CS376: Computer Vision: Assignment 2 Due: Oct. 7th, 11:59 PM

**Format for writeup:** You may use any tool for preparing the assignment write up that you like, as long as it is organized and clear, and figures are embedded in an easy to find way alongside your descriptive text. **Submission:** See the end of this document for submission instructions.

**Assignment questions:** Please see Piazza for questions and discussion from the class.

# Short answer problems [20 pts]

1. Compare the effects of 1) Dilation + Erosion against 2) Erosion + Dilation. Do they have the same effects? Why?

The results of the matlab functions imopen (erosion then dilation) and imclose (dilation then erosion) are different. The order matters because of the way that the first morphological operation alters the edges of the objects in the images. When erosion is performed first, small objects are removed while large objects are only made thinner, and then the dilation returns the thinned objects to normal but can’t recreate the objects that no longer exist. When dilation is performed first, small gaps/holes are shrunk or completely filled in, and then the erosion thins the object and expands the holes that still exist—it can’t recreate the holes that no longer exist.

1. List two examples of regular texture and two examples of near-regular texture.

Regular texture examples: plaid pattern, checkered pattern

Near regular texture examples: brick wall, tiled floor.

1. What are the cases where optical flow is not well-defined? Please given two concrete examples.

Optical flow is not well defined in cases where there aren’t moving objects. E.g. footage of a solid color wall with no foreground objects, a clear sky, still water with no reflection.

1. What are the advantages of RANSAC when compared with Hough Transform?

RANSAC vs Hough Transform generally manifests in a space/time complexity trade-off. RANSAC doesn’t require an accumulation array, so it can be more space efficient. Also, RANSAC only needs to iterate through each data point once per fitting attempt, whereas Hough Transform needs to iterate through a set number of possibilities to give votes to for every data point. This means that in cases where the fit is more obvious, RANSAC could have a better runtime because it won’t need many attempts. Where fits are less obvious however, more attempts will be necessary and will result in a worse runtime and possibly less accuracy than Hough Transform.

# Circle Detection (50 points)

Implement two circle detectors (one based on Hough Transformation and another based on RANSAC) that takes an input image and a fixed (known) radius, and returns the centers of any detected circles of about that size.

Include two functions with the following form:

[centers] = detectCirclesHT(im, radius)

[centers] = detectCirclesRANSAC(im, radius)

where ‘im‘ is the input image, ‘radius‘ specifies the size of circle we are looking for. Your detector should not exploit the gradient direction. The output centers is an N x 2 matrix in which each row lists the x,y position of a detected circle’s center. Write whatever helper functions are useful.

Then experiment with the basic framework, and in your writeup analyze the following:

* (10 pts) Explain your implementation in concise steps (English, not code).

First, use a built in MATLAB function to get the edges of the original image. The edges will be stored in a binary matrix in which 1’s represent points belonging to an edge. Create a matrix of zeros to store votes for each point on the image. Pad this votes matrix with (radius – 1) pixels above, below, left, and right so that when giving votes, we don’t have to check for out of bounds errors.

Get the coordinates of every edge point and then iterate through them. For each edge point, use a helper function that uses the Pythagorean theorem to get the coordinates of all points that are of a given distance from a location. Use that helper function with the distance set to the given radius. Take the returned coordinates and add the radius to both their row and column values to get the corresponding coordinates in the padded votes matrix. Go to these coordinates in the padded votes matrix and give each of them a vote by incrementing their values by 1.

After iterating through all the edge points, determine the minimum number of votes for a point to be considered the center of a circle, and return the coordinates of all points in the padded votes matrix with a number of votes at or above the minimum (exclude the padding). This minimum will be a certain percentage of the highest number of votes. Subtract (radius – 1) from the row and column values of each of the coordinates to get the coordinates in the unpadded image. To eliminate false positives/repeat-circles, iterate through these centers and find the distance from them to all the other centers. If the distance between the two is less than the diameter of the circle we are searching for, only keep the one with more votes.

(10 pts) Demonstrate the functions applied to the provided images ‘coins.jpg‘ and ‘planets.jpg‘ and one image of your choosing. Display the images with detected circle(s), labeling the figure with the radius. Note: you only need to select one reasonable radius and display all detected circles (i.e., those with highest votes) under that radius. You are not required to consider circles with a center off the image.A picture containing background pattern

Description automatically generatedBackground pattern

Description automatically generated with medium confidence

*•*

Graphical user interface

Description automatically generatedA picture containing text

Description automatically generatedGraphical user interface, application

Description automatically generated with medium confidenceGraphical user interface

Description automatically generated

(5 pts) For Hough Transform, explain how your implementation post-processes the accumulator array to determine automatically how many circles are present.

*•*

My implementation finds the maximum number of votes and establishes a vote threshold that is a certain percentage of that maximum. All points that have an amount of votes greater than or equal to that threshold are considered to be the center of a circle. Since swirly textures and dense edges tend to cause false positives and these false positives often overlap each other, my implementation goes through all the center points to find overlaps and only keeps the one with the highest vote count.

* (5 pts) For RANSAC, explain how you implement circle fitting.

Get an outer bounding radius by adding 2 to the given radius. Get an inner bound by subtracting 2 from the given.

Calculate the circumference of a circle with the given radius and multiply it by some value of choice between 0 and 1. This is our threshold number of inliers.

Set a successive failure limit of choice.

Get all edge points in the image. Record the number of edge points.

Pick a sample size that is a square with dimensions that are some multiple of the given radius. Start with the sample in the top left corner. Each sample will overlap the previous sample by 50% to account for circles that may exist in multiple samples.

Randomly select a pixel in the sample. Iterate through each edge point, calculating its distance from the selected pixel. If its distance is under the outer bounding radii, we record the point as one to remove if the randomly selected pixel is determined to be the center of a circle. If the distance is also greater than the inner bounding radii, increment the number of inliers by 1 and store the edge point in an “inliers” array for that pixel. After performing this on every edge point, this pixel and its inliers are our current best fit. Find the inliers for another randomly selected pixel. If it has more inliers than the current best fit, the new pixel and its inliers becomes the new best fit. If it doesn’t, ignore the new fit and increment successive failures.

Repeat this process until the successive failure limit is reached. If the current best fit model’s inlier count exceeds the threshold inliers, the best fit is determined to be a circle and all points inside the circle are removed from the total set of edgepoints.

If the successive failure limit is reached and the current best fit model does not have an inlier count exceeding the threshold, we have found all circles in this sample. Repeat this process for all samples.

* (5 pts) For one of the images, display and briefly comment on the Hough space accumulator array.

A picture containing background pattern

Description automatically generated

In this image, higher votes are shown with brighter pixels. The centers of the soccerballs in the images generally correspond to the brightest spots in this accumulator. The soccerballs in the lower half of the image have patterns that introduce more edges that contribute votes to other locations, causing a more even distribution of votes in the lower half and therefore less accuracy in locating the centers of the soccerballs.

(5 pts) For one of the images, demonstrate and explain the impact of the vote space quantization (bin size). In other words, alter the bin size and compare and contrast with a brief explanation why what happened makes sense.

*•*

(10 pts) For one of the images, plot the progress of the RANSAC as the number of tries increase. The x axis of the plot should be the number of tries, and the y axis should be the number of inliers that the best model produces.

*•*

Chart, histogram

Description automatically generatedThis is the plot of tries on with on the soccerballs image:

Useful Matlab functions: ‘hold on‘; ‘plot‘, ‘fspecial‘, ‘conv2‘, ‘im2double‘, ‘sin‘, ‘cos‘, ‘axis equal‘; ‘edge‘, ‘impixelinfo‘; ‘viscircles‘

# Image segmentation with k-means [30 pts]

For this problem you will write code to segment an image into regions using k-means clustering to group pixels.

(15 pts) Given an h x w x 3 matrix ‘Im‘, where h and w are the height and width of the image, apply k-means clustering to associate pixels with clusters. Return ‘labelIm‘, an *h w* matrix of integers indicating the cluster membership (e.g., from 1 to *k*) for each pixel. Please use the following form:

*×*

*•*

function [labelIm] = clusterPixels(Im, k)

* + (10 pts) Detect cluster boundary pixels from ‘labelIm‘.

function [boundaryIm] = boundaryPixels(labelIm)

(5 pts) Please test both functions on the provided images ‘gumballs.jpg‘, ‘snake.jpg‘, and ‘twins.jpg‘ and one other image of your choosing, and then displays the results.

*•*

# Submission instructions:

Create a single **zip** file so submit on Canvas that includes

* + Your well-commented code, including the files and functions named as specified above.
  + A **PDF** writeup of your results with embedded figures where relevant. Please do not include any saved matrices or images etc. within your zip file.