**LAB ASSIGNMENTS**

**Q. Write a program to add 2 greylevel images of same size and display the output.**

**import cv2**

**import numpy as np**

**from matplotlib import pyplot as plt**

**color = cv2.imread('img.jpg')**

**grayscale = cv2.imread('img.jpg',0)**

**bw = cv2.imread('img.jpg',0)**

**row,col = bw.shape**

**for x in range(row):**

**for y in range(col):**

**if bw[x][y]<126:**

**bw[x][y]=0**

**else:**

**bw[x][y]=255**

**plt.subplot(131),plt.imshow(color)**

**plt.title('Color Image'),plt.xticks([]),plt.yticks([])**

**plt.subplot(132),plt.imshow(grayscale,cmap='gray')**

**plt.title('Grayscale Image'),plt.xticks([]),plt.yticks([])**

**plt.subplot(133),plt.imshow(bw,cmap='gray')**

**plt.title('Black and White'),plt.xticks([]),plt.yticks([])**

**plt.show()**

**Q. Write a program to zoom and shrink a graylevel image at a desired level.**

**import cv2**

**import numpy as np**

**def zoom(img):**

**scale = int(input("Enter scale: "))**

**x,y = img.shape**

**new\_x,new\_y = x\*scale,y\*scale**

**new\_img = np.zeros(shape=(new\_x,new\_y))**

**for i in range(new\_x):**

**for j in range(new\_y):**

**ist = i//scale**

**jst = j//scale**

**new\_img[i][j] = img[ist][jst]**

**return new\_img**

**def shrink(img):**

**scale = int(input("Enter scale: "))**

**x, y = img.shape**

**new\_x, new\_y = x // scale, y // scale new \_img = np.zeros(shape=(new\_x, new\_y)) for i in range(new\_x):**

**for j in range(new\_y):**

**ist = i \*scale**

**jst = j \* scale**

**new\_img[i][j] = img[ist][jst]**

**return new\_img**

**img = cv2.imread('a.jpg',0)**

**zoom\_image = zoom(img)**

**shrink\_image = shrink(img)**

**#cv2.imshow('original',img)**

**cv2.imwrite('zoom.jpg',zoom\_image)**

**cv2.imwrite('shrink.jpg',shrink\_image)**

**Q. Write a program to obtain bitplane slicing in 8 bit planes.**

**import cv2**

**import numpy as np**

**def processing(img,n):**

**img2 = img\*n**

**cv2.imshow("img\_"+(str)(n),img2)**

**img1 = cv2.imread("img1.jpg",0)**

**img1 = cv2.resize(img1,(900,900))**

**for i in range(900):**

**for j in range(900):**

**st = img1[i][j]**

* **= (int)(st/32)**
* **= l\*32**

**u = l+32**

**mid = (int)(l/2 + u/2)**

**img1[i][j] = l**

**cv2.imshow("img",img1)**

**for i in range(1,9):**

**img = img1**

**processing(img,i)**

**cv2.imshow("img\_sp",img1)**

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**

**Q. Write a program to decrease the gray level from 256 to 128,64,32,16,8,4,2 of a monochrome image.**

**import cv2**

**import numpy as np**

**img = cv2.imread('dog.jpg',0)**

**st = []**

**for i in range(8):**

**new\_img = img**

**s = 1<<i**

**new\_img = new\_img & s**

**st.append(new\_img)**

**for i in range(len(st)):**

**cv2.imshow("layer"+str(i),st[i])**

**cv2.imshow("Original",img)**

**new\_img = img&15**

**final\_img = img-new\_img**

**result = cv2.equalizeHist(final\_img)**

**cv2.imshow("masked",new\_img)**

**cv2.imshow("final",final\_img)**

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**

**Q. Write a program to decompose an image into 8 1-bit planes and set 0 to 4 most significant bits then subtract the resultant image from the original image and show the output.**

**import cv2**

**import numpy as np**

**from matplotlib import pyplot as plt**

**img = cv2.imread('gray2.jpg',0)**

**row,col = img.shape**

**new = np.array(np.zeros((row,col)))**

**for x in range(row):**

**for y in range(col):**

**val=(img[x][y])//32**

**new[x][y]=((val)\*8+4)\*(8-val)**

**plt.subplot(121),plt.imshow(img,cmap='gray')**

**plt.title('Image'),plt.xticks([]),plt.yticks([])**

**plt.subplot(122),plt.imshow(new,cmap='gray')**

**plt.title('Transformed Image'),plt.xticks([]),plt.yticks([])**

**plt.show()**

**Q. Write a program to enhance an image using histogram equalisation and matching.**

**import cv2**

**import numpy as np**

**import matplotlib.pyplot as plt**

**def find\_freq(img):**

**val =[0]\*256**

**x,y = img.shape**

**for i in range(x):**

**for j in range(y):**

**v = img[i][j]**

**val[v] = val[v]+1**

**total = x\*y**

**cumu =[0]\*256**

**cumu[0] = val[0]/total**

**for i in range(1,256):**

**cumu[i] = (val[i]/total)+cumu[i-1] return (val,cumu)**

**def histogram\_equilization(img):**

**fre,cfre = find\_freq(img)**

**cdf = [0]\*256**

**for i in range(256):**

**cdf[i] = (int)(255\*cfre[i])**

**x,y = img.shape**

**n\_img = img.copy()**

**for i in range(x):**

**for j in range(y):**

**v = img[i][j]**

**f = fre[v]**

**inten = cdf[v]**

**n\_img[i][j] = inten**

**cv2.imshow("normal",img)**

**cv2.imshow("equalized",n\_img)**

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**

**def histogram\_specialisation(img1,img2): fre1,cum1= find\_freq(img1) fre2,cum2= find\_freq(img2) plt.hist(fre1,bins=100) plt.ylabel("Frequency") plt.show()**

**val1= [0]\*256**

**val2= [0]\*256**

**for i in range(256):**

**val1[i]= int(cum1[i]\*255)**

**for i in range(256):**

**val2[i]= int(cum2[i]\*255)**

**mapping= [0]\*256**

**for i in range(256):**

**val=val1[i]**

**dif = 256**

**index = 0**

**for j in range(256):**

* **= abs(val-val2[j]) if(h> dif):**

**mapping[i]=index break**

**else: dif=h**

**index=j**

**n\_img = img1.copy()**

**x,y = img1.shape**

**for i in range(x):**

**for j in range(y):**

**val=img1[i][j]**

**n\_img[i][j] = mapping[val]**

**cv2.imshow("targer image",img2)**

**cv2.imshow("specialised image",n\_img)**

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**

**img = cv2.imread("/home/shahrukh/Desktop/DP/list\_one/fast\_f.jpg",0)**

**img2 = cv2.imread("/home/shahurkh/Desktop/DP/house.jpg",0)**

**histogram\_equilization(img)**

**histogram\_specialisation(img2,img)**

1. **Write a program to obtain one-dimensional discrete Fourier transform of a given one-dimensional vector consisting of integer numbers generated randomly.**

**from math import e, pi**

**PI = pi**

**inputsignal = list(map(int, input("Array:").split()))**

**print(inputsignal)**

**M = len(inputsignal)**

**m = 0**

**Ftransformed = [None] \* M**

**for m in range(M):**

**sum = 0**

**for i in range(len(inputsignal)):**

**sum += inputsignal[i] \* e \*\* (-2 \* PI \* i \* m \* 1j / M)**

**Ftransformed[m] = sum / M**

**print(Ftransformed)**

**Q. Write a program to obtain two-dimensional discrete Fourier transform and its inverse of a grey level image of size 500\*500.**

**from math import e, pi**

**PI = pi**

**twodinput = []**

**M = int(input("give the no of rows\n"))**

**for i in range(M):**

**twodinput.append(list(map(int, input("Array[" + str(i) +**

**"]").split())))**

* **= len(twodinput[0]) x, y = 0, 0**

**twodftransform = []**

**for i in range(M):**

**twodftransform.append([])**

**for j in range(N):**

**twodftransform[-1].append(0)**

**for x in range(M):**

**for y in range(N):**

**sum = 0**

**m, n = 0, 0**

**for m in range(M):**

**for n in range(N):**

**sum += twodinput[m][n] \* e \*\* (-2j \* pi \* (((m \* x) / M)**

**+ ((n \* y) / N)))**

**twodftransform[x][y] = sum / (M \* N)**

**for row in twodftransform:**

**print(row)**

**Q. Write a program to apply frequency domain low pass and high pass filters to a gray level image of a size 500\*500 and display the filtered images.**

**from math import e, pi**

**import copy, cv2**

**PI = pi**

**twodinput = cv2.imread("img1.jpg", 0)**

**twodinput = cv2.resize(twodinput, (500, 500))**

**M = len(twodinput)**

* **= len(twodinput[0]) x, y = 0, 0**

**twodftransform = []**

**for i in range(M):**

**twodftransform.append([])**

**for j in range(N):**

**twodftransform[-1].append(0)**

**for x in range(M):**

**for y in range(N):**

**sum = 0**

**m, n = 0, 0**

**for m in range(M):**

**for n in range(N):**

**sum += twodinput[m][n] \* e \*\* (-2j \* pi \* (((m \* x) / M)**

**+ ((n \* y) / N)))**

**twodftransform[x][y] = sum / (M \* N)**

**print(x)**

**realtf = twodinput[:]**

**for i in range(M):**

**for j in range(N):**

**realtf[i][j] = twodftransform[i][j].real**

**inversetf = copy.deepcopy(twodinput)**

**for x in range(M):**

**for y in range(N):**

**sum = 0**

**m, n = 0, 0**

**for m in range(M):**

**for n in range(N):**

**sum += realtf[m][n] \* e \*\* (2j \* pi \* (((m \* x) / M) +**

**((n \* y) / N)))**

**inversetf[x][y] = sum.real**

**cv2.imshow("input", twodinput)**

**cv2.imshow("fourier", realtf)**

**cv2.imshow("ff", inversetf)**

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**

**Q. Write a program to apply frequency domain low pass and high pass filters to a gray level image . Also display the transfer functions as an image and obtain the histogram of each of these filtered images. Also find the magnitude spectrum.**

**import cv2**

**import numpy as np**

**from matplotlib import pyplot as plt**

**img = cv2.imread('messi.jpg',0)**

**f = np.fft.fft2(img)**

**fshift = np.fft.fftshift(f)**

**#print(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()**

**#High Pass Filter**

**rows, cols = img.shape**

**crow,ccol = int(rows/2) , int(cols/2)**

**for i in range(crow-30, crow +30):**

**for j in range(ccol - 30, ccol +30):**

**fshift[i][j] = 0 + 0j**

**f\_ishift = np.fft.ifftshift(fshift)**

**img\_back = np.fft.ifft2(f\_ishift)**

**img\_back = np.abs(img\_back)**

**plt.subplot(131),plt.imshow(img, cmap = 'gray')**

**plt.title('Input Image'), plt.xticks([]), plt.yticks([])**

**plt.subplot(132),plt.imshow(img\_back, cmap = 'gray')**

**plt.title('Image after HPF'), plt.xticks([]), plt.yticks([])**

**plt.subplot(133),plt.imshow(img\_back)**

**plt.title('Result in JET'), plt.xticks([]), plt.yticks([])**

**plt.show()**

**#Low pass Filter**

**fshift = np.fft.fftshift(f)**

**rows, cols = img.shape**

**crow,ccol = int(rows/2) , int(cols/2)**

* **create a mask first, center square is 1, remaining all zeros mask = np.zeros((rows,cols),np.uint8)**

**for i in range(crow-30, crow +30): for j in range(ccol - 30, ccol +30): mask[i][j] = 1**

**#mask[crow-30:crow+30, ccol-30:ccol+30] = 1**

* **apply mask and inverse DFT fshift = fshift\*mask**

**f\_ishift = np.fft.ifftshift(fshift) img\_back = np.fft.ifft2(f\_ishift) img\_back = np.abs(img\_back) plt.subplot(121),plt.imshow(img, cmap = 'gray') plt.title('Input Image'), plt.xticks([]), plt.yticks([]) plt.subplot(122),plt.imshow(img\_back, cmap = 'gray') plt.title('Image after LPF'), plt.xticks([]), plt.yticks([]) plt.show()**

**Q. Write a program to apply alpha trimmed min filter and adaptive median filter and then display the output with corresponding histogram.**

**import cv2**

**import matplotlib.pyplot as plt**

**import numpy as np**

**import random**

**def sp\_noise(image,prob):**

**output=np.zeros(image.shape,np.uint8)**

**thres=1-prob**

**for i in range(image.shape[0]):**

**for j in range(image.shape[1]):**

**rdn=random.random()**

**if rdn <prob:**

**output[i][j]=0**

**elif rdn>thres:**

**output[i][j]=255**

**else:**

**output[i][j]=image[i][j]**

**return output**

**img=cv2.imread("messi.jpg",0)**

**noise\_img=sp\_noise(img,0.05)**

**plt.subplot(131)**

**plt.imshow(img,cmap='gray')**

**plt.title('original')**

**plt.xticks([]),plt.yticks([])**

**plt.subplot(132)**

**plt.imshow(noise\_img,cmap='gray')**

**plt.title('noisy image')**

**plt.xticks([]),plt.yticks([])**

**plt.subplot(133)**

**plt.hist(noise\_img[100:200,100:200].ravel(),255,[0,255])**

**plt.title('histogram')**

**plt.show()**

**cv2.waitKey(0)**

**Q. Write a program to estimate noise added in image by analysing histogram.**

**import cv2**

**import numpy as np**

**import matplotlib.pyplot as plt**

**img = cv2.imread('cat.jpg',0)**

**row, col = img.shape**

**gauss = np.random.normal(10,10,(row,col))**

**rayleigh = np.random.rayleigh(50,(row,col))**

**gamma = np.random.gamma(10,10,(row,col))**

**exponential = np.random.exponential(50,(row,col))**

**uniform = np.random.uniform(10,50,(row,col))**

**noisy = img + gauss**

**smooth\_part = noisy[1:100,189:270]**

**plt.subplot(221),plt.imshow(smooth\_part,cmap = 'gray')**

**plt.title('Smooth Part'), plt.xticks([]), plt.yticks([])**

**plt.subplot(222),plt.imshow(noisy,cmap = 'gray')**

**plt.title('Noisy Image'), plt.xticks([]), plt.yticks([])**

**plt.subplot(223),plt.hist(smooth\_part.ravel(),256,[0,256])**

**plt.title('Smooth Part Histogram'), plt.xticks([]), plt.yticks([])**

**plt.subplot(224),plt.hist(noisy.ravel(),256,[0,256])**

**plt.title('Noisy Image Histogram'), plt.xticks([]), plt.yticks([])**

**plt.show()**

**Q. Write a program to apply a sequence of following filtering operation on a noisy finger print image with a structuring element of dimension 3\*3 whose elements are all 1.**

**Erosion**

**Opening of eroded image**

**Dilation of result obtained in (b)**

**Closing of the result obtained by (b)**

**import cv2**

**import numpy as np**

**import copy**

**def threshold(image):**

**for i in range(image.shape[0]):**

**for j in range(image.shape[1]):**

**if image[i][j]>126:**

**image[i][j]=255**

**else:**

**image[i][j]=0**

**return image**

**def erode(image):**

**movements=[[0,0],[1,0],[0,1],[1,1],[-1,0],[0,-1],[-1,-1],[-1,1],[1,-1]]**

**nimage=copy.deepcopy(image)**

**rows=image.shape[0]**

**cols=image.shape[1]**

**for i in range(rows):**

**for j in range(cols):**

**flag=1**

**for k in range(8):**

**imagex=i+movements[k][0]**

**imagey=j+movements[k][1]**

**if (imagey<cols and imagey>=0) and (imagex<rows and**

**imagex>=0):**

**if image[imagex][imagey]==0:**

**flag=0**

**break**

**nimage[i][j]=flag\*255**

**return nimage**

**def dilate(image):**

**movements = [[0, 0], [1, 0], [0, 1], [1, 1], [-1, 0], [0, -1], [-1, - 1], [-1, 1], [1, -1]]**

**nimage = copy.deepcopy(image)**

**rows = image.shape[0]**

**cols = image.shape[1]**

**for i in range(rows):**

**for j in range(cols):**

**flag = 0**

**for k in range(8):**

**imagex = i + movements[k][0]**

**imagey = j + movements[k][1]**

**if (imagey < cols and imagey >= 0) and (imagex < rows and**

**imagex >= 0):**

**if image[imagex][imagey] == 255:**

**flag = 1**

**break**

**nimage[i][j] = flag \* 255**

**return nimage**

**def bounds(dilated, eroded):**

**result = dilated[:]**

**rows=dilated.shape[0]**

**cols=dilated.shape[1]**

**for i in range(rows):**

**for j in range(cols):**

**result[i][j] = max(0, dilated[i][j] - eroded[i][j])**

**return result**

**img=cv2.imread("img3.jpg",0)**

**img=threshold(img)**

**cv2.imshow("original",img)**

**#**

**newimage=erode(img)**

**cv2.imshow("eroded",newimage)**

* **cv2.imshow('cv2Erosion', img\_erosion)**
* **cv2.waitKey(0)**
* **cv2.destroyAllWindows()**

**dilimage=dilate(img)**

**cv2.imshow("dilated",dilimage)**

**# cv2.imshow('cv2dilation', img\_dilation)**

**openingImage=dilate(erode(img))**

**cv2.imshow("opening",openingImage)**

**er\_image=erode(img)**

**dil\_image=dilate(img)**

**boundaryimage=bounds(img,er\_image)**

**cv2.imshow("boundary",boundaryimage)**

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**

**Q. Write a program to fill regions of image using appropriate structuring element.**

**import cv2**

**from scipy import ndimage**

**import numpy as np**

**import matplotlib.pyplot as plt**

**def threshold(image):**

**for i in range(image.shape[0]):**

**for j in range(image.shape[1]):**

**if image[i][j]>126:**

**image[i][j]=255**

**else:**

**image[i][j]=0**

**return image**

**def dilate(image):**

**movements = [[0, 0], [1, 0], [0, 1], [1, 1], [-1, 0], [0, -1], [-1, - 1], [-1, 1], [1, -1]]**

**nimage = image[:]**

**rows = image.shape[0]**

**cols = image.shape[1]**

**for i in range(rows):**

**for j in range(cols):**

**flag = 0**

**for k in range(8):**

**imagex = i + movements[k][0]**

**imagey = j + movements[k][1]**

**if (imagey < cols and imagey >= 0) and (imagex < rows and**

**imagex >= 0):**

**if image[imagex][imagey] == 255:**

**flag = 1**

**break**

**nimage[i][j] = flag \* 255**

**return nimage**

**img=cv2.imread("img3.jpg",0)**

**img=threshold(img)**

**imagem = cv2.bitwise\_not(img)**

* **plt.imread("img3.jpg",cmap="grey") plt.imshow(img,cmap='gray') plt.show()**

**hole\_coors=[[33,27]**

**[101,22],**

**[202,20],**

**[50,88]**

**[97,110],**

**[140,82],**

**[256,76],**

**[97,112],**

**[56,162],**

**[130,171],**

**[230,130],**

**[211,260],**

**[29,248],**

**[129,253],**

**[220,260]]**

**hole=img[:]**

**for i in range(hole.shape[0]):**

**for j in range(hole.shape[1]):**

**hole[i][j]=0**

**if [i,j] in hole\_coors:**

**hole[i][j]=255**

* **cv2.imshow("hole",imagem)**
* **cv2.waitKey(0)**
* **cv2.destroyAllWindows() acomp = cv2.bitwise\_not(img)**

**holenew=[[]]**

**while 1 :**

**holenew=dilate(hole)**

**holenew=cv2.bitwise\_and(holenew,acomp)**

**if np.array\_equal(hole,holenew):**

**break**

**hole=holenew**

**filledimage=cv2.bitwise\_or(img,holenew)**

**cv2.imshow("filled",filledimage)**

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**

**Q. Write a program to extract the boundary of an object given in an image with the help of an appropriate structuring elements.**

**import cv2**

**import numpy as np**

**from scipy import ndimage**

**from math import sqrt**

**def magnitude(img1,img2,t):**

**newimage=img1[:]**

**for i in range(img1.shape[0]):**

**for j in range(img1.shape[1]):**

**newimage[i][j]=sqrt((img1[i][j]\*\*2+img2[i][j]\*\*2))**

**if newimage[i][j]<t:**

**newimage[i][j]=0**

**else:**

**newimage[i][j]=255**

**return newimage**

**image =cv2.imread("img7.jpg",0)**

* **print(image) avg=np.array([[1,1,1],**

**[1, 1, 1], [1, 1, 1]])**

* **avg=np.array(avg)**

**img1=ndimage.convolve(image,avg , mode="constant", cval=0)**

**img1=img1/9**

**cv2.imshow("avg",img1)**

**vertical\_kernel=np.array([[-1,0,1],**

**[-2,0,2],**

**[-1,0,1]])**

**horizontal\_kernel=np.array([[-1,-2,-1],**

**[0,0,0],**

**[1,2,1]])**

**img1=ndimage.convolve(img1, vertical\_kernel, mode="constant", cval=0)**

**img2=ndimage.convolve(img1, horizontal\_kernel, mode="constant", cval=0)**

**img4=magnitude(img1,img2,72)**

**cv2.imshow("edges",img4)**

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**

**Q. Write a program to detect points in image, detect vertical edges and horizontal edges. Find edges using suitable edge detection mask.**

**import cv2**

**import numpy as np**

**from scipy import ndimage**

**from math import sqrt**

**def threshold(img3,T):**

**for i in range(len(img3)):**

**for j in range(len(img3[0])):**

**if img3[i][j] > T:**

**img3[i][j] = 255**

**else:**

**img[i][j] = 0**

**return img3**

**def magnitude(img1,img2):**

**newimage=img1[:]**

**for i in range(img1.shape[0]):**

**for j in range(img.shape[1]):**

**newimage[i][j]=sqrt((img1[i][j]\*\*2+img2[i][j]\*\*2))**

**return newimage**

**img=cv2.imread("img (5).jpg",0)**

**# cv2.imshow("original",img)**

**vertical\_kernel=np.array([[-1,0,1],**

**[-2,0,2],**

**[-1,0,1]])**

**horizontal\_kernel=np.array([[-1,-2,-1],**

**[0,0,0],**

**[1,2,1]])**

**laplacian\_kernel=np.array([[-1,-1,-1],**

**[-1, 8, -1],**

**[-1, -1, -1]])**

**# img1=np.sum(np.multiply(img, vertical\_kernel))**

**img1=ndimage.convolve(img, vertical\_kernel, mode="constant", cval=0)**

**img2=ndimage.convolve(img, horizontal\_kernel, mode="constant", cval=0)**

**img3=ndimage.convolve(img,laplacian\_kernel,mode="constant",cval=0)**

**cv2.imshow("original",img)**

**cv2.imshow("vertical",img1)**

**cv2.imshow("horizontal",img2)**

* **img4=(img1\*img1+img2\*img2)\*\*0.5**

**img4=magnitude(img1,img2)**

* **cv2.imshow("edge",img4)**

**img3=threshold(img3,5)**

**cv2.imshow("point",img3)**

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**

**Q. Write a program to find alignment of a set of points using hough transform.**

**import cv2**

**import numpy as np**

**img = cv2.imread('house.jpeg')**

**gray = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)**

**ret, bw\_img = cv2. threshold(gray,127,255,cv2. THRESH\_BINARY)**

**edges = cv2.Canny(bw\_img,50,150,apertureSize = 3)**

**lines = cv2.HoughLines(edges,1,np.pi/180,200)**

**for rho,theta in lines[0]:**

**a = np.cos(theta)**

**b = np.sin(theta)**

**x0 = a\*rho**

**y0 = b\*rho**

**x1 = int(x0 + 1000\*(-b))**

**y1 = int(y0 + 1000\*(a))**

**x2 = int(x0 - 1000\*(-b))**

**y2 = int(y0 - 1000\*(a))**

**cv2.line(img,(x1,y1),(x2,y2),(0,0,255),2)**

**plt.imshow(img)**

**plt.title('HoughLines')**

**plt.show()**