EECS 504 Foundations of Computer Vision: HW2

Term: Fall 2018
Instructor: Jason J. Corso, EECS, University of Michigan
Due Date: 10/22 23:59 Eastern Time

Constraints: This assignment may be discussed with other students in the course but must be written independently. Programming assignments should be written in Python using the open-source library ETA. Over-the-shoulder Python coding is strictly prohibited. **Web/Google-searching for background material is permitted. However, everything you need to solve these problems is presented in the course notes and background materials, which have been provided already.**

Goals: Test mathematical basics and deepen the understanding of images as functions.

Data: You need to download `hw2.zip` to complete this assignment. All paths in the assignment assume the data is off the local directory.

Problem 1 (30): Local Image Features and Image Stitching
We motivated and derived an initial local image feature point detector based on an eigen decomposition of the structure tensor. We also demonstrated an application example of image stitching. In this question, you will explore these topics further. You can use `pipelines/image_stitching.json` for this question

(a) (10) Implement the local structure tensor construction and eigen decomposition as described in class. This method is called Harris detector, but implement the simplified corner response measure based on the minimum eigenvalue of the structure tensor, as discussed in class. You should fill in your code in `_get_harris_corner()` of `modules/harris.py`. Read comments in code for further instructions on arguments and return values.
You should run the pipeline on `checkerboard.png`. This will execute your Harris function and will run a non-maximal suppression routine and extract corner locations from response image.
Include both the images:- (a) checkerboard response image (directly from Harris) and (b) actual corner detections, and the code verbatim in the pdf report.

You can run the the pipeline using the following command:
`eta build -r requests/only_harris_request.json --run-now`

(b) (5) Run the pipeline using `concentric_circles.png` as your input. Include the corner response image in the write-up. Also please discuss the response image explaining why it looks like the way it does. Specifically describe why there are so many responses, when there are no "corners" in the image, and why, even though, there are many responses, no full ring has a high corner response over the entire ring.

(c) (12) **Image Stitching**. This question will show you an end-to-end use of these computer vision tools by stitching together the two images below `red1.png` and `red2.png` The code for doing this has three basic pieces: reduction (extraction of corners and representation with HOG features), matching (finding the best `K` correspondences across the images), and estimation (computation of the homography that aligns the two images).

Figure 1: red1.png

Figure 2: red2.png

To make this possible, the code for extracting HOG features and matching correspondences is provided to you. You have to fill in the functions `_get_homography()` (to compute the homography matrix) and `_overlap()` in file `modules/image_stitching.py`. Perform the stitching from the perspective of `red1.png`.

You can run the the pipeline using the following command:
`eta build -r requests/image_stitching_request.json --run-now`

(d) (3) Repeat the process with different number of correspondences (4,10 and 20)
Explain (1) why 4 correspondences is worse than 10; (2) why 20 correspondences, which we may expect to be better, completely fails. Include the output images and your explanations in the report.

Problem 2 (15): In the last assignment, you have implemented kernels for x and y direction for finding the derivative and gradient. Think of a situation when you have to do the same in an arbitrary direction. Will it be feasible to construct specific kernels for all the possible directions?
The **steerable filters** are a class of filters in which a filter of arbitrary orientation is synthesized as a linear combination of a set of "basis filters."

(a) (5) You are given an image `lines.png`. You have implemented the Sobel horizontal and vertical kernels in HW1. Use these kernels to detect the lines in the image that do not lie along 'x' or 'y' axis and attach the output image in your writeup. Your output should consist of **only** the lines that are not horizontal or vertical. You can use `pipelines/steerable_filter.json` and `modules/steerable_filter.py` to do this task

You can run the the pipeline using the following command:
`eta build -r requests/steerable_filter_request.json --run-now`

(b) (7) Is the first-order derivative of 2-D Gaussian filter steerable? Why? Prove your answer. If yes, what are its basis functions?

(c) (3) Is the second-order derivative of 2-D Gaussian filter steerable? If yes, give the dimension of its basis. Can this be extended to n^{th} order derivative?

Problem 3 (20): Mapping Kernels, Cost and Separability

Kernel-based spatial range maps allow for operator generality but can be computationally expensive.

(a) (5) For an image of size $n \times n$ and a kernel of size $2k+1 \times 2k+1$ what is the total number of operations in terms of multiplies and adds to map the kernel over the image (to convolve the image by the kernel) only apply it in *valid* locations where the kernel fits fully over the image lattice Λ.

(b) (5) For certain convenient forms of a $2D$ kernel, such as one that samples a Gaussian weighting function of a given variance σ^2, this cost can be reduced by sequential operations of a $1D$ kernel. Show that a convolution by a 2D Gaussian kernel is equivalent to two sequential operations (one vertical and one horizontal) of the appropriate 1D Gaussian kernel. Specify the relationship between the 2D and 1D Gaussian kernel. (Hint: observe the linearity of convolution.)

(c) (5) For this separable kernel of size $2k + 1 \times 1$ and an image of size $n \times n$, how many operations (multiplies and adds) are needed to achieve the equivalent operation as if we just applied the $2D$ kernel (i.e., nonseparable).

(d) (5) Make an argument for or against whether or not we can leverage this separability when looking for step edges in images as zero crossings of the Laplacian operator output.

Submission Process: Submit a single pdf using **Latex** with your answers to these problems, including all plots and discussion. Submit the pdf to Gradescope.

For coding assignments, include your code verbatim in your writeup. Pack the original program files into one zip file and upload it to Canvas. **Code should be well-commented for grading**

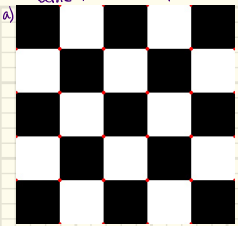
Grading and Evaluation: The credit for each problem in this set is given in parentheses at the stated question (sub-question fraction of points is also given at the sub-questions). Partial credit will be given for both paper and python questions. For python questions, if the code does not run, then limited or no credit will be given.

Alex Goh

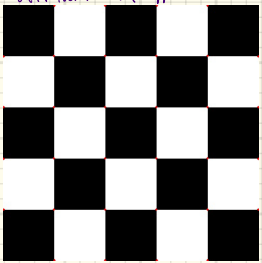
6

a)

Before Non-Max Suppression

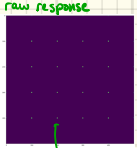


After Non Maximal Supp.



1a

raw response



zoomed

```
def _get_harris_corner(Gx, Gy, win_half_size, threshold):

    winh = win_half_size
    #get some sizes
    row = Gx.shape[0]
    col = Gx.shape[1]

   #structure tensor creation
    Ixx = (Gx**2)
    Ixy = (Gx*Gy)
    Iyy = (Gy**2)

    #matrix of corner locations output as [x,y]    # REPLACE THE CODE BELOW WITH YOUR IMPLEMENTATION
    corner_response_matrix = np.zeros_like(Gx)
    corner_location_matrix = np.zeros((row*col,3))
    num_corners = 0

    #find corners
    for y in range(winh, row-winh):
        for x in range(winh, col-winh):
            #Calculate sum of squares
            winIxx = Ixx[y-winh:y+winh+1, x-winh:x+winh+1]
            winIxy = Ixy[y-winh:y+winh+1, x-winh:x+winh+1]
            winIyy = Iyy[y-winh:y+winh+1, x-winh:x+winh+1]

            #local structure tensor Matrix
            Sxx = winIxx.sum()
            Sxy = winIxy.sum()
            Syy = winIyy.sum()

            #Find determinant and trace, use to get corner response at x,y
            r = min(np.linalg.eigvals([[Sxx,Sxy],[Sxy,Syy]]))

            #If corner response is over threshold, color the point and add to corner list
            if r > threshold:
                corner_response_matrix[y,x] = r
                corner_location_matrix[num_corners,:] = np.array([[y,x,r]])
                num_corners +=1
                #print(corner_location_matrix)
            if (y%100== 0 and x%100== 0) :
                print('Finding Corners at : ',y,x)

    print('Out of loop')
    #remove zeros
    corner_location_matrix = corner_location_matrix[0:num_corners+1,:]
    #sort high response to lowest
    corner_location_matrix = np.flip(corner_location_matrix[np.argsort(corner_location_matrix[:, 2])],
axis=0)
    #return only x,y locations
    corner_location_matrix = corner_location_matrix[:,0:2]
    #print(corner_location_matrix.shape)
    return corner_response_matrix, corner_location_matrix
```

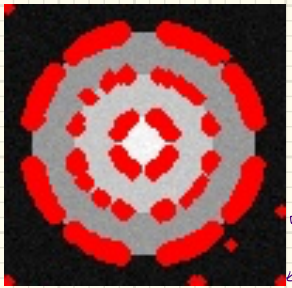
1c)

```
def _overlap(img1, img2, homog_matrix):
    '''Applies a homography transformation to img2 to stitch img1 and img2
    togther.

    Args:
        img1: the first image
        img2: the second image
        homog_matrix: the homography matrix

    Returns:
        stitched_image: the final stitched image
    '''

    #make sure H is normalized to 1 for h33
    np.set_printoptions(precision=4)
    print(homog_matrix)
    H = np.linalg.inv(homog_matrix)
    H = H/H[2,2]
    print(H)

    row = img2.shape[0]
    col = img2.shape[1]

    #find corners of the image set up as y,x standard
    img2_corners = np.zeros((4,3))
    img2_corners[0,:] = np.transpose(np.matmul(H, [[  [0], [0],    [1]]])).flatten()
    img2_corners[1,:] = np.transpose(np.matmul(H, [[  [0], [col],  [1]]])).flatten()
    img2_corners[2,:] = np.transpose(np.matmul(H, [[[row], [col],  [1]]])).flatten()
    img2_corners[3,:] = np.transpose(np.matmul(H, [[[row], [0],  [1]]])).flatten()
    #renormalize
    for i in range(4):
        img2_corners[i,:] = img2_corners[i,:]/ img2_corners[i,2]
    print(img2_corners)

    #Determine offsets due to negative indices, offset will be zero w/ no negative idices
    offsety = int(min(min(img2_corners[:,0]),0))
    offsetx = int(min(min(img2_corners[:,1]),0))
    print("Offsets, y = ",offsetx,offsety)

    #find sizes of output image y,x  = row, col + height,width
    width  = int(max(img2_corners[:,1]) - offsetx)
    height = int(max(img2_corners[:,0]) - offsety)
    print("w,h: ", width, height)
    #create empty output image
    img = np.zeros((height+1, width+1, 3))
    interp = np.zeros(3)

    #insert img1
    img[abs(offsety):int(row+abs(offsety)),offsetx:int(col+offsetx),:] = img1

    #insert img2
    #for index in np.ndindex(img.shape[0], img.shape[1]):
    for i in range(offsety, img.shape[0]):
        for j in range(offsetx, img.shape[1]):

            ind2 = np.asarray([i,j,1])
            ind_warp = np.matmul(homog_matrix,ind2)# transform 2 to 1
            ind_warp = (ind_warp/ind_warp[2]).astype(int)

            #ind(2,502) should map to indwarp(0,0)
            if ind_warp[1]<img2.shape[1] and ind_warp[1]>-1 and ind_warp[0]<img2.shape[0] and ind_warp[0]>-1:
                # print('ind_warp: ',ind_warp[0], ind_warp[1])
                # print('img index = ', i-abs(offsety),j+offsetx)
                #bilinear interpolation
                ind_floor = np.floor(ind_warp[0:2]).astype(int)
                ind_ceil  = np.floor(ind_warp[0:2]).astype(int)

                interpy   = np.asarray([1-(ind_warp[0]%1), (ind_warp[0]%1)])
                interpx   = np.asarray([1-(ind_warp[1]%1),(ind_warp[1]%1)]).T

                for c in range(3):
                    im_array = np.asarray([[img2[ind_floor[0],ind_floor[1],c], img2[ind_floor[0], ind_ceil[1],c]],
                                           [img2[ind_ceil[0], ind_floor[1],c], img2[ind_ceil[0] , ind_ceil[1],c]]])
                    interp[c] = np.matmul(np.matmul(interpy,im_array), interpx)
                img[i+abs(offsety),j+offsetx,:] = interp

            if (i%100== 0 and j%100== 0) :
                print('Smoozing at : ',i,j)

    stitched_image = img/255
    plt.imshow(stitched_image)
    plt.show()

    return stitched_image*255
```

1b)

```
def _get_homography(img1_keypoints, img2_keypoints):
    print(img1_keypoints)
    print(img2_keypoints)

    homog_matrix = np.zeros((3,3))

    img1_keypoints = np.asarray(img1_keypoints)
    img2_keypoints = np.asarray(img2_keypoints)

    A = np.zeros((2*img1_keypoints.shape[0], 9))
    for m in range(img1_keypoints.shape[0]):

        (y,x) = img1_keypoints[m,:]
        (ys,xs) = img2_keypoints[m,:]

        A[2*m,:] = [x,y,1,0,0,0,-x*xs,-y*xs,-xs]
        A[2*m+1,:] = [0,0,0,x,y,1,-x*ys,-y*ys,-ys]

    U,S,Vh = np.linalg.svd(A, compute_uv=True)

    #grab right singular vector corresponding to smallest SingValue,
    x = Vh[-1,:]/ np.linalg.norm(Vh[-1,:])

    homog_matrix = np.reshape(x,(3,3))

    return homog_matrix
```

b)



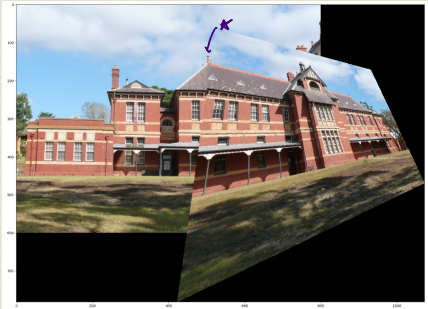
corner detection noise

noise in the image → causes corner detects outside of the concentric circles

the circles are projected to a rectilinear grid, the top, bottom, & sides of the circles are "octagonal", there is no corner response detected because there is no corner, it is an edge. In the structure tensor, in the case of the bottom edge, the I_x, I_{xy} responses will be very small or null, so the smaller of the eigenvalues will be below the threshold.

10

4 correspondences 2



10 corr ↑

20 correspondences 2

4 corres. has very suboptimal performance, one of the points used to estimate the homography may have had an incorrect match. Notice that the top of the building is correctly placed. From the chimney that is starred ✱, when the rest of the image has an incorrect rotation and shear. So it must have correctly identified / corresponded a corner in the top left of im2, but may have misclassified a corner on the right side of im2. The homography is set up to return the right singular vector associated w/ the smallest singular value, so it determined min projection matrix gave the least error.

10 correspondences is well performing, the line that defines the edge of the roof is almost a perfectly a straight line; rotation estimation is good, not perfect. Translation is also almost perfect but is a 3-7 pixels off from perfect. This is probably due to one or two suboptimal matches, RANSAC would provide a more robust feature matching than the brute force matcher used, to reject bad matches.

20 correspondences is slightly better than 10, the shear and rotation is slightly more accurate to the eye. The same 3-7 pixel translation error is still present however.

#2

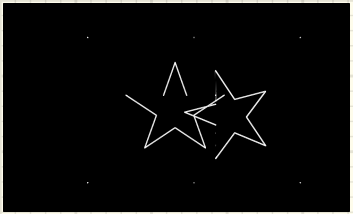
Problem 2 (15): In the last assignment, you have implemented kernels for x and y direction for finding the derivative and gradient. Think of a situation when you have to do the same in an arbitrary direction. Will it be feasible to construct specific kernels for all the possible directions?
The **steerable filters** are a class of filters in which a filter of arbitrary orientation is synthesized as a linear combination of a set of "basis filters."

(a) (5) You are given an image `lines.png`. You have implemented the Sobel horizontal and vertical kernels in HW1. Use these kernels to detect the lines in the image that do not lie along 'x' or 'y' axis and attach the output image in your writeup. Your output should consist of **only** the lines that are not horizontal or vertical. You can use `pipelines/steerable_filter.json` and `modules/steerable_filter.py` to do this task.

You can run the the pipeline using the following command:
`eta build -r requests/steerable_filter_request.json --run-now`

(b) (7) Is the first-order derivative of 2-D Gaussian filter steerable? Why? Prove your answer. If yes, what are its basis functions?

(c) (3) Is the second-order derivative of 2-D Gaussian filter steerable? If yes, give the dimension of its basis. Can this be extended to n^{th} order derivative?



```python
def _apply_steerable_filter(Gx, Gy):
    '''Applies the steerable filter on the given image, using the results
    from sobel kernel convolution.

    Args:
        G_x: the x derivative of the image, given as the result of convolving
            the input image with the horizontal sobel kernel
        G_y: the y derivative of the image, given as the result of convolving
            the input image with the vertical sobel kernel

    Returns:
        g_intensity: the intensity of the input image, defined as
            sqrt(Gx^2 + Gy^2), at every line that does not lie on the
            x-axis or y-axis
    '''
    # TODO
    # REPLACE THE CODE BELOW WITH YOUR IMPLEMENTATION
    g_int = np.zeros_like(Gx)
    g_orient = np.zeros_like(Gx)

    #Create orientation matrix
    g_int = np.sqrt(Gx**2 + Gy**2)
    g_orient = np.arctan2(Gy,Gx)

    #Find pixels where orienation is up, down, left, right
    g_orient  = g_orient*360/(2*np.pi)

    #normalize range of angles to [0,pi] from [-pi,pi]
    g_orient[g_orient<0] = g_orient[g_orient<0] + 180
    g_orient = np.round(g_orient)
    #g_orient = (2* np.round(g_orient/2)).astype(int)

    print(g_orient[385,613])
    plt.subplot(211)
    plt.imshow(g_int, cmap='gray')
    plt.subplot(212)
    g_int[g_orient == 0] = 0
    g_int[g_orient == 90] = 0
    g_int[g_orient == 180] = 0
    plt.imshow(g_int, cmap='gray')
    plt.show()
    return g_int
```

b) $G = e^{-(x^2+y^2)}$

$\frac{\partial G}{\partial x} = -2x\, e^{-(x^2+y^2)}$ $\quad$ $\frac{\partial G}{\partial y} = -2y\, e^{-(x^2+y^2)}$

$= G^{0^\circ}$ $\quad$ $= G^{90^\circ}$

$G^{\theta} = \cos\theta\, G^{0^\circ} + \sin\theta\, G^{90^\circ}$

First derivative of Gaussian is steerable because you can break it up into the x dir + y dir, use a linear combination of the two to generate a arbitrary orientation θ filter with basis vectors G^{0° and G^{90°.

c) 2nd order 2D gaussian is steerable
the dimension of basis is 3

It can be extended to N dimensions w/ N+1 dim basis

Problem 3 (20): Mapping Kernels, Cost and Separability

Kernel-based spatial range maps allow for operator generality but can be computationally expensive.

(a) (5) For an image of size $n \times n$ and a kernel of size $2k+1 \times 2k+1$ what is the total number of operations in terms of multiplies and adds to map the kernel over the image (to convolve the image by the kernel) only apply it in *valid* locations where the kernel fits fully over the image lattice Λ.

(b) (5) For certain convenient forms of a $2D$ kernel, such as one that samples a Gaussian weighting function of a given variance σ^2, this cost can be reduced by sequential operations of a $1D$ kernel. Show that a convolution by a 2D Gaussian kernel is equivalent to two sequential operations (one vertical and one horizontal) of the appropriate 1D Gaussian kernel. Specify the relationship between the 2D and 1D Gaussian kernel. (Hint: observe the linearity of convolution.)

(c) (5) For this separable kernel of size $2k+1 \times 1$ and an image of size $n \times n$, how many operations (multiplies and adds) are needed to achieve the equivalent operation as if we just applied the $2D$ kernel (i.e., nonseparable).

(d) (5) Make an argument for or against whether or not we can leverage this separability when looking for step edges in images as zero crossings of the Laplacian operator output.

Submission Process: Submit a single pdf using **Latex** with your answers to these problems, including all plots and discussion. Submit the pdf to Gradescope.

For coding assignments, include your code verbatim in your writeup. Pack the original program files into one zip file and upload it to Canvas. **Code should be well-commented for grading**

Grading and Evaluation: The credit for each problem in this set is given in parentheses at the stated question (sub-question fraction of points is also given at the sub-questions). Partial credit will be given for both paper and python questions. For python questions, if the code does not run, then limited or no credit will be given.

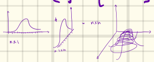
a) n [I] n kernel $2k+1$ [ ] $2k+1$

\# pixels written in valid convolution: $(n-2k)^2$
\# muls / convolution = $(2k+1)^2$
\# adds / convolution = $(2k+1)^2-1$

\# muls = $(2k+1)^2 \cdot (n-2k)^2$
\# adds = $((2k+1)^2-1) \cdot (n-2k)^2$

b) 2D gaussian is Rank 1 and can be represented by an outer product

$G_{2D} = gg^T = [\ \ ][\ \ ] = [\ \ ]$



Due to linearity property of convolution:

$G_{2D} * I = gg^T * I = g * (g^T * I) = (g * I) * g^T$

$G_{2D} = gg^T$

seperable

c) kernel = $2k+1$
image = $n \times n = n^2$ pixels

m = operations/convolution = $((2k+1)^2 + (2k+1)^2 - 1) = (mul + adds)/conv = m$ ops.

\# convolution w/ 2D kernel = $(n-2k)^2$
\# convolution w/ 1D kernel ⇒ $g*(g^T*I)$
$(n-2k) + (n-2k) = 2(n-2k)$

$\#ops_{1D} = 2m(n-2k)$

d) Laplacian is not rank 1, it is not seperable by itself, but we can separate the laplacian of the gaussian to reduce the # of convolutions to increase efficiency.

$\nabla^2 (G * I) = 0 \Rightarrow$ edge location

$= \nabla^2 G * I$

$= \left(\frac{\partial^2}{\partial x^2} G + \frac{\partial^2}{\partial y^2} G\right) * I$

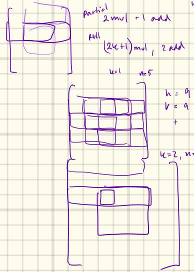
$= \frac{\partial^2}{\partial x^2} G * I + \frac{\partial^2}{\partial y^2} G * I$

$= g_{2x} g_{2x}^T * I + g_{2y} g_{2y}^T * I$

$= g_{2x} * (g_{2x}^T * I) + g_{2y} * (g_{2y}^T * I)$

notes

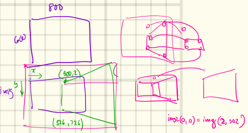
$n = 4$, $k = 1$
$n/2k = 2$
$n/2k - 1$ add



(2k+1) output

$h = 9 \times (3\ mul + 2\ add) = 9 \times (2k+1)\ mul\ 2k\ add$
$v = 9 \times (3\ mul + 2\ adds)$

Partial
3 mul, 2 adds



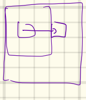
H
1 → 2
$I = J(H(v))$
2 → 1

img2(0,0) = img(2,502)

x: min → max (Right corners)
y: min → max

$(0,0) = H(2,502,1)$
$(2,502,1) = H^{-1}(0,0,1)$

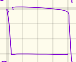
n=5
window = 3
5 - 3 = 2 + 1



img1 → H → img2
← H⁻¹ ←

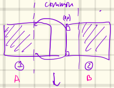
$T_x = 190$ or T_y
$T_y = 414$ T_x

1×2×3 2×2×3 0 = (0,0) 1 (col, 0)



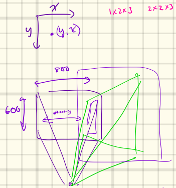
2 (col, row)
3 (0, row)

find all features
↓
find good features to match
common



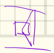
↓
homography on common
↓
map top left of 2 to 1 w/ H^{-1}, 1 → 2
→ find corners

$H : A \to B$ $A(v) = B(H(v))$
$H^{-1} : B \to A$ $A(v) = B(H^{-1}(v))$



im2 warp = H + im2
im2 = H^{-1} + im2 warp

$J(u,v) = I(H^{-1}(u,v))$
$J(t(u,v)) = I$



$x,y = H^{-1}(u,v)$