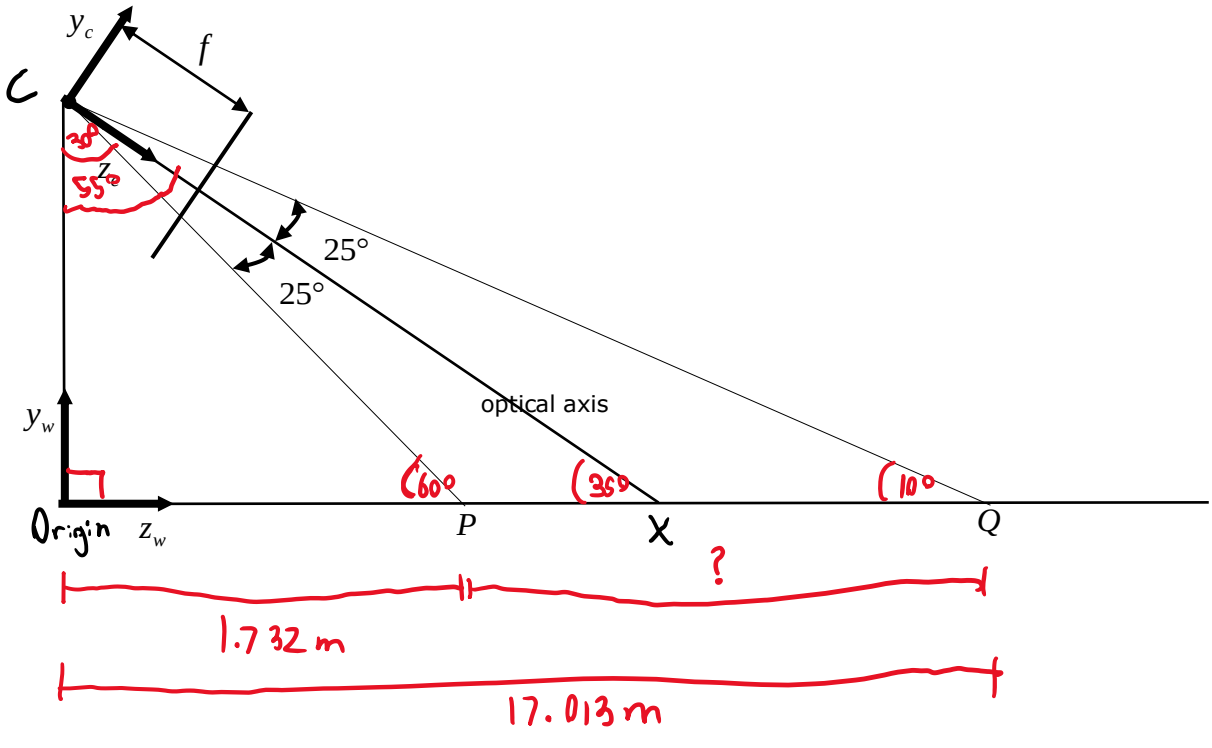
ECE 4554 / 5554: Computer Vision: Homework 2 (revised Sept. 23)

Fall 2020

Instructions

- The assignment is due at Canvas before midnight on Sept. 30. Late submissions are allowed at the cost of 1 token per 24-hour period. A submission received only a minute after midnight will cost an additional token.
- Each problem is worth 10 points. Notice that one of the problems is required for 5554 students, but is optional (extra-credit) for 4554 students.
- Prepare an answer sheet that contains all of your written answers in a single file named `LastName_FirstName_HW2.pdf`. (Use your own name, of course.) Handwritten solutions are permitted, but they must be easily legible to the grader. Do not place this pdf file inside the zip file that is described in the next bullet.
- Place all of your Python-related files (`*.py, *.npy, *.png`) into a single zip file named `LastName_FirstName_HW2.zip`. (Again, use your own name.) Upload 2 files to Canvas: this zip file and the pdf file that is described above.
- For the Python implementation problems, try to ensure that the grader can run your code "out of the box". For example, do not encode absolute path names for input files; provide relative path names, assuming that the input files reside in the same directory as the source files.
- If any plots are required, include them in your answer sheet (the pdf file). Also, your Python code must display the plots when executed.
- After you have submitted to Canvas, it is your responsibility to download the files that you submitted and verify that they are correct and complete. *The files that you submit to Canvas are the files that will be graded.*

Problem 1. Consider a camera that is viewed from the side, as shown in the figure below. (The figure is not drawn to scale). In this two-dimensional side view, a camera-centered coordinate system is represented by (y_c, z_c). (The x_c axis is perpendicular to this drawing.) The camera is mounted at a position 3 meters above the ground, which is assumed to be planar. The camera's optical axis makes an angle of 35° with respect to the ground. The focal length is f = 100 mm. A world coordinate system is represented by (y_w, z_w) in our 2-dimensional side view, and its z_w axis is aligned with the ground.



The field of view of a camera is often specified using vertical and horizontal angles, relative to the point of projection. Assume that this particular camera has a vertical field of view of 50°, which is shown in the figure as 25° above the optical axis combined with 25° below the optical axis.

In the figure, points *P* and *Q* lie on the ground at the limits of the camera's field of view. (The distance separating *P* and *Q* is sometimes called the field of view informally, and for some applications it can be very useful to know that distance.) Solve for the distance (in meters) separating points *P* and *Q* on the ground in this diagram.

Given :

$OX = 3m$ Angle made by the optical axis and the ground: $\angle CXO = 35°$

$\angle COX = 90°$

Adding up angles $\angle OCX + \angle COX + \angle OXC = 180°$

Using this we can get $\angle OCX: 180° - (90° + 35°) = 55°$

Therefore the angle $\angle OCP = \angle OCX - \angle PCX = 55° - 25° = 30°$

In triangle OCP, $\angle CPO = 180° - (90° + 30°) = 60°$

Now use tan. to get adjacent side OP:

$$\tan 60° = \frac{OC}{OP} = \frac{3m}{OP} \Rightarrow OP = 1.732\ m$$

Now we find $\angle CQO = 180° - (55° + 25° + 90°) = 10°$

In triangle OCQ, $\tan 10° = \frac{OC}{OQ} = \frac{3m}{OQ}$

$$OQ = \frac{3m}{\tan 10°} = 17.013\ m$$

Thus,

$PQ = 17.013\ m - 1.732\ m = \boxed{15.28\ m}$

Problem 2. We recently discussed the 2-dimensional *affine transformation*, which maps a point $[x \quad y]^T$ to a new location $[x' \quad y']^T$ in the plane. The transformation can be represented using the following equation:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a_1 & a_2 \\ a_4 & a_5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} a_3 \\ a_6 \end{bmatrix} \tag{1}$$

Six constant parameters, shown here as a_1 through a_6, completely specify the transformation. The equivalent expression using homogeneous coordinates is often more useful to us:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{2}$$

a) The expression shown in equation (2) can also be written as the matrix equation $\mathbf{p}' = A\,\mathbf{p}$, where A is the 3×3 matrix shown above. Write an expression (using scalar values, possibly including a_1 through a_6) for the inverse mapping, $\mathbf{p} = A^{-1}\,\mathbf{p}'$. Your answer should also be an affine transformation.

b) Consider the 3 points $[x \quad y]^T = [0 \quad 0]^T$ and $[x \quad y]^T = [1 \quad 0]^T$ and $[x \quad y]^T = [0 \quad 1]^T$. For each case, use equation (2) to write an expression (using scalar values, possibly including a_1 through a_6) for the corresponding new location $[x' \quad y']^T$. (Notice that these 3 points can provide insights about how one coordinate reference frame maps onto another coordinate reference frame. Another take-away is that affine transformation can map the vertices of *any triangle* to and from the vertices of a reference triangle.)

c) Consider the composition of 2 affine transformations. An example is $\mathbf{p}' = B\,A\,\mathbf{p}$, where B and A are both 3×3 affine transformation matrices. Show that the product BA is also a 3×3 affine transformation matrix.

d) We can interpret part (c) as transformation by A followed by transformation by B. Should we expect the same result if the order of the transformations were reversed? (I.e., transformation by B followed by transformation by A.) Provide a succinct reason for your answer.

a.)

$p' = Ap \qquad p = [x, y, 1]^T \qquad p' = [x', y', 1]^T$

$$A = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} x' - a_3 \\ y' - a_6 \end{bmatrix} = \begin{bmatrix} a_1 & a_2 \\ a_4 & a_5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \Rightarrow \begin{bmatrix} a_1 & a_2 \\ a_4 & a_5 \end{bmatrix}^{-1} \begin{bmatrix} x' - a_3 \\ y' - a_6 \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}$$

Since, $\begin{bmatrix} a_1 & a_2 \\ a_4 & a_5 \end{bmatrix}^{-1} = \frac{1}{a_1 a_5 - a_2 a_4} \begin{bmatrix} a_5 & -a_2 \\ -a_4 & a_1 \end{bmatrix}$

Thus, $\begin{bmatrix} x \\ y \end{bmatrix} = \frac{1}{a_1 a_5 - a_2 a_4} \begin{bmatrix} a_5 & -a_2 \\ -a_4 & a_1 \end{bmatrix} \begin{bmatrix} x' - a_3 \\ y' - a_6 \end{bmatrix}$

$\Rightarrow \begin{bmatrix} x \\ y \end{bmatrix} = \frac{1}{a_1 a_5 - a_2 a_4} \begin{bmatrix} a_5 & -a_2 \\ -a_4 & a_1 \end{bmatrix} \begin{bmatrix} x' \\ y' \end{bmatrix} + \begin{bmatrix} -a_3 \\ -a_6 \end{bmatrix}$

Thus,

$$\underbrace{\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}}_{P} = \underbrace{\begin{bmatrix} \frac{a_5}{a_1 a_5 - a_4 a_2} & \frac{-a_2}{a_1 a_5 - a_2 a_4} & -a_3 \\ \frac{-a_4}{a_1 a_5 - a_2 a_4} & \frac{a_1}{a_1 a_5 - a_2 a_4} & -a_6 \\ 0 & 0 & 1 \end{bmatrix}}_{A^{-1}} \underbrace{\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}}_{P'}$$

b.)

$[x, y]^T = [0, 0]^T$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} a_1 \cdot 0 + a_2 \cdot 0 + a_3 \cdot 1 \\ a_4 \cdot 0 + a_5 \cdot 0 + a_6 \cdot 1 \\ 0 \cdot 0 + 0 \cdot 0 + 1 \cdot 1 \end{bmatrix} = \begin{bmatrix} a_3 \\ a_6 \\ 1 \end{bmatrix}$$

Thus, $[x', y']^T = [a_3 \quad a_6]^T$

$[x', y']^T = [1 \quad 0]^T$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} a_1 \cdot 1 + a_2 \cdot 0 + a_3 \cdot 1 \\ a_4 \cdot 1 + a_5 \cdot 0 + a_6 \cdot 1 \\ 0 \cdot 1 + 0 \cdot 0 + 1 \cdot 1 \end{bmatrix} \Rightarrow \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_1 + a_3 \\ a_4 + a_6 \\ 1 \end{bmatrix}$$

Thus, $[x' \quad y']^T = [a_1 + a_3 \quad a_4 + a_6]^T$

$[x \quad y]^T = [0 \quad 1]^T$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} a_1 \cdot 0 + a_2 \cdot 1 + a_3 \cdot 1 \\ a_4 \cdot 0 + a_5 \cdot 1 + a_6 \cdot 1 \\ 0 \cdot 0 + 0 \cdot 1 + 1 \cdot 1 \end{bmatrix} \Rightarrow \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_2 + a_3 \\ a_5 + a_6 \\ 1 \end{bmatrix}$$

Thus, $[x' \quad y']^T = [a_2 + a_3 \quad a_5 + a_6]^T$

c.)

Considering the following,

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} x'' \\ y'' \\ 1 \end{bmatrix} = \begin{bmatrix} b_1 & b_2 & b_3 \\ b_4 & b_5 & b_6 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$$

Rewrite,

$$\begin{bmatrix} x'' \\ y'' \\ 1 \end{bmatrix} = \begin{bmatrix} b_1 & b_2 & b_3 \\ b_4 & b_5 & b_6 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Now,

$$B = \begin{bmatrix} b_1 & b_2 & b_3 \\ b_4 & b_5 & b_6 \\ 0 & 0 & 1 \end{bmatrix}, \quad A = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ 0 & 0 & 1 \end{bmatrix}$$

Thus,

$$AB = \begin{bmatrix} b_1 \cdot a_1 + b_2 \cdot a_4 + b_3 \cdot 0 & b_1 a_2 + b_2 a_5 + 0 & b_1 a_3 + b_2 \cdot a_6 + b_3 \\ b_4 \cdot a_1 + b_5 \cdot a_4 + b_6 \cdot 0 & b_4 \cdot a_2 + b_5 a_5 + 0 & b_4 \cdot a_3 + b_5 \cdot a_6 + b_6 \\ 0 \cdot a_1 + 0 \cdot a_4 + 0 \cdot 1 & 0 + 0 + 0 & 0 + 0 + 1 \end{bmatrix}$$

$$= \begin{bmatrix} a_1 b_1 + a_4 b_2 & b_1 a_2 + b_2 a_5 & b_1 a_3 + b_2 a_6 + b_3 \\ a b_4 + a_4 b_5 & b_4 a_2 + b_5 a_5 & b_4 a_3 + b_5 a_6 + b_6 \\ 0 & 0 & 1 \end{bmatrix}$$

Therefore, AB is a 3x3 matrix

d.) No, the product of the two matrices are not guaranteed to be commutive with each other. For example;

$$AB = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} b_1 & b_2 & b_3 \\ b_4 & b_5 & b_6 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} a_1 b_1 + a_2 b_4 & a_1 b_2 + a_2 b_5 & a_1 b_3 + a_2 b_6 + a_3 \\ a_4 b_1 + a_5 b_5 & a_4 b_2 + a_5 b_5 & a_4 b_3 + a_5 b_6 + a_6 \\ 0 & 0 & 1 \end{bmatrix}$$

Thus, $AB \neq BA$

Problem 3. A 2D linear filter $h(x,y)$ is called *separable* if it can be decomposed into the convolution of two 1D filters: $h(x,y) = h_1(x) * h_2(y)$, where * represents convolution. For example, consider the box filter:

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} = \frac{1}{3}\begin{bmatrix}1\\ 1\\ 1\end{bmatrix} * \frac{1}{3}[1 \quad 1 \quad 1]$$

a) Show that a familiar edge-detection filter is separable by considering $\begin{bmatrix}1\\ 0\\ -1\end{bmatrix}$ and $[1 \quad 2 \quad 1]$.

b) The following is a rough approximation to a 1D Gaussian filter: $g = \frac{1}{16}[1 \quad 4 \quad 6 \quad 4 \quad 1]$. Assume, temporarily, that 2D Gaussian filters are separable. Construct a 2D Gaussian filter using the filter *g* and its transpose, g^T.

c) Show, analytically, that 2D Gaussian filters are separable.

a.)

$$\begin{bmatrix}1 & 0 & -1\\ 2 & 0 & -2\\ 1 & 0 & -1\end{bmatrix} \Rightarrow \begin{bmatrix}1\\ 2\\ 1\end{bmatrix} * [1 \quad 0 \quad -1]$$

b.)

$$\frac{1}{16}\begin{bmatrix}1\\ 4\\ 6\\ 4\\ 1\end{bmatrix} * \frac{1}{16}[1 \quad 4 \quad 6 \quad 4 \quad 1] \Rightarrow \frac{1}{16}\begin{bmatrix}1 & 4 & 6 & 4 & 1\\ 4 & 16 & 24 & 16 & 4\\ 6 & 24 & 36 & 24 & 6\\ 4 & 16 & 24 & 16 & 4\\ 1 & 4 & 6 & 4 & 1\end{bmatrix}$$

c.) Gaussian filter in 2D

$$G_\sigma(x,y) = \frac{1}{2\pi\sigma^2} e^{\frac{-x^2+y^2}{2\sigma^2}} \Rightarrow \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-x^2}{2\sigma^2}} \times \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-y^2}{2\sigma^2}}$$

Thus, $G_\sigma(x,y) = G_\sigma(x) \times G_\sigma(y)$

Problem 4. (For 5554 students, this problem is required. For 4554 students, this problem is optional and can be submitted for extra credit.)

Let $G_\sigma(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-x^2}{2\sigma^2}}$ represent the one-dimensional Gaussian function.

a) Show, analytically, that it is possible to approximate $\frac{\partial^2 G_\sigma}{\partial x^2}$ by the difference of two Gaussian functions. (Hint: compare $\frac{\partial^2 G_\sigma}{\partial x^2}$ with $\frac{\partial G_\sigma}{\partial \sigma}$.)

b) Create plots (using `matplotlib.pyplot.plot` or a similar tool) to illustrate your results from part (a), and paste the plots into your answer sheet. Your plots should show that $\frac{\partial^2 G_\sigma}{\partial x^2}$ closely resembles the difference of two Gaussian functions, based on some appropriate values for σ that you select.

Problem 5. Write a Python/OpenCV program that will allow you to compare Sobel edge detection with Canny edge detection. Name your script `P5.py`. Paste this script into your answer sheet, and also submit the source file.

Your program should do the following:

- Load an image in grayscale format. The image is provided to you in file `boat.png`, and you should hard-code this file name into your program.
- For each pixel in the input image, apply the horizontal and vertical Sobel operators and compute the magnitude of the intensity gradient. (For this part, your code must access pixel values directly, possibly using `for` loops. Yes, I know that OpenCV has a built-in `cv2.Sobel` function, but you cannot use it for this problem. However, you could use it during debugging to check your implementation.)
- Scale the gradient magnitude values to the range [0, 255] to simplify viewing. Write the resulting image to a file named `outputP5sobel.png`. Paste this output image into your answer sheet, and also submit the image file.
- Separately, perform Canny edge detection on the input image. For this case, you are allowed to use OpenCV functions such as `cv2.Canny`. Try to select parameters (for sigma, etc.) that find edges similar to your Sobel result. Write your Canny result to a file named `outputP6canny.png`. Paste this output image into your answer sheet, and also submit the image file.

Notes:

- For this problem, except for the Canny part, do not use any OpenCV functions other than for loading and saving image files.
- Your output images should be the same size (rows and columns) as the input image. You may use any technique to assign pixel values near the image border.

```
import numpy as np
import matplotlib.pyplot as plt
import cv2

def sobel_edge_detection(img, sobel_filter):
    image_row, image_col = img.shape
    kernel_row, kernel_col = sobel_filter.shape
    res_x = np.zeros(img.shape)
    res_y = np.zeros(img.shape)
    res = np.zeros(img.shape)

    sobel_filter_x = sobel_filter

    sobel_filter_y = sobel_filter.transpose()

    # print(sobel_filter_x)
    # print(sobel_filter_y)

    pad_height = int((kernel_row - 1) / 2)
    pad_width = int((kernel_col - 1) / 2)

    padded_image = np.zeros((image_row + (2 * pad_height), image_col + (2 * pa
d_width)))

    padded_image[pad_height:padded_image.shape[0] - pad_height, pad_width:padd
ed_image.shape[1] - pad_width] = img


    for row in range(image_row):
        for col in range(image_col):
            res_x[row, col] = np.sum(sobel_filter_x * padded_image[row:row + k
ernel_row, col:col + kernel_col])

    for row in range(image_row):
        for col in range(image_col):
            res_y[row, col] = np.sum(sobel_filter_y * padded_image[row:row + k
ernel_row, col:col + kernel_col])


    gradient_magnitude = np.sqrt( np.square(res_x) + np.square(res_y) )
    gradient_magnitude *= 255/ gradient_magnitude.max()
    #print(gradient_magnitude)
    res = np.array(gradient_magnitude)
    plt.figure("After Sobel Edge Detection")
    plt.imshow(res, cmap = 'Greys_r')
    #cv2.imshow('Sobel Operator Result', gradient_magnitude)
    plt.imsave('outputP5sobel.png', res, cmap = 'Greys_r')


#Load an image in grayscale
```

```
img = cv2.imread('boat.png', 0)
plt.figure("Grayscale Image")
plt.imshow(img, cmap = 'Greys_r')
#cv2.imshow('Grayscale Image',img)


sobel_filter = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])

sobel_edge_detection(img, sobel_filter)

#The source I got the following code from to make a more accurate
# canny edge detection result https://www.pyimagesearch.com/2015/04/06/zero-
parameter-automatic-canny-edge-detection-with-python-and-opencv/

# compute the median of the single channel pixel intensities
sigma = 0.33
v = np.median(img)
# apply automatic Canny edge detection using the computed median
lower = int(max(0, (1.0 - sigma) * v))
upper = int(min(255, (1.0 + sigma) * v))
edges = cv2.Canny(img, lower, upper)

plt.figure("After Canny Edge Detection")
plt.imshow(edges, cmap = 'Greys_r')
#cv2.imshow('Sobel Operator Result', gradient_magnitude)
plt.imsave('outputP5canny.png', edges, cmap = 'Greys_r')

plt.show()
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Canny Edge Detection

Sobel Edge Detection

Problem 6. Write a Python/OpenCV program that will perform *affine image warping*. (You may want to refer again to Problem 2, where you investigated 2D affine transformations.) Name your script `P6.py`. Paste this script into your answer sheet, and also submit the source file.

As part of your solution, write your own function named `affine_warp` that accepts an arbitrary image as an input parameter, along with an arbitrary affine transformation matrix `A` as introduced in Problem 2. Your function must compute and return an output image that results from the warping procedure. For example, an input image is shown at the left of the figure below. The image in the middle shows the result after clockwise rotation by a few degrees about the image center. The image at the right shows rotation followed by translation. (In this figure, the red and blue axes are not part of the images, but are included to illustrate the transformations.)

Your program should do the following:

- Load an image in grayscale format. The image is provided to you in file `boat.png`, and you should hard-code this file name into your program.
- Create an affine transformation matrix `A` that will produce a result very similar to the rightmost image in the figure above. Use `numpy.save()` to store this matrix in file `outputP6A.npy`. Submit this file with your solutions.
- Invoke your function `affine_warp` using the input image and matrix `A`. This function must initialize its output image so that all pixel values are 0. Then for each pixel in the *output* image, your function should determine the appropriate intensity value by mapping that pixel location (through the inverse affine transformation) to the corresponding location in the *input* image. For the case that your transformed point lies outside the boundaries of the input image, do not change anything in the output image.
- Write the image that was computed by `affine_warp` to a file named `outputP6.png`. Paste this output image into your answer sheet, and also submit the image file.

Notes:

- For this problem, do not use any OpenCV functions other than for loading and saving image files. Do not use any NumPy functions except for file I/O and basic math and trigonometry.
- Your output image should be the same size (rows and columns) as the input image.
- You are free to decide the coordinate reference frame for your implementation. A common choice for the origin is the upper left pixel of the image. Another possible choice is the center of the image (as shown in the figure above). In a few implementations, the lower left pixel of the image is used as the origin. In your answer sheet, carefully explain the coordinate system(s) that you have chosen.

- It may be helpful to look at reference material, such as https://docs.opencv.org/3.4/d4/d61/tutorial_warp_affine.html. However, unlike most online tutorials, your program must calculate all of the affine transformation parameters directly.

```
import numpy as np
import matplotlib.pyplot as plt
import cv2
import math

def affine_warp(A, input_image):
    #takes an image and matrices and applies it.
    x_min = 0
    y_min = 0
    x_max = input_image.shape[0]
    y_max = input_image.shape[1]

    res_image = np.zeros((x_max, y_max), dtype= "uint8")

    for y_counter in range(0, y_max):
        for x_counter in range(0, x_max):
            curr_pixel = [x_counter,y_counter,1]

            curr_pixel = np.dot(A, curr_pixel)

            # print(curr_pixel)

            if curr_pixel[0] > x_max - 1 or curr_pixel[1] > y_max - 1 or x_min
 > curr_pixel[0] or y_min > curr_pixel[1]:
                next
            else:
                res_image[x_counter][y_counter] = input_image[int(curr_pixel[0
])][int(curr_pixel[1])]

    return res_image


#Load an image in grayscale
img = cv2.imread('boat.png', 0)

rotate_A = np.array([[ math.cos( math.radians(-10) ), -
(math.sin( math.radians(-10) )), 0 ], [ math.sin( math.radians(-
10) ), math.cos( math.radians(-10) ), 0 ], [ 0, 0, 1 ]])
translate_A = np.array([ [1, 0, -90], [0, 1, 120], [0, 0, 1] ])

multiplied_matrices = np.dot(rotate_A, translate_A)
```

```
inverse_transform_matrix = np.linalg.inv(multiplied_matrices)

plt.figure("Grayscale Image")
plt.imshow(img, cmap = 'Greys_r')

plt.figure("Transformed Image")

transformed_img = affine_warp(inverse_transform_matrix, img)
plt.imshow(transformed_img, cmap='Greys_r')
plt.imsave('outputP6.png', transformed_img, cmap = 'Greys_r')
np.save("outputP6A.npy", inverse_transform_matrix)

plt.show()
```

I decided to go with the default rotation about the top left pixel in the corner. Choose this origin for rotation made it easier for me to see the angle rotate down the page before I applied the translation matrix to shift it into the right position.