EECS 504 Foundations of Computer Vision: HW1

Term: Fall 2018
Instructor: Jason J. Corso, EECS, University of Michigan
Due Date: 10/5 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 `hw1.zip` to complete this assignment. All paths in the assignment assume the data is off the local directory.

Problem 1 (20): Estimate homography with Linear Least Squares
We discussed least squares estimation in lecture. As you may know, affine transformation preserves parallelism. For example, a square could become a parallelogram after applying an affine transformation, but it can never be a trapezoid. Now, we consider a more general case where only straight lines are preserved in the transformed space, while other geometrical properties are not guaranteed. This type of transformation is called a projective transformation, or a homography. The transformation can be described mathematically as the following:

$$w \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \tag{1}$$

where $\mathbf{p} = [x, y, 1]$ and $\mathbf{p}' = [x', y', 1]$ are homogeneous coordinates of the original and the transformed point respectively. w is a constant to make sure the third element of $\mathbf{p}'$ is 1.

(a) (6) Assume we know a list of pairs of points under homography, $\{(x_1, y_1), (x_1', y_1')\}, \{(x_2, y_2), (x_2', y_2')\}, ..., \{(x_n, y_n), (x_n', y_n')\}$. Derive the least-squares formulation for estimating the parameters of the homograph.

$$\min \|\mathbf{Ax} - \mathbf{y}\|_2^2. \tag{2}$$

Write down the values of **A**,**x** and **y**, given the points above. Also include your derivation in the writeup.

(b) (4) What is the solution to our estimation problem? How many pairs of points do you think are needed to ensure that the solution is unique? Write down your answer and include a brief explanation in your writeup.

(c) (10) Projective transformations are very common in our real world. When you take pictures of the same object from different viewing angles, the objects in the pictures are projections of the real one. Based on this, we can make useful applications. For example, American football games are played on a rectangular field marked with end lines, side lines and goal lines which are 10 yards inward from each end line, as shown in the Fig. 1. In the NFL broadcasts, virtual yellow lines are drawn on top of real scenes to create a better view for TV audience (see more in How the NFL's magic yellow line works). In this problem, you are expected to implement this based on the homography estimation you derived in previous questions.

To simplify the problem, you will work on two images instead of a video. `football1.jpg` and `football2.jpg` are two screenshots captured from a football game and are provided to you. We use the first image as a baseline (though it is not a good one). The yellow line for the marker 33 has been highlighted, as shown in Fig. 2. You need to select a couple of correspondences in two images first. Then you estimate the transformation matrix and draw the corresponding line in the second picture.

Draw line 33 in picture #2 -> take uv -> multiply By tf

Fill in the code in `hw1p1.py`. The code and images are given in the folder `hw1p1`. Include your code as verbatim and report the output image in your pdf writeup. Also submit your original program files on Canvas.

P1 = 40 (top)
P2 = Pink ribbon (top
P3 = Star, 35
P4 = Helmet, face mask, number

Points X,Y

1)

$$w\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \quad (1)$$

where $\mathbf{p} = [x, y, 1]$ and $\mathbf{p}' = [x', y', 1]$ are homogeneous coordinates of the original and the transformed point respectively. w is a constant to make sure the third element of $\mathbf{p}'$ is 1.

(a) (6) Assume we know a list of pairs of points under homography, $\{(x_1, y_1), (x'_1, y'_1)\}, \{(x_2, y_2), (x'_2, y'_2)\}, ..., \{(x_n, y_n), (x'_n, y'_n)\}$. Derive the least-squares formulation for estimating the parameters of the homograph.

$$\min \|\mathbf{Ax} - \mathbf{y}\|_2^2. \quad (2)$$

Write down the values of **A**,**x** and **y**, given the points above. Also include your derivation in the writeup.

$p = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$ $p' = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$ $H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix}$ $\min \|Ax - y\|_2^2$

$w p' = Hp$

$\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix} = Hp = \begin{matrix} h_{11}x + h_{12}y + h_{13} \\ h_{21}x + h_{22}y + h_{23} \\ h_{31}x + h_{32}y + 1 \end{matrix}$

$h_{11}x + h_{12}y + h_{13} - wx' = 0$
$h_{21}x + h_{22}y + h_{23} - wy' = 0$
$h_{31}x + h_{32}y + 1 = w$

$h_{11}x + h_{12}y + h_{13} - (h_{31}x + h_{32}y + 1)x' = 0$
$h_{21}x + h_{22}y + h_{23} - (h_{31}x + h_{32}y + 1)y' = 0$

$x' = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + 1}$ $y' = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + 1}$

$$\underbrace{\begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -xx' & -yx' & -x' \\ 0 & 0 & 0 & x & y & 1 & -xy' & -yy' & -y' \end{bmatrix}}_{A\ (2n \times 9)} \underbrace{\begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ 1 \end{bmatrix}}_{x\ (9\times 1)} - \underbrace{\begin{bmatrix} 0 \\ 0 \end{bmatrix}}_{y\ (2n)} = 0$$

(b) (4) What is the solution to our estimation problem? How many pairs of points do you think are needed to ensure that the solution is unique? Write down your answer and include a brief explanation in your writeup.

$\min \|Ax - y\|_2^2 \Rightarrow \left(\sqrt{(Ax-y)^T(Ax-y)}\right)^2$

$(Ax)^T Ax = x^T A^T A x$

$\frac{d}{dx}(x^T A^T A x) = 0 = 2A^T A x$

$A = U\Sigma V'$
$= A^T A x$
$= V\Sigma U' U \Sigma V' x$
$V' 0 = V' V \Sigma^2 V' x$
$0 = \Sigma^2 V' x$

$\begin{bmatrix} \sigma_1^2 & & \\ & \ddots & \\ & & \sigma_9^2 \end{bmatrix} [v_1 \dots v_n]' x = 0$

x equals the transpose of the right-singular vector corresponding to the smallest singular value of A or any zero valued singular value of A

to ensure uniqueness, we need 8 equations because we have 8 unknowns, $2n = 8, n = 4$

or we need rank of 8 for A

3) (a) (3) Convolution is a simple mathematical operation which is fundamental to many common image processing operations like blurring, edge-detection etc. Discrete convolution in 1-D is given by:

$$y[n] = x[n] * h[n] = \sum_{-\infty}^{\infty} x[n] \cdot h[n-m] \quad (3)$$

Think how would you perform convolution on the boundary values. Perform a 2-D convolution on the following image **A** with kernel **B**:

$$\mathbf{A} = \begin{bmatrix} 4 & 8 \\ 2 & 3 \\ 1 & 6 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 1 & 2 \\ 2 & 0 \end{bmatrix} \quad (4)$$

(0,0) (0,1)
(1,0) (1,1)

$y[q,r] = \sum_i \sum_j A[i,j] B[q-i, r-j]$, $A \in \mathbb{R}^{3\times 2, m\times n}$, $B \in \mathbb{R}^{2\times 2, s\times t}$

for $q = 0{:}3$, $r = 0{:}2$

$y[q,r] = \sum_{i=0}^{2} \sum_{j=0}^{1} A[i,j] B[q-i, r-j]$

output size $= (m+s-1) \times (n+t-1)$
$= (3+2-1) \times (2+2-1)$
$= 4 \times 3$, $q \times r$

$y[0,0] = A[0,0]B[0,0] + A[1,0]B[-1,0] + A[2,0]B[-2,0] + A[0,1]B[0,-1] + A[1,1]B[-1,-1] + A[2,1]B[-2,-1]$

$= 4(1)$

for B out of bounds, set to zero

$y = \begin{bmatrix} 4 & 16 & 16 \\ 10 & 23 & 6 \\ 5 & 14 & 12 \\ 2 & 12 & 0 \end{bmatrix}$ from numpy

```
def _convolve(kernel, in_img):
    '''Convolve the input image "in_img" with the kernel "kernel".
    Assume the kernel has already been flipped.

    Args:
        kernel: a 2d kernel that is assumed to be flipped appropriately
        in_img: the input image

    Returns:
        out_img: the result of convolving the input image with the specified
            kernel
    '''
    # Get some sizes
    m1, n1 = in_img.shape
    m2 = kernel.shape[0]
    n2 = kernel.shape[1]
    #print(m1,n1)
    # Image Frame
    pad = int(max(m2,n2))//2 #3x3//2 = 1, 5x5//2=2 width, 15x15//2 = 7
    out_img = np.zeros((m1-2*pad,n1-2*pad))

    # Perform Convolution with no padding
    for (x,y) in np.ndindex(int(m1-2*pad),int(n1-2*pad)):
         out_img[x,y]= np.dot(np.reshape(kernel,(m2*n2)),np.reshape((in_img[x:x+m2,y:y+n2]),(m2*n2)))

    return out_img
```

```
def _create_x_derivative_kernel():
    return np.array([[-1,1]])

def _create_y_derivative_kernel():
    return np.transpose(np.array([[-1,1]]))
```

3b) Potts model $= \left(\text{for } n \text{ for } n + \text{for } n \text{ for } n\right) = 2(n-1)^2 + 1 \Rightarrow O(n^2)$

3c) Original = 168,022 + 173,434 = 341,456
Gaussian = 82,083 + 90,643 = 172,726

} the gaussian filter smoothes the image in the x,y directions so # of neighboring pixels w/ different values goes down.

3D)

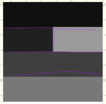


Figure 4: img_1.jpg

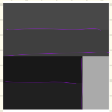


Figure 5: img_2.jpg

$3 \cdot 1/4 = 3.25$ $2 + \frac{1}{2} + \frac{3}{4} = 3.25$

potts = 2510 potts = 2510

the length of the edges for both images is 3 1/4 lengths, which makes sense because the energies are the same

```
def _calculate_potts_energy(data):
    '''Calculates the pott's energy of an image, given the x-derivative
    and y-derivative of the image at each pixel. Beta is assumed to be 1.

    Args:
        data: the data configuration values

    Returns:
        potts_energy: the Pott's Energy for the original image
    '''
    xder = np.load(data.x_derivative_path)["filtered_matrix"]
    yder = np.load(data.y_derivative_path)["filtered_matrix"]

    # ADD CODE HERE
    # Write code to calculate the Pott's Energy of the original image given
    # 'x_derivative' and 'y_derivative'
    potts_energy = 0

    xen = np.count_nonzero(xder[0:-1, 0:xder.shape[1]])
    yen = np.count_nonzero(yder[0:yder.shape[0], 0:-1])
    print("X Energy: ",xen, "Y Energy: ",yen)
    potts_energy = xen + yen
    xen,yen = 0,0

    for ind,x in np.ndenumerate(xder):
        #print(ind)
        if xder[ind] != 0:
            xen +=1
    for ind,x in np.ndenumerate(yder):
        if yder[ind] != 0:
            yen +=1
    print("X Energy2: ",xen, "Y Energy2: ",yen)
    return potts_energy
```

1c)


```
def calc_homography(n):
    pts = np.load('football_pts_'+str(n)+'.npy')
    print('POINTS:', pts, '\n')
    print(('Points.shape', pts.shape, '\n'))
    print('Points.shape[0]', pts.shape[0], '\n')
    print('Points[0,0]', pts[0,0], '\n')

    X = pts[0,0,0]
    print ('X1:', X, "\n")

    A = np.zeros((2*pts.shape[1], 9))
    for m in range(pts.shape[1]):
        print('m: ', m)
        (x,y) = pts[0,m]
        (xs,ys) = pts[1,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]

    #np.set_printoptions(precision=3)
    #print(A)

    U,S,Vh = np.linalg.svd(A, compute_uv=True)
    #print('Vh.shape:', Vh.shape)

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

    H = np.reshape(x,(3,3))
    print(H)
    return H
```

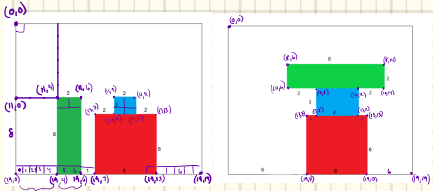
```
H = calc_homography(n)

fig = plt.figure()
ax = fig.add_subplot(111)#
ax.imshow(img2)
u=[1210,1701]
v=[126,939]
#transform the points
u_t = np.zeros_like(u)
v_t = np.zeros_like(u)

test = np.matmul(H ,np.array([u[0], v[0], 1]))
print('test:', test/test[-1])

[temp_u,temp_v,w] = np.matmul(H,np.array([u[0], v[0], 1]))
(u_t[0],v_t[0]) = (temp_u/w, temp_v/w)

[temp_u,temp_v,w] = np.matmul(H,np.array([u[1], v[1], 1]))
(u_t[1],v_t[1]) = (temp_u/w, temp_v/w)

ax.plot(u_t, v_t, color='yellow')
ax.set(title='Football image 2')
plt.show()
```

zoomed in

2) a) they will observe the same reflection intensity because the lambertian model doesn't include specularity and angle of incidence of

b) a drawback of lambertian model is a failure to capture specularity, so a surface that would not be represented well in a lambertian model is a mirror like surface, such as brushed aluminum.

4)



Setting up the { green as active → pick corner points so that there is no reflection
blue as active → pick corresponding points
Red doesn't move, so its transformation is $I_{3\times3}$

Assume color Bleed between RGB channels

Apply transforms individually to set of pixels which compromise each logo

- select pixels for red logo piece
- white pixels to a similar shape zero matrix for each logo
- apply transform for each logo
- Sum the transformed logo images into output

Green: $\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$

$\begin{matrix} (x,y) \\ \begin{bmatrix} 11,4 \\ 11,6 \\ 19,4 \\ 19,6 \end{bmatrix} \end{matrix} \Rightarrow \begin{matrix} (x',y') \\ \begin{bmatrix} 10,6 \\ 8,16 \\ 13,11 \\ 8,19 \end{bmatrix} \end{matrix}$

Blue: $\begin{matrix} (x,y) \\ \begin{bmatrix} 6,8 \\ 11,8 \\ 11,11 \\ 8,11 \end{bmatrix} \end{matrix} \rightarrow \begin{matrix} (x',y') \\ (13,8) \\ (20,8) \\ (20,12) \\ (13,12) \end{matrix}$

$h_{11}x + h_{12}y + h_{13} = x'$
$h_{21}x + h_{22}y + h_{23} = y'$

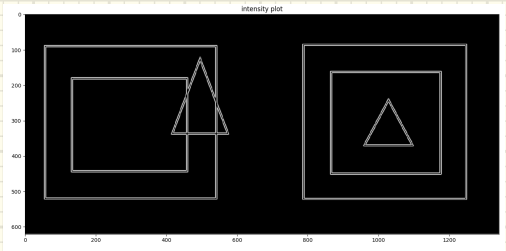
$A \quad x \quad = \quad b$

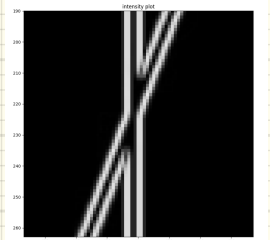
$\Rightarrow \begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -x'x & -x'y \\ 0 & 0 & 0 & x & y & 1 & -y'x & -y'y \\ & & & & \vdots & & & \end{bmatrix} \cdot \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ \vdots \end{bmatrix}$

$Ax = b$
$AA^T x = Ab$
$x = (AA^T)^{-1}Ab$

Green: $\begin{bmatrix} 0 & -1 & 19 \\ 1 & 0 & -5 \end{bmatrix}$ Blue: $\begin{bmatrix} 1.5 & -0 & -15 \\ 0 & 2 & -10 \end{bmatrix}$

5) a)

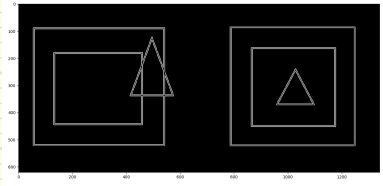




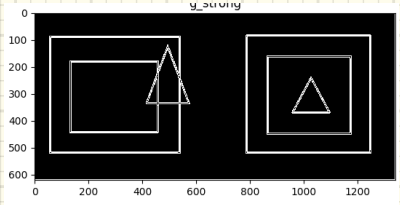
Intensity output image does not give "hard" edges, there is a "soft" pixel next to each hard edge due to the gaussian smoothing + the sobel convolution.

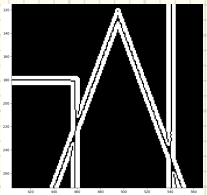
5b)

Non-max suppression output:



5c)





5d) canny detector is not rotationally invariant in that if you have image with pixels (x,y) and you rotate image 45° so corresponding points are @ (x',y') the new location will not be an integer value, thus the output of canny(I) and canny(I') will not have linear correspondence of edge locations. However, Canny(I') will still detect the same edge and if we ignore the discretized grid mapping or handle it losslessly, then the canny edge detector should be rotationally invariant.

```
def _create_intensity_orientation_matrices(Gx, Gy):
    '''Creates two matrices: one for intensity and one for orientation.
    The intensity at each pixel is defined as sqrt(Gx^2 + Gy^2) and
    the orientation of each pixel is defined as arctan(Gy/Gx).

    Args:
        Gx: the result of convolving with the "sobel_horizontal" kernel
        Gy: the result of convolving with the "sobel_vertical" kernel

    Returns:
        (g_intensity, orientation): a tuple with the first element as
            the intensity matrix and the second element as the
            orientation matrix.
    '''
    g_int = np.zeros_like(Gx)
    g_orient = np.zeros_like(Gx)

    for ind, _x in np.ndenumerate(Gx):
        g_int[ind] = np.sqrt(Gx[ind]**2 + Gy[ind]**2)
        g_orient[ind] = np.arctan2(Gy[ind],Gx[ind])

    return (g_int, g_orient)
```

```
def _double_thresholding(g_suppressed, low_threshold, high_threshold):
    '''Performs a double threhold. All pixels with gradient intensity larger
    than 'high_threshold' are considered strong edges, all pixels with gradient
    intensity in between 'high_threshold' and 'low_threshold' are considered
    weak edges, and all pixels with gradient intensity smaller than
    'low_threshold' are suppressed to 0.

    Args:
        g_suppressed: the gradient intensities of all pixels, after
            non-maxiumum suppression
        low_threshold: the lower threshold in double thresholding
        high_threshold: the higher threshold in double thresholding

    Returns:
        g_thresholded: the result of double thresholding
    '''
    g_thresholded = np.zeros_like(g_suppressed)

    #extract regions
    ind_h = (g_suppressed > high_threshold)
    ind_w = ((g_suppressed>low_threshold) & (g_suppressed < high_threshold))
    ind_l =  g_suppressed<low_threshold

    g_thresholded[ind_h] = STRONG
    g_thresholded[ind_w] = WEAK
    g_thresholded[ind_l] = SUPPRESSED

    plt.imshow(g_thresholded, cmap='gray')
    plt.show('g_thresholded')
    plt.show()

    return g_thresholded
```

```
def _non_maximum_suppression(g_int, orientation, input_image):
    '''Performs non-maximum suppression. If a pixel is not a local maximum
    (not bigger than it's neighbors with the same orientation), then
    suppress that pixel.

    Args:
        g_intensity: the gradient intensity of each pixel
        orientation: the gradient orientation of each pixel
        input_image: the input image

    Returns:
        g_sup: the gradient intensity of each pixel, with some intensities
            suppressed to 0 if the corresponding pixel was not a local
            maximum
    '''
    p = np.pi
    g_sup = g_int

    #normalize range of angles to [0,pi] from [-pi,pi]
    orientation[orientation<0] = orientation[orientation<0] + p

    plt.imshow(g_int, cmap='gray')
    plt.title('intensity plot')
    plt.show()

    for (x,y) in np.ndindex(g_int.shape[0]-1,g_int.shape[1]-1):

        t_xy = orientation[x,y]

        # gradient = 0,180; edge is north,south dir, check east and est
        if (t_xy <= p/8 or t_xy >= 7*p/8):
            if (g_int[x,y] < g_int[x,y+1] and g_int[x,y] < g_int[x,y-1]):
                g_sup[x,y] = 0
        # gradient = 90,270; edge is east,west dir, check north and south
        if (t_xy > 3*p/8 and t_xy <= 5*p/8):
            if (g_int[x,y] < g_int[x+1,y] and g_int[x,y] < g_int[x-1,y]):
                g_sup[x,y] = 0
        # gradient = +45; edge is northwest, southeast, check northeast and southwest
        if (t_xy > p/8 and t_xy <= 3*p/8):
            if (g_int[x,y] < g_int[x-1,y-1] and g_int[x,y] < g_int[x+1,y+1]):
                g_sup[x,y] = 0
         # check -45=+135, edge is northeast,southwest, check northwest and southeast
        if (t_xy > 5*p/8 and t_xy <= 7*p/8):
            if (g_int[x,y] < g_int[x+1,y-1] and g_int[x,y] < g_int[x-1,y+1]):
                g_sup[x,y] = 0
    plt.imshow(g_sup, cmap='gray')
    plt.show()
    return g_sup
```

```
def _hysteresis(g_thresholded):
    '''Performs hysteresis. If a weak pixel is connected to a strong pixel,
    then the weak pixel is marked as strong. Otherwise, it is suppressed.
    The result will be an image with only strong pixels.

    Args:
        g_thresholded: the result of double thresholding

    Returns:
        g_strong: an image with only strong edges
    '''
    #intialize
    g_strong = g_thresholded

    ## find weak edges connected to hard edges and suppress
    x,y = np.where(g_thresholded==WEAK)

    for i in range(x.shape[0]):
        #check a 3x3 cell for strongs
        if np.any(g_thresholded[x[i]-1:x[i]+2, y[i]-1:y[i]+2] == STRONG):
            g_strong[x[i],y[i]] = SUPPRESSED

    plt.imshow(g_strong, cmap='gray')
    plt.title('g_strong')
    plt.show()
    return g_strong
```

629 | 1350
1329
21

7+15=22

1350

Figure 1: football field

Figure 2: yellow line at marker 33

Problem 2 (6): Lambertian Model

The Lambertian model of reflectance is $R(\mathrm{x}) = \rho \boldsymbol{\ell}(\mathrm{x})^{\mathrm{T}} \mathbf{n}(\mathrm{x})$, as shown in Fig. 3.

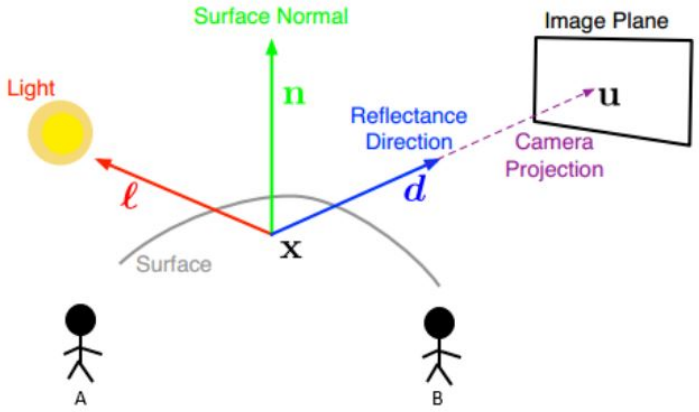


Figure 3: Lambertian model

(a) (3) There are two people standing at location A and B. Who will observe stronger reflection at point x? Why?

(b) (3) Tell one drawback of Lambertian model and name a specific object in the real world whose reflectance is not well modeled by the Lambertian model.

Problem 3 (15): Potts model

(a) (3) Convolution is a simple mathematical operation which is fundamental to many common image processing operations like blurring, edge-detection etc. Discrete convolution in 1-D is given by:

$$y[n] = x[n] * h[n] = \sum_{-\infty}^{\infty} x[n] \cdot h[n-m] \tag{3}$$

Think how would you perform convolution on the boundary values. Perform a 2-D convolution on the following image **A** with kernel **B**:

$$\mathbf{A} = \begin{bmatrix} 4 & 8 \\ 2 & 3 \\ 1 & 6 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 1 & 2 \\ 2 & 0 \end{bmatrix} \tag{4}$$

(b) (6) Recall the Potts model we discussed in the lecture. Based on that model, the energy of an image I is

$$E(I) = \beta \sum_{s=1}^{n-1} \sum_{t=1}^{n} (\mathbf{1}(I(s,t) \neq I(s+1,t))) + \beta \sum_{s=1}^{n} \sum_{t=1}^{n-1} (\mathbf{1}(I(s,t) \neq I(s,t+1))) \tag{5}$$

Implement a function to compute the energy of the image `data/Michigan_letter.jpg` using this model and submit the value of the resultant energy. Use $\beta = 1$ in your implementation. You need to fill in the missing parts of the functions `_create_x_derivative_kernel()`, `_create_y_derivative_kernel()` and `_convolve()` in `modules/convolution.py` and `_calculate_potts_energy()` in `modules/_calculate_potts_energy.py`.

Compute the complexity O(n) of this algorithm.

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

(c) (3) Perform a 2-D convolution on the given image using the `_create_gaussian_kernel()` in `modules/convolution.py` and calculate its Potts energy. You need to use the pipeline provided in `pipelines/potts_energy_gaussian.json` to do this. Explain the difference in energies for the original and final image. (Hint: Think in the direction as to what the given kernel is doing to the image).

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

(d) (3) You are given two images `data/img_1.jpg` and `data/img_2.jpg`. You need to modify the `requests/potts_energy_request.json` to input these images. Compute the energies of these images using the function you implemented in part (b). How are the energies of the two images related? Explain your observations.

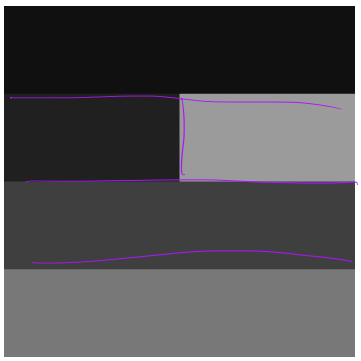


Figure 4: img_1.jpg

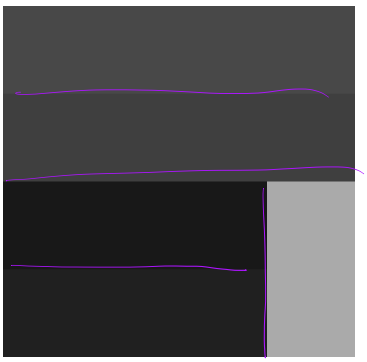


Figure 5: img_2.jpg

Problem 4 (14): Transformation
Imagine you are in Lego world. You are given 3 blocks - red, blue and green. The blocks are arranged as shown in Fig 6. You have to rearrange and stack the lego blocks to form structure shown in Fig 7. The images are 20x20 and the top-left corner is (0,0). The edge-length (in number of pixels) of the blocks are provided in the figure.

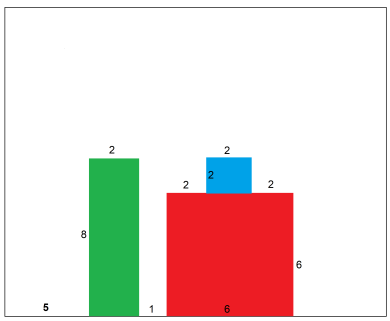


Figure 6: Current state

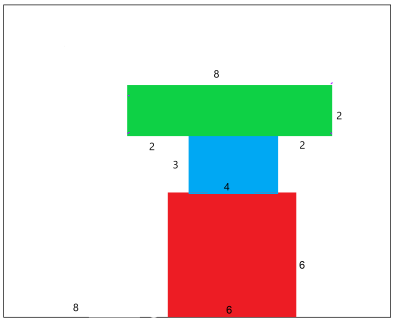


Figure 7: Final state

(a) (10) Explain your approach in detail. Provide the transformation matrix (matrices) you have created to achieve Fig 7.

(b) (4) State which transformation(s) is (are) used in part (a). Do these transformation matrix(matrices) preserve orientation, angle and parallelism of line segments?

Problem 5 (25): Edge Detection
An edge is a place where image intensity changes abruptly. Edge detection is the method of finding these boundaries of objects within images. Through this problem you are going to implement a well-known algorithm called Canny Edge Detector. You need to run the detector on `data/Canny_input.jpg` by using `pipelines/edge_detection.json`.

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

(a) (4) The first step in the Canny edge detector is to get the gradient images. One of the most popular ways to compute the gradient images is by using vertical and horizontal sobel operators. Fill out the missing part of the functions `_create_sobel_horizontal_kernel()` and `_create_sobel_vertical_kernel()` in file `modules/convolution.py` to create the kernels, and function `_create_intensity_orientation_matrices()` in file `modules/canny_edge_detector.py` to compute the intensity and orientation matrices of the gradients.
Report the gradient intensity image in your writeup. Observe the edges extracted from this. Does it give a sharp edge? Explain your observation.
does not give a "sharp" edge - there is some blurring along edge, artifact from convolution + sqrt (int) Gaussian blur

(b) (7) Implement an algorithm to get a sharp edge response from the output of part (a). This process is called Non-Maximum Suppression. Fill in the missing part of the function `_non_maximum_suppression()` in the file `modules/canny_edge_detector.py` and report the output image in your writeup.

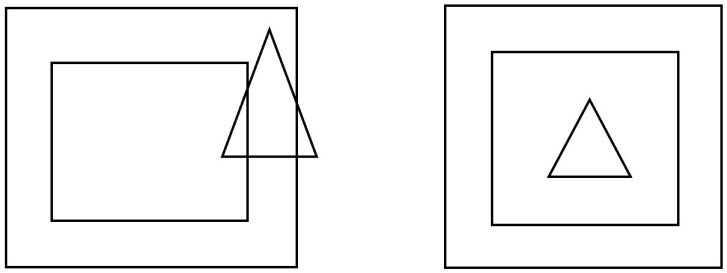


Figure 8: Canny_input.jpg

(c) (10) Observe the output from part (b). The edge-pixels can be considered as strong, weak, or suppressed based on the intensity of their gradients. You should then perform Edge Tracking by connecting each weak-edged pixel to any neighboring strong-edged pixel (if any) to get the complete edge. Fill in the missing part of the code for the functions `_double_thresholding()` and `_hysteresis()` in the file `modules/canny_edge_detector.py` to implement double thresholding and perform edge tracking by hystersis, respectively. Include the output image in your report.

(d) (4) Is the Canny Edge Detector rotation invariant? Provide a mathematical proof or counter argument.

(e) (**Extra Credits: 10**) Observe the line-segments formed by intersecting edges in the output of part (c). Write a function to get the coordinates of these line-segments. Be creative! For this problem, you can use `pipelines/line_segments.json` and modify `modules/find_line_segments.py`. You can run the the pipeline using the following command:

```
eta build -r requests/line_segments_request.json --run-now
```

You should provide the output as a .json file called `output_points.json` (You can use `eta.core.serial.write_json()` to write into .json format). Your `output_points.json` should be in the following format:

```
{
    "Line_segments": {
                        "No": 1
                        "coordinates": [(x1, y1), (x1',y1')}
                      }
                      {
                        "No": 2
                        "coordinates": [(x2, y2), (x2',y2')}
                      }
                      .
                      .
                      .
}
```

Submission Process: Submit a single pdf 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. The problem description will clarify whether you need to attach your code verbatim or turn in the original files for that problem, or both. **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.