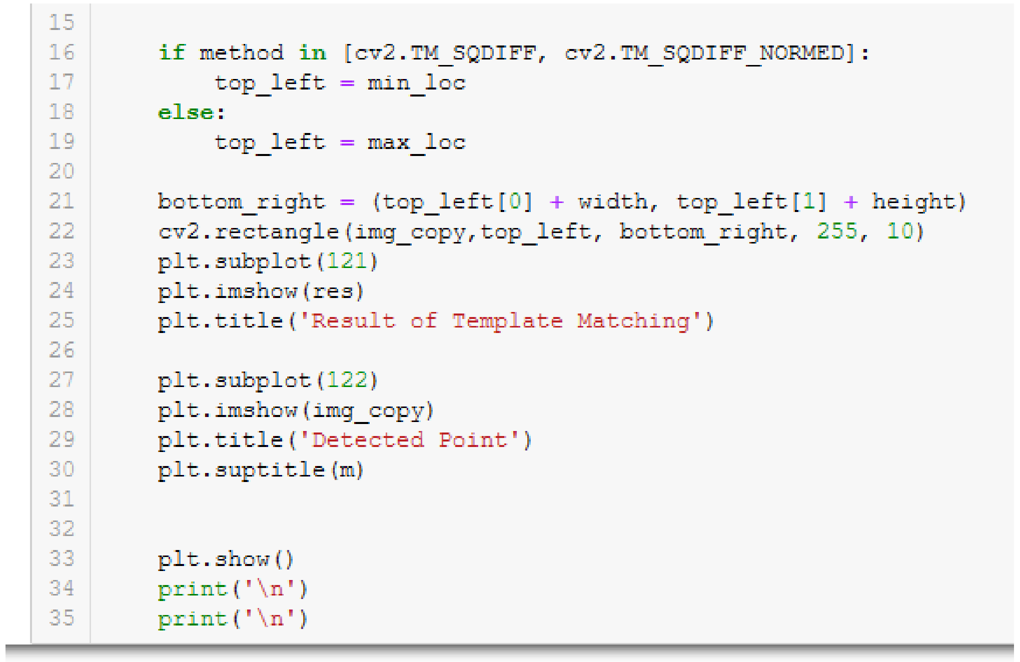
```
In [4]:  img = cv2.imread('D:/Ranjan_Dataset/images/dept.jpg')
         img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
         plt.imshow(img)

Out[4]: <matplotlib.image.AxesImage at 0x2d1f34b8348>
```



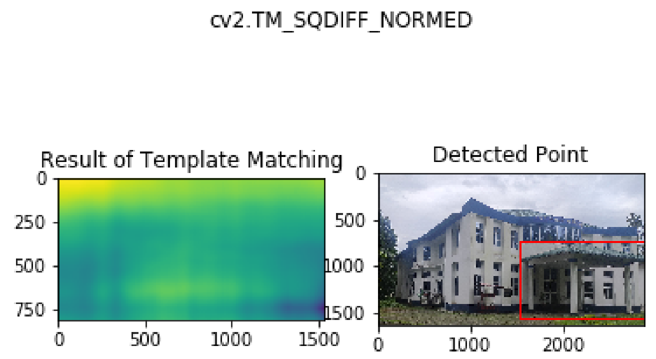


```
In [5]:  img_sub = cv2.imread('D:/Ranjan_Dataset/images/dept_small.jpg')
         img_sub = cv2.cvtColor(img_sub, cv2.COLOR_BGR2RGB)
         plt.imshow(img_sub)

Out[5]: <matplotlib.image.AxesImage at 0x2d1f42f9e08>
```



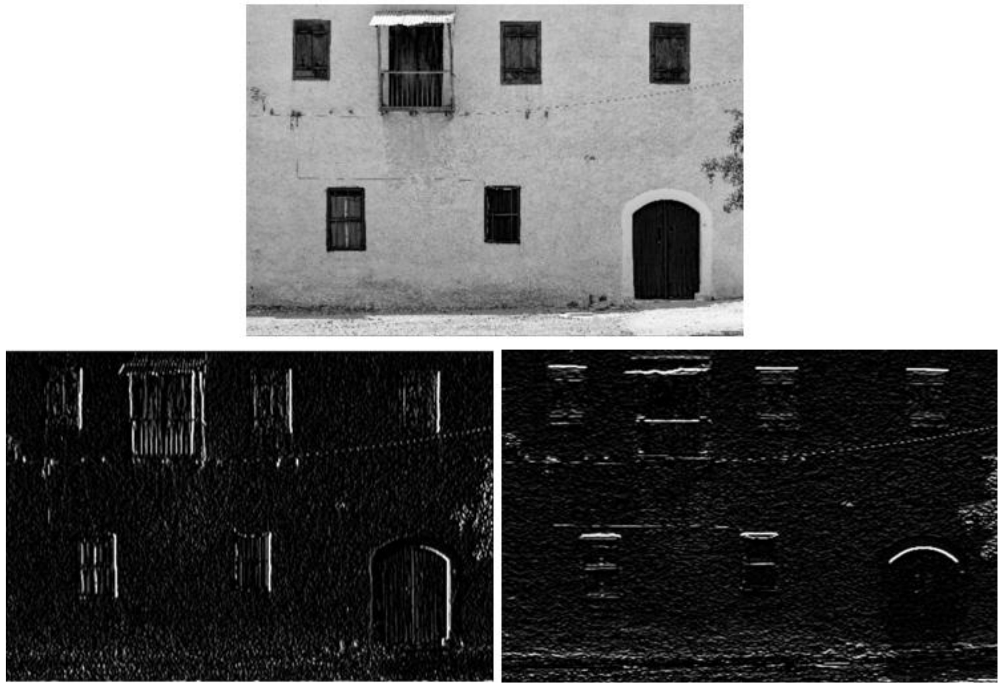


```

    if method in [cv2.TM_SQDIFF, cv2.TM_SQDIFF_NORMED]:
        top_left = min_loc
    else:
        top_left = max_loc

    bottom_right = (top_left[0] + width, top_left[1] + height)
    cv2.rectangle(img_copy,top_left, bottom_right, 255, 10)
    plt.subplot(121)
    plt.imshow(res)
    plt.title('Result of Template Matching')

    plt.subplot(122)
    plt.imshow(img_copy)
    plt.title('Detected Point')
    plt.suptitle(m)


    plt.show()
    print('\n')
    print('\n')
```



Worked out Examples

Question 7: Look at the grayscale image at the top of the collection of images below. Deduce what type of convolutional filter was used to get each of the lower images. Explain briefly and include the values of these filters. The filters have a shape of (3,3).



Answer:

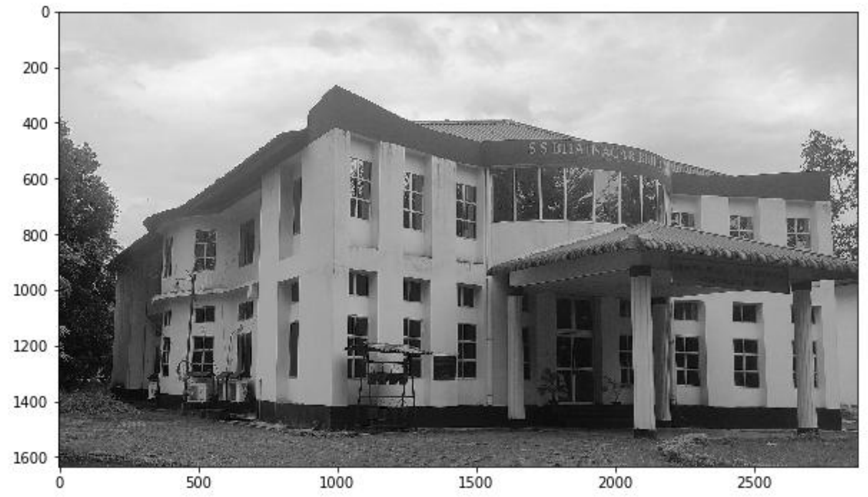
- Left image: Vertical edge detector. Filter: [[1,0,-1][1,0,-1][1,0,-1]]
- Right image: Horizontal edge detector. Filter: [[1,1,1][0,0,0][-1,-1,-1]]

Worked out Examples

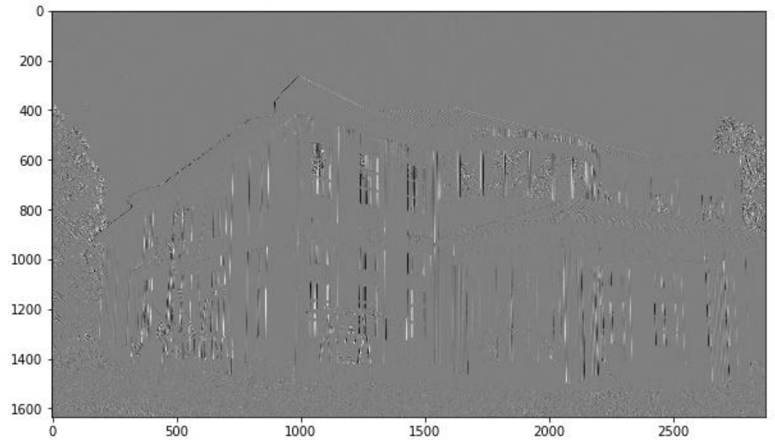
Question 8: Write codes in OpenCV to detect the vertical and horizontal lines?

Answer:

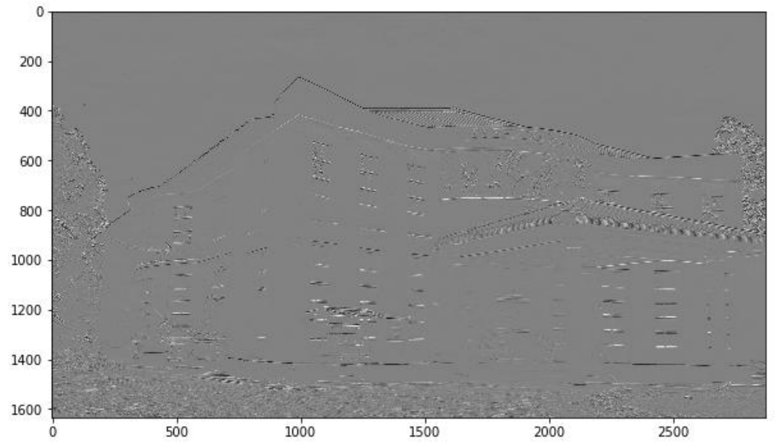
```
In [207]:
img =cv2.imread('D:/Ranjan_Dataset/Images/dept.jpg', 0)
#img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
display(img,cmap='gray')
```



```
In [210]:
sobelx = cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=5)
display(sobelx,cmap='gray')
```



```
In [211]:
sobely = cv2.Sobel(img, cv2.CV_64F, 0, 1, ksize=5)
display(sobely,cmap='gray')
```

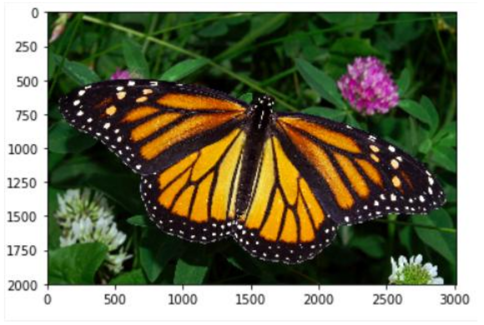


Worked out Examples

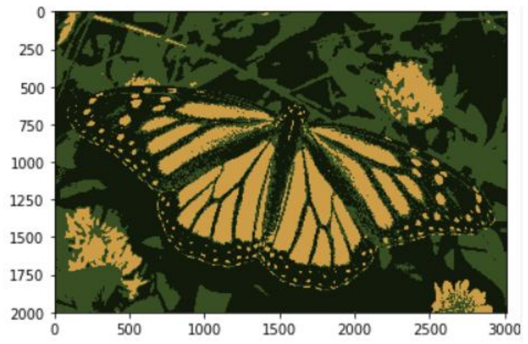
Question 9: Perform Image Segmentation using K Means Clustering?

Answer:

```
import numpy as np
import matplotlib.pyplot as plt
import cv2

%matplotlib inline

# Read in the image
image = cv2.imread('images/monarch.jpg')

# Change color to RGB (from BGR)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

plt.imshow(image)
```

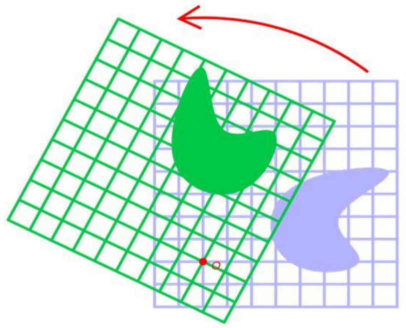


```
# Reshaping the image into a 2D array of pixels and 3 color values (RGB)
pixel_vals = image.reshape((-1,3))

# Convert to float type
pixel_vals = np.float32(pixel_vals)

criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 0.85)

# then perform k-means clustering wit h number of clusters defined as 3
#also random centres are initally chosed for k-means clustering
k = 3
retval, labels, centers = cv2.kmeans(pixel_vals, k, None, criteria, 10, cv2.KMEANS_RANDOM_CENTERS)

# convert data into 8-bit values
centers = np.uint8(centers)
segmented_data = centers[labels.flatten()]

# reshape data into the original image dimensions
segmented_image = segmented_data.reshape((image.shape))

plt.imshow(segmented_image)
```



Worked out Examples

Question 10: Suppose you have to rotate an image. Image rotation is nothing but multiplication of image by a specific matrix to get a new transformed image.



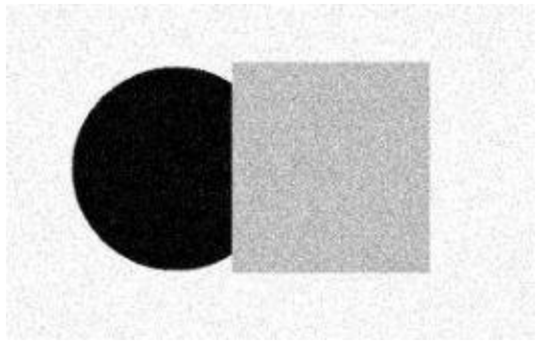
For simplicity, we consider one point in the image to rotate with co-ordinates as (1, 0) to a co-ordinate of (0, 1), what matrix would we have to multiply with?

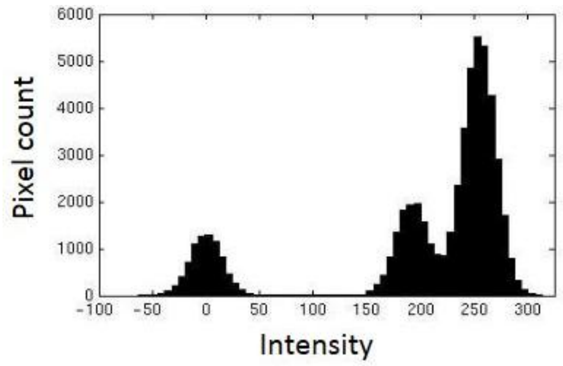
Answer: [[0, -1], [1, 0]]

The calculation of would be like this: [[0], [1]] = [[0, -1], [1, 0]] x [1, 0]

Worked out Examples

Question 11: Suppose we have an image given below.





Our task is to segment the objects in the image. A simple way to do this is to represent the image in terms of the intensity of pixels and the cluster them according to the values. On doing this, we got this type of structure. Suppose we choose k-means clustering to solve the problem, what would be the appropriate value of k from just a visual inspection of the intensity graph?

Answer: 3 because Three clusters will be formed; points in the circle, points in the square and the points excluding both of these objects.

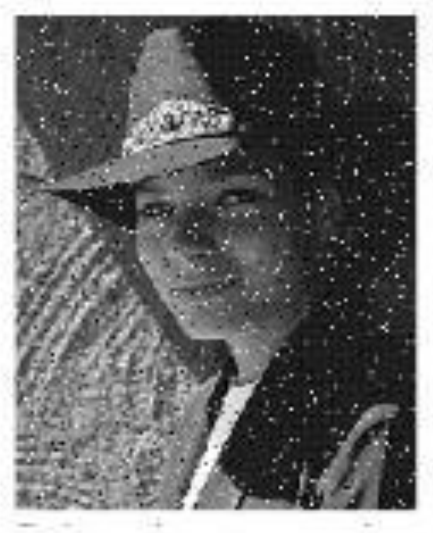
Worked out Examples

Question 12: Consider and image with width and height as 100×100. Each pixel in the image can have a color from Grayscale, i.e. values. How much space would this image require for storing in terms of bites?

Answer: 8x100x100 = 800 because 8 bits will be required to represent a number from 0-256.

Worked out Examples

Question 13: Suppose we have an image which is noisy. This type of noise in the image is called salt-and-pepper noise.



How will you remove the noise from the image?

Answer: Median filter technique

Worked out Examples

Question 14: If we convolve an image with the matrix given below, what would be the relation between the original and modified image?

0	0	0
0	0	1
0	0	0

Answer: The image will be shifted to the right by 1 pixel

Worked out Examples

Question 15: What are the data augmentation technique would you prefer for an object recognition problem?

Answer: Horizontal flipping, Rescaling, Zooming in the image

Worked out Examples

Question 16: Explain what is OpenCV? Write some applications of it?

Answer: OpenCV is a software released in June 2000 by the intel corporation. It mainly deals with real-time things such as videos, image, & machine learning. It is used in both the fields i.e., academics, & commercial. OpenCV is written in C++ & from version 3.4, OpenCV.js is a JavaScript. It can be run easily on any OS :- Windows, Linux, Mac.

Some most common application of OpenCV is:-

- Medical image analysis.
- Monitor tracking.
- gesture recognization.
- Automated inspection and surveillance.
- Movies – 3D structure from motion.
- 2D and 3D feature toolkits.
- Street view image stitching.
- Vehicle counting on highways along with their speeds.

Worked out Examples

Question 17: Given a dataset that consists of images of the Hoover tower, your task is to learn a classifier to detect the Hoover tower in new images. You implement PCA to reduce the dimensionality of your data, but find that your performance in detecting the Hoover tower significantly drops in comparison to your method on the original input data. A sample of your input training images are given in Fig. 2. Why is the performance suffering?

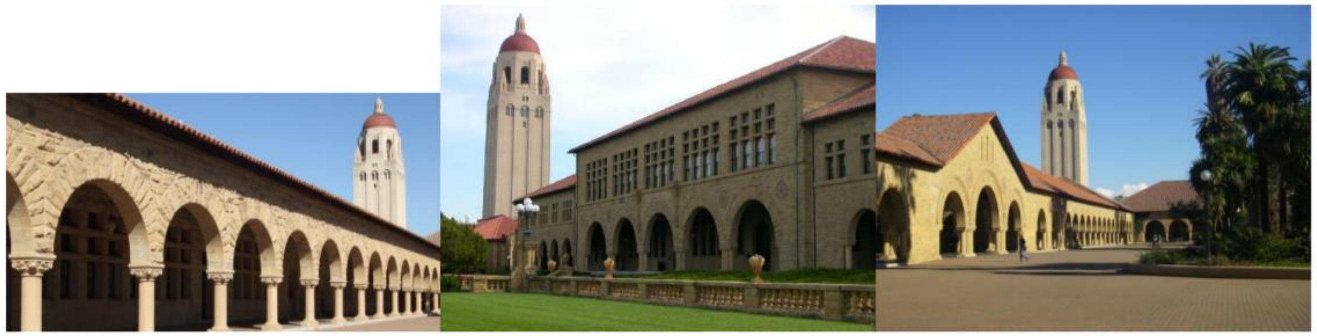


Figure 2: Example of input images

Answer: The Hoover tower in the images are not aligned, thus applying PCA to reduce the dimensionality here will not preserve the performance of the algorithms since we are trying to extract some signal out of the tower.

Worked out Examples

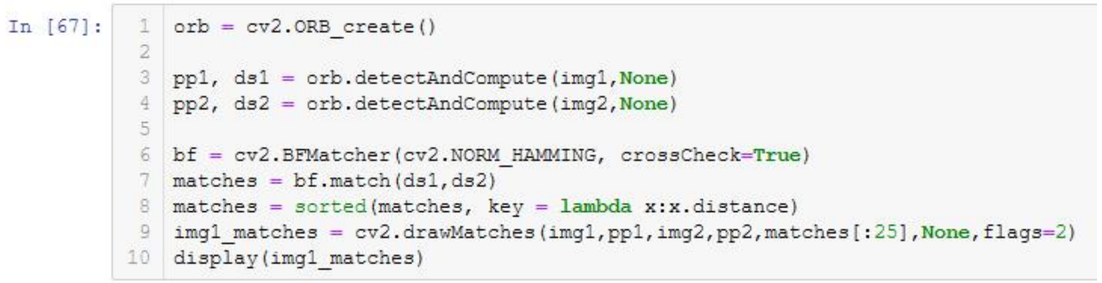
Question 18: You are using k-means clustering in color space to segment an image. However, you notice that although pixels of similar color are indeed clustered together into the same clusters, there are many discontiguous regions because these pixels are often not directly next to each other. Describe a method to overcome this problem in the k-means framework.

Answer: Concatenate the coordinates (x, y) with the color features as input to the k-means algorithm.

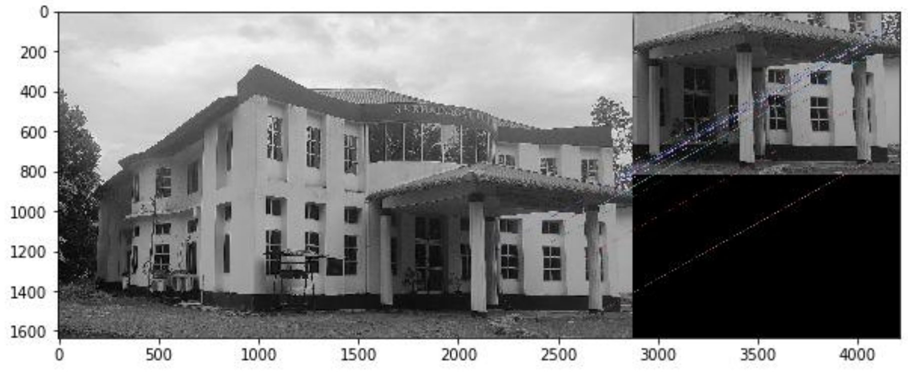
Worked out Examples

Question 19: Write code to match the features in the following two images?

Answer:



```
In [67]:
orb = cv2.ORB_create()

pp1, ds1 = orb.detectAndCompute(img1,None)
pp2, ds2 = orb.detectAndCompute(img2,None)

bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
matches = bf.match(ds1,ds2)
matches = sorted(matches, key = lambda x:x.distance)
img1_matches = cv2.drawMatches(img1,pp1,img2,pp2,matches[:25],None,flags=2)
display(img1_matches)
```

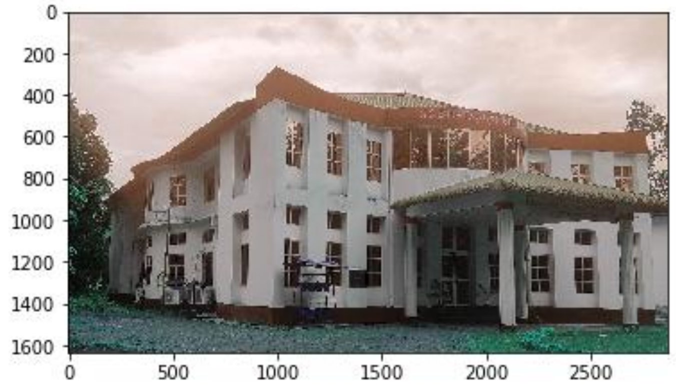


Worked out Examples

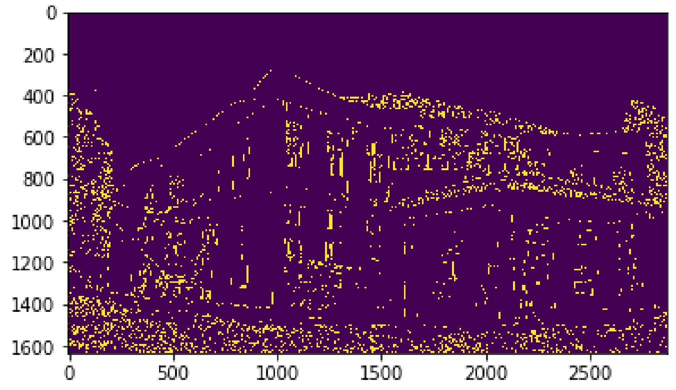
Question 20: Write codes to perform line detection in image?

Answer:

```
In [16]:  img = cv2.imread('D:/Ranjan_Dataset/Images/dept.jpg')
          plt.imshow(img)

Out[16]:  <matplotlib.image.AxesImage at 0x289dc1334c8>
```



```
In [17]:  edges = cv2.Canny(image=img, threshold1=127, threshold2=127)
          plt.imshow(edges)

Out[17]:  <matplotlib.image.AxesImage at 0x289dc1a14c8>
```



17.15 References

- Beyeler, Michael. OpenCV with Python blueprints. Packt Publishing Ltd, 2015.

➢ Discusses how to capture high-quality image data, detect and track objects, and process the actions of animals or humans Implement your learning in different areas of computer vision.

- Minichino, Joe, and Joseph Howse. Learning OpenCV 3 Computer Vision with Python. Packt Publishing Ltd, 2015.

➢ Provides a theoretical foundation of image processing and video analysis, and progress to the concepts of classification through machine learning, acquiring the technical know-how that will allow to create and use object detectors and classifiers, and even track objects in movies or video camera feeds..

- Villán, Alberto Fernández. Mastering OpenCV 4 with Python: a practical guide covering topics from image processing, augmented reality to deep learning with OpenCV 4 and Python 3.7. Packt Publishing Ltd, 2019.

➢ Provides a key concepts on computer vision. Also advanced applications of Python and OpenCV is being covered. Application of artificial intelligence and deep learning techniques using the popular Python libraries TensorFlow, and Keras are also part of this book.

- Gollapudi, Sunila. Learn computer vision using OpenCV. Apress, 2019.

➢ Build practical applications of computer vision using the OpenCV library with Python. Discusses different facets of computer vision such as image and object detection, tracking and motion analysis and their applications with examples.

Thank You