**Practical 8: Image Processing and Computer Vision**

**Pre-requisite:** Before you start this practical, please ensure you have “Python 2.7” and “opencv-2.4.12” installed in your computer. You may follow the following steps for installation:

1. Double-click to extract “opencv-2.4.12”.
2. Go to “opencv/build/python/2.7/x86 folder.”
3. Copy cv2.pyd to C:/Python27/lib/site-packages.

**Instruction**

In this practical, student is required to perform the following image processing and computer vision functions. Image and video data will be provided for the purposes.

(A) Image processing: perform the following functions for Image processing:

1. Read image
2. Resize image
3. Convert to grayscale image
4. Perform Morphological operation
5. Image denoise
6. Edge detection
7. Corner detection

(B) Computer Vision: Track a car/person on the road using optical flow

1. Read a video
2. Perform background subtraction
3. Perform motion tracking

**Section A: Image Processing**

1. Create a new py file and type the following using the Python Editor to import necessary libraries. “cv2” is a well-known library (OpenCV) to perform image processing and video analysis.

import numpy as np

import cv2

1. “*imread*” is used to read an image in Python while “*imshow*” can be used to visualize an image output. The following lines read “Lenna.png” and visualize it in python.

img = cv2.imread('Lenna.png')

cv2.imshow('picOriLenna',img)

1. An image can be resized with the “*resize*” function. “fx” and “fy” indicate the scale that the image to be resized (Example: 0.5 means to downsize the image into ½ of the original size).

imgResize = cv2.resize(img,None,fx=0.5, fy=0.5)

cv2.imshow('picResized',imgResize)

1. An image can be converted to grayscale from color image with the following lines.

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

cv2.imshow('picGray',gray\_image)

1. Images may contain numerous imperfections. In particular, the binary regions produced by simple thresholding are distorted by noise and texture (example, gaps between the pixels). Morphological image processing pursues the goals of removing these imperfections by accounting for the form and structure of the image. Morphological operators often take a binary image and a structuring element as input and combine them using a set operator (intersection, union, inclusion, complement). They process objects in the input image based on characteristics of its shape, which are encoded in the structuring element. There are four common types of morphological operations which are; erosion, dilation, opening and closing. Following lines perform morphological operation with a Digit image.

# We use Digit image to perform morphological operation

imgDigit = cv2.imread('Digit3.png')

cv2.imshow('picOriDigit',imgDigit)

# Erosion

kernel = np.ones((3,3),np.uint8) #To define the structuring element

erosion = cv2.erode(imgDigit,kernel,iterations = 1)

cv2.imshow('picErosion',erosion)

# Dilation

kernel = np.ones((5,5),np.uint8)

dilation = cv2.dilate(imgDigit,kernel,iterations = 1)

cv2.imshow('picDilation',dilation)

# Opening

kernel = np.ones((3,3),np.uint8)

opening = cv2.morphologyEx(imgDigit, cv2.MORPH\_OPEN, kernel)

cv2.imshow('picOpening',opening)

# Closing

kernel = np.ones((5,5),np.uint8)

closing = cv2.morphologyEx(imgDigit, cv2.MORPH\_CLOSE, kernel)

cv2.imshow('picClosing',closing)

1. Image noise is random (not present in the object imaged) variation of brightness or color information in images. It is an unwanted signal that could be an obstacle in the later processes (e.g feature extraction) and it might affect the overall system performance. Thus, it is important to have noise removal in the image preprocessing step. Following lines demonstrates noise removal by using three different filters (Average, Gaussian, and Median) on camera man image with salt and pepper noise.

#Read and convert image into grayscale

imgNoise = cv2.imread("cameraman\_noise.jpg")

gray\_imgNoise = cv2.cvtColor(imgNoise, cv2.COLOR\_BGR2GRAY)

cv2.imshow('picOriNoise',gray\_imgNoise)

# Averaging filtering

blurAverage = cv2.blur(gray\_imgNoise,(5,5))

cv2.imshow('picBlur', blurAverage)

# Gaussian filtering

blurGauss = cv2.GaussianBlur(gray\_imgNoise,(5,5),0)

cv2.imshow('picBlurGauss',blurGauss)

# Median filtering

blurMedian = cv2.medianBlur(gray\_imgNoise,5)

cv2.imshow('picBlurMedian',blurMedian)

1. Edge and corner are very useful image features commonly used for feature extraction and perform recognition in an image. Following lines used canny operator to extract the edge and Harris corner detector to extract the corner of the Chessboard image.

filename = 'Chessboard.png'

img = cv2.imread(filename)

img = cv2.resize(img,None,fx=0.5, fy=0.5)

cv2.imshow('picOriChessboard',img)

# Edge detection using canny operator

edgesCanny = cv2.Canny(img,100,200)

cv2.imshow('picEdgeCanny',edgesCanny)

# Corner detection using Harris corner detector

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

gray = np.float32(gray)

dst = cv2.cornerHarris(gray,2,3,0.04)

dst = cv2.dilate(dst,None) #result is dilated for marking the corners, not important

img[dst>0.01\*dst.max()]=[0,0,255] # Threshold for an optimal value, it may vary depending on the image.

cv2.imshow('picCornerHarris', img)

1. It is important that in Python we need to hold the visualization by using “*cv2.waitKey(0)*” until the user press any key for further processing. User may close all windows to release the memory by using “*cv2.destroyAllWindows()*”.

cv2.waitKey(0)

cv2.destroyAllWindows()

**Section B: Computer Vision**

1. Beforehand, please make sure you have “opencv\_ffmpeg2412.dll” in your local directory together with your code and sample of videos.
2. Read video from file and loop it for visualization. “*ret*” is true only if the video is successfully loaded and each frame of the video will be assigned to the variable “*frame*” for each loop.

cap = cv2.VideoCapture('walk.avi')

while(True):

ret, frame = cap.read()

if ret == True:

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

cv2.imshow('frame',frame)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

else:

break

1. Perform background subtraction on the video.

cap = cv2.VideoCapture('walk.avi')

fgbg = cv2.BackgroundSubtractorMOG()

while(True):

ret, frame = cap.read()#

if ret == True:

fgmask = fgbg.apply(frame)

cv2.imshow('frame',fgmask)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

else:

break

1. Perform motion tracking by using optical flow on the video.

cap = cv2.VideoCapture("walk.avi")

ret, frame1 = cap.read()

prvs = cv2.cvtColor(frame1,cv2.COLOR\_BGR2GRAY)

hsv = np.zeros\_like(frame1)

hsv[...,1] = 255

while(True):

ret, frame2 = cap.read()

if ret == True:

next = cv2.cvtColor(frame2,cv2.COLOR\_BGR2GRAY)

flow = cv2.calcOpticalFlowFarneback(prvs,next, 0.5, 3, 15, 3, 5, 1.2, 0)

mag, ang = cv2.cartToPolar(flow[...,0], flow[...,1])

hsv[...,0] = ang\*180/np.pi/2

hsv[...,2] = cv2.normalize(mag,None,0,255,cv2.NORM\_MINMAX)

rgb = cv2.cvtColor(hsv,cv2.COLOR\_HSV2BGR)

cv2.imshow('frame2',rgb)

k = cv2.waitKey(1) & 0xff

if k == ord('q'):

break

prvs = next

else:

break

1. Similar to image processing, we need to hold the visualization by using “cv2.waitKey(0)” until the user press any key for further processing or release the video capture with “*cap.release()*”. User may close all windows to release the memory by using “cv2.destroyAllWindows()”.

cv2.waitKey(0)

cap.release()

cv2.destroyAllWindows()

**Exercise:**

1. Segment different elements from the map images (“map.png”) and visualize it in different windows such as lake, road, field, and housing area. Hint: you may refer to the following steps and distinguish each of them by using different range of RGB values.
   1. Read “map.png” in python.
   2. Insert the following code to extract the river. Understand the code and repeat the same process to extract road, field and housing area.

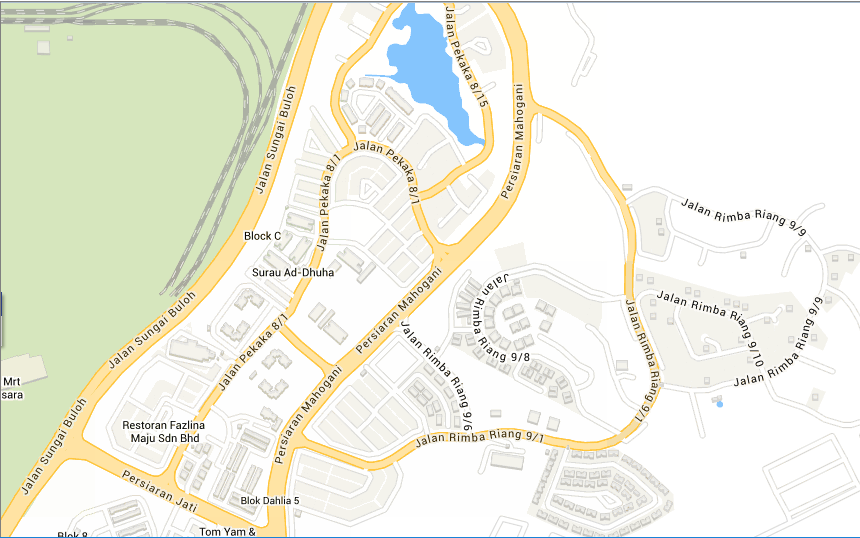
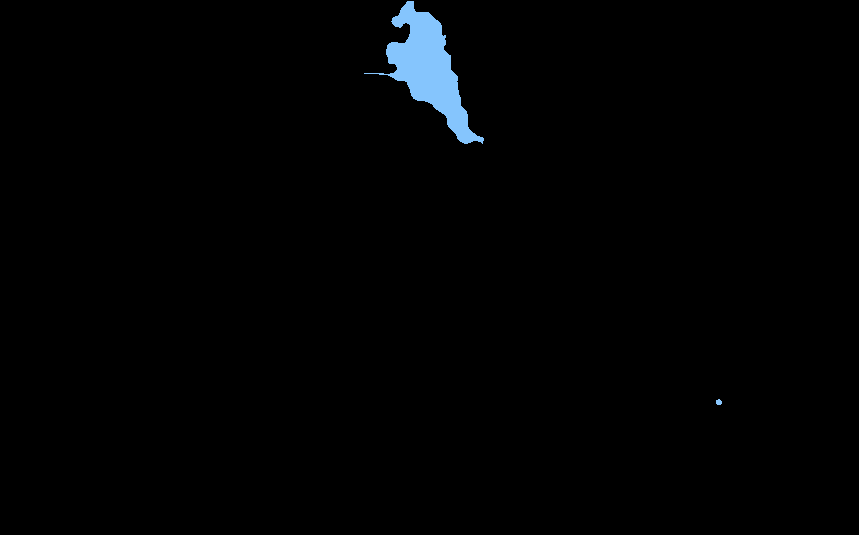
lower\_river = np.array([240,185,120]) #lower boundary of RGB value

upper\_river = np.array([260,205,150]) #upper boundary of RGB value

mask\_river = cv2.inRange(img, lower\_river, upper\_river)

river = cv2.bitwise\_and(img, img, mask = mask\_river)

cv2.imshow("pic\_river", river)

Original map Lake

Road Field



Housing area

Hint: You may check the RGB values for the map using the following website:

<http://imagecolorpicker.com/>

1. Implement an intrusion detection system which will continually loop the program and fire an alarm when Intrusion is True (you may use the sample video and declare an intrusion as “1” when intrusion is true).
2. Try this if you have a webcam. You may use “cap = cv2.VideoCapture(0)” to retrieve the live video from your webcam.

**References:**

<http://opencv.org/>

<http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_tutorials.html>