Kunal Dhaimade – CS532 Homework Assignment 3

Source Code:

```
# -*- coding: utf-8 -*-
"""
Created on Tue Oct 24 02:29:32 2017

@author: Kunal
"""

from PIL import Image
import numpy as np
import math as mt

'''To resize the image to replicate border pixels before convolution as per the filter_size'''
def resize_image(ip_im, filter_size):
    r, c = ip_im.shape
    filter_n = int((filter_size-1)/2)
    op_r = r+2*(filter_n)
    op_c = c+2*(filter_n)
    op_im = np.zeros((op_r,op_c))
    for i in range(r):
        for j in range(c):
            op_im[i+filter_n][j+filter_n] = ip_im[i][j]
    for i in range(filter_n):
        for j in range(filter_n):
            op_im[i][j] = op_im[filter_n][filter_n]
    for i in range(filter_n):
        for j in range(op_c-filter_n, op_c):
            op_im[i][j] = op_im[filter_n][op_c-filter_n-1]
    for i in range(op_r-filter_n, op_r):
        for j in range(filter_n):
            op_im[i][j] = op_im[op_r-filter_n-1][filter_n]
    for i in range(op_r-filter_n, op_r):
        for j in range(op_c-filter_n, op_c):
            op_im[i][j] = op_im[op_r-filter_n-1][op_c-filter_n-1]
    for i in range(filter_n):
        for j in range(filter_n, op_c-filter_n):
            op_im[i][j] = op_im[filter_n][j]
    for i in range(op_r-filter_n, op_r):
        for j in range(filter_n, op_c-filter_n):
            op_im[i][j] = op_im[op_r-filter_n-1][j]
    for i in range(filter_n, op_r-filter_n):
        for j in range(filter_n):
            op_im[i][j] = op_im[i][filter_n]
    for i in range(filter_n, op_r-filter_n):
        for j in range(op_c-filter_n, op_c):
            op_im[i][j] = op_im[i][op_c-filter_n-1]
```

```
    return op_im

'''To perform convolution of ip with filter'''
def convolution(ip,filter):
    filter_size = int(mt.sqrt(filter.size))
    filter_n = int((filter_size-1)/2)
    ip_r, ip_c = ip.shape
    r = ip_r - 2*filter_n
    c = ip_c - 2*filter_n
    op_im = np.zeros((r, c))
    for i in range(r):
        for j in range(c):
            for k in range(filter_size):
                for l in range(filter_size):
                    op_im[i][j] = op_im[i][j] + (filter[k][l] * ip[i+k][j+l])
            if(op_im[i][j] < 0):
                op_im[i][j] = 0
            else:
                op_im[i][j] = int(op_im[i][j])
    return op_im

'''To create the gaussian filter'''
def gauss_filter(og_im, size, sigma):
    size = int(size)
    sigma = float(sigma)
    #og_im = np.array(im)
    filter = np.zeros((size,size))
    filter_n = int((size-1)/2)
    y, x = np.ogrid[float(-filter_n):float(filter_n+1),float(-filter_n):float(filter_n+1)]
    sum = 0
    for i in range(size):
        for j in range(size):
            e = mt.exp((-((x[0][j]**2)+(y[i][0]**2))/(2*(sigma**2))))
            filter[i][j] = e*(1/(2*mt.pi*(sigma**2)))
            sum = sum + filter[i][j]
    for i in range(size):
        for j in range(size):
            filter[i][j] = filter[i][j]/sum
    #r, c = og_im.shape
    m_im = resize_image(og_im, size)
    m_r, m_c = m_im.shape
    op_im = convolution(m_im, filter)
    return op_im

'''To obtain the X-derivative'''
def grad_x(ip_im):
    filter_x = [[-1,0,+1], [-1,0,+1], [-1,0,+1]]
    filter_x = np.array(filter_x)
```

```
    m_im = resize_image(ip_im, 3)
    op_im = convolution(m_im, filter_x)
    return op_im

'''To obtain the Y-derivative'''
def grad_y(ip_im):
    filter_y = [[-1,-1,-1], [0,0,0], [+1,+1,+1]]
    filter_y = np.array(filter_y)
    m_im = resize_image(ip_im, 3)
    op_im = convolution(m_im, filter_y)
    return op_im

'''To calculate the moment matrix for each pixel'''
def calc_moment_matrix(ip_im):
    im_ar = np.asarray(ip_im)
    im_x_ar = grad_x(im_ar)
    im_y_ar = grad_y(im_ar)
    r, c = im_x_ar.shape
    im_xx_ar = np.zeros((r, c))
    im_yy_ar = np.zeros((r, c))
    im_xy_ar = np.zeros((r, c))
    for i in range(r):
        for j in range(c):
            im_xx_ar[i][j] = im_x_ar[i][j]*im_x_ar[i][j]
            im_yy_ar[i][j] = im_y_ar[i][j]*im_y_ar[i][j]
            im_xy_ar[i][j] = im_x_ar[i][j]*im_y_ar[i][j]
    im_xxg_ar = gauss_filter(im_xx_ar, 5, 1)
    im_yyg_ar = gauss_filter(im_yy_ar, 5, 1)
    im_xyg_ar = gauss_filter(im_xy_ar, 5, 1)
    mm = [[[] for i in range(c)] for j in range(r)]
    for i in range(r):
        for j in range(c):
            mm[i][j] = [im_xxg_ar[i][j], im_xyg_ar[i][j], im_yyg_ar[i][j]]
    return mm

'''To calculate the response value for each pixel'''
def corner_response(ip_im, mm, k, t):
    im_ar = np.asarray(ip_im)
    r, c = im_ar.shape
    res = np.zeros((r, c))
    no_c = 0
    corners = []
    for i in range(r):
        for j in range(c):
            res[i][j] = ((mm[i][j][0]*mm[i][j][2]) - (mm[i][j][1]**2)) - (k*((mm[i][j][0]+mm[i][j][2])**2))
    res = resize_image(res, 3)
    rn, cn = res.shape
    for i in range(1, rn-1):
```

```
      for j in range(1, cn-1):
         if(res[i][j] > t):
            if(res[i][j] == max(res[i-1][j-1], res[i-1][j], res[i-1][j+1], res[i][j-1], res[i][j], res[i][j+1], res[i+1][j-1],
res[i+1][j], res[i+1][j+1])):
               corners.append([i-1, j-1, res[i-1][j-1]])
               no_c = no_c + 1
   return corners

'''To calculate the SAD distance to obtain the various correspondences'''
def calc_sad(im_tr, im_tl, trc, tlc):
   im_tr = np.asarray(im_tr)
   im_tl = np.asarray(im_tl)
   r, c = im_tr.shape
   dist = []
   for left in tlc:
      for right in trc:
         s = 0
         for i in range(-1, 2):
            li = i + left[0]
            ri = i + right[0]
            for j in range(-1, 2):
               lj = j + left[1]
               rj = j + right[1]
               if(0<=li<r and 0<=lj<c and 0<=ri<r and 0<=rj<c):
                  s = s + abs(int(im_tl[li][lj]) - int(im_tr[ri][rj]))
         dist.append([s, [left[0], left[1]], [right[0], right[1]]])
   dist.sort()
   return dist

'''To report the accuracies for the various correspondences'''
def report(dist, disp):
   disp = np.asarray(disp)
   no_c = 0
   no_ic = 0
   pm = 5
   p = (pm/100)*len(dist)
   for corr in dist:
      i = corr[1][0]
      j = corr[1][1]
      og_dis = disp[i][j]
      c_dis = abs(corr[1][1] - corr[2][1])
      if(abs(og_dis-c_dis) < 2):
         no_c = no_c + 1
      else:
         no_ic = no_ic + 1
      total = no_c + no_ic
      p = int((pm/100)*len(dist))
      if(p == total):
```

```
        cor_p = (no_c/total)*100
        print("Percent: "+str(pm)+"; Correct: "+str(no_c)+"; Total: "+str(total)+"; Accuracy: "+str(cor_p))
        pm = pm + 5

'''The main function'''
def main():
    og_im_tr = Image.open('teddy/teddyR.pgm')
    og_im_tl = Image.open('teddy/teddyL.pgm')
    og_im_disp = Image.open('teddy/disp2.pgm')
    tr_mm = calc_moment_matrix(og_im_tr)
    tr_co = corner_response(og_im_tr, tr_mm, 0.05, 12000000)
    tl_mm = calc_moment_matrix(og_im_tl)
    tl_co = corner_response(og_im_tl, tl_mm, 0.05, 12000000)
    dist = calc_sad(og_im_tr, og_im_tl, tr_co, tl_co)
    report(dist, og_im_disp)

main()
```

Kunal Dhaimade – CS532 Homework Assignment 3

Notes:

I have written the code in Python. I have created functions for obtaining the first order derivative using the given kernel, for performing Gaussian blur, for creating the moment matrix, for obtaining the corners using the corner response function, computing SAD distances to obtain correspondences, and to validate the correspondences and report the accuracy as required. I compute the gradients, blur them, obtain the moment matrices, and obtain corners for both the right and the left images. After this, I compute the SAD distances for every corner (320) in left image against every corner (363) in right, thus obtaining 320 * 363 correspondences. I then use the ground truth values in the given disparity map to check which of the obtained correspondences are correct and which are incorrect.

Percent Correspondences	Correct Correspondences	Total Correspondences	Accuracy Percentage
5	77	*5808*	*1.3257575757575757*
10	137	*11616*	*1.1794077134986225*
15	187	*17424*	*1.073232323232323*
20	255	*23232*	*1.0976239669421488*
25	306	*29040*	*1.0537190082644627*
30	358	*34848*	*1.0273186409550046*
35	404	*40656*	*0.9937032664305392*
40	453	*46464*	*0.974948347107438*
45	504	*52272*	*0.9641873278236914*
50	553	*58080*	*0.9521349862258953*
55	602	*63888*	*0.9422739794640622*
60	645	*69696*	*0.9254476584022039*
65	677	*75504*	*0.8966412375503284*
70	730	*81312*	*0.8977764659582841*
75	763	*87120*	*0.8758034894398531*
80	812	*92928*	*0.8737947658402203*
85	859	*98736*	*0.8699967590341923*
90	918	*104544*	*0.878099173553719*
95	970	*110352*	*0.8790053646512976*
100	1046	*116160*	*0.9004820936639119*