COMP 558 Assignment 1

Prepared by Prof. Kaleem Siddiqi
Posted: Thursday, Sept 23, 2021
Due: Thursday, Oct. 7, 2021 (by midnight, 11:59pm)

Introduction

This assignment covers material up to Lecture 6. The questions concern basic image processing using convolution, and some of core ideas of edge detection, and RANSAC.

Please use the mycourses discussion board forum for assignment 1, for any assignment related questions. This will help us reach the entire class when we respond. However, follow appropriate protocol, e.g., do not reveal the answer to a particular question or ask if your proposed solution is correct. Save these kinds of questions for one-on-one interactions during office hours. Do not post pictures of your solutions because these could give ideas to the rest of the class. *You are free to discuss the general aspects of questions with each other, but not to the point where you are revealing answers or sharing solutions. This is an individual assignment and the solutions that you submit must reflect your own work and must be written by you.* You are not permitted to submit code or text that you have copied from one another or from any third party sources, including internet sites. By submitting this assignment you agree to abide by these rules.

Instructions

Submit a single zipped file **A1.zip** to your mycourses Assignment 1 folder. The zip file must contain:

- A single PDF file with figures and text for each question. Please do not spend time with fancy typesetting. Just make sure that your explanations are clear and that the figures and their captions are easy to interpret and that you answer each part of each question.
- The Matlab code that you wrote so that we can run it if necessary. Make sure that you name the files as specified, in case the TAs wish to run your code using their own script.
- Any images that you used to test your code, in case we need to run your code to reproduce your results.

In order to receive full points, your solution code must be properly commented, and you must provide a *clear and concise* description of your computed results in the PDF. Note that the TAs have limited time and will spend at most 20-30 minutes grading each submission.

Late assignment policy: Late assignments will be accepted up to only 3 days late and will be penalized by 10% of the total marks, per day late. For example, an assignment that is submitted 2 days late and that receives a grade of 90/100 would then receive an 18 point penalty. Submitting one minute after midnight on the due date means it is 1 day late (not a fraction of a day), and same for the second day.

Tips

- To examine the RGB values in an image, use the figure tab *Tools→Datatips*. This will allow you to click on pixels and see their values.

- To speed up computation when testing code, resize the image (Matlab `imresize`).
- Pay attention to the types of your data. You might need to use logical type (boolean), uint8, and double. Matlab functions can be finicky and you may find you are spending much of your debugging time on type errors. Another common error is to swap the x and y indices. If your image is a 2D matrix, then you will index it with image(y, x) since matrices index row first and column second. Also, since sometimes we will want to treat the centre of the matrix as the location (0l,0), you might need to apply a shift in the (x, y) coordinates for some of the steps below.
- Make sure that you have the Computer Vision Toolbox and Image Processing Toolbox installed.
- When you are working with "intensity" it is fine to convert an RGB image to a grayscale one, e.g., using `rgb2gray`
- In Matlab think about how to vectorize computations and avoid loops if possible, `e.g.` $C = A.*B$ creates a matrix C which is the element-wise multiplication of two matrices A, B that have compatible sizes.

Question 1 Marr-Hildreth [20 points]

a) Consider a scalar function $f(x, y)$ of two variables x and y, where x and y are variables that are associated with two *orthogonal* directions. The Laplacian of f is given by $\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$, where we assume that f is at least twice differentiable. Show that the Laplacian is rotation invariant, i.e., the value of the expression does not depend on the actual choice of the orthogonal directions.

[*Hint: Consider an arbitrary orientation θ, with r a variable representing position on a line along that orientation. Then compute Δf using calculus and the chain rule, but using this line as one of the chosen coordinate directions.* To do this properly you need to introduce a change of coordinates.]

b) Edge detection in 2D is in general computationally expensive because a priori we do not know the local orientation of an edge. Hence, one has to look for zero-crossings of a second directional derivative, taken in multiple local directions. This is in fact how the mammalian visual system implements a form of edge detection in particular layers of the visual cortex. In their paper on edge detection Marr and Hildreth get around this complexity by associating edge locations with zero-crossings of the non-directional (as in rotation invariant) operator, the Laplacian. They claim that this operation picks out the direction which has highest slope of the zero-crossing (if one were to take directional derivatives that is). However, they make an important assumption about the local intensity profile. *What is that assumption? Explain briefly how, if that assumption holds, their argument works.*

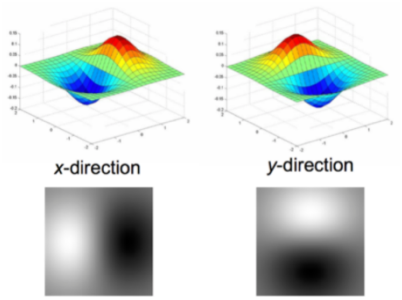
c) Given your answer in part b) when would you expect the Marr-Hildreth edge detection strategy to give good results and when you would expect it to fail? You can draw on your intuition from a natural image of a complex scene, which you could present and use to defend your argument. *You don't have to implement the strategy, you just have to provide your reasoning.*

Question 2 Edge detection Using Oriented Operators [40 points]

Write a script `Q2_Edgedetection.m`, using Matlab, to answer the following questions.

a) In class we've seen 1D Gaussian's and their derivatives. Let σ be an integer variable that is associated with the standard deviation of a 1D Gaussian. Create a 2D array with dimensions

$(2\,\sigma + 1)$ x $(2\,\sigma + 1)$. The centre of the array represents the (x, y) position (0,0). Now populate the entries at each row in the array with the value of the second derivatives of a 1D Gaussian in the x direction, with standard deviation σ. The 1D Gaussian should have its peak value in the middle of the row, and the values in each row should be identical. The original Gaussian should also be normalized to have area equal to 1 underneath it, prior to the computation of the second derivative. *Show an image of this filter and show the colormap using the* `colorbar` *command. Also visualize the filter values as a 2D height function, using Matlab's surface plotting functions* (e.g., `surf`, `surfc`, `mesh`, etc.). As an example, if you had a 2D Gaussian and you computed its partial derivatives in the x and y directions, a visualization of the results would look something like the following. (Ofcourse, the filter you are designing in this question is not the same thing.)



b) Now modify the filter above, by multiplying its entries by a normalized 1D Gaussian in the y direction, with standard deviation σ_y, with $\sigma_y > \sigma$. This should have the effect of tapering the filter values in the y direction. *Show an image of this modified filter and show the colormap using the* `colorbar` *command. If you like you can also visualize it using Matlab's surface plotting functions* (`surf`, `surfc`, `mesh`, etc.).

c) You will now write a routine to create a rotated version of the 2D filter in part b). The goal is to rotate it by a small amount θ. An effective way of doing this is just to rotate the coordinates. In other words, for the value at a coordinate (x', y'), in the destination 2D filter, where these coordinates represent a rotation of (x, y) $by\ \theta$ about the origin, we simply copy the value from the source 2D filter, at location (x, y). *Show images of this modified filter for different* $\theta = 0, 45, 90\ and\ 135\ degrees$*, including the colormap using the* `colorbar` *command. Also visualize the rotated filters as 2D height functions, as you did for part b).*

d) We have provided an image of skyscrapers in Manhattan, viewed under perspective projection. Filter this image with your filter from c), but using the 4 rotated copies of it (0,45,90 and 135 degrees). For each copy, find the pixels at which there is a zero crossing in either the x or y or diagonal directions. Adjust your choice of σ so that you get acceptable results. Create a binary image that indicates where these zero crossing points are. Note: you will actually have 4 sets of

zero-crossings, one for each orientation of the filter so you can think of a clever way of presenting these results together. One idea could be to show each set of zero-crossings in a different colour and then use the *Matlab functions such as* `imfuse` *to create a composite image. Alternatively, you can use different colours for each filter orientation. You just have to explain your visualization.*

For this part you have some flexibility in how you choose to display your results. You might need to tune your parameters σ, σ_y. Do not use `for loops` to find the zero crossings. Instead, use vectorization. Hint: You want to compare neighboring pixels of the filtered image and see where the values have different sign, namely if their product is negative. To avoid the annoyances of image boundaries, use Matlab's `circshift` operation to compare an image to a shifted version of itself. Note that logical operations can also be vectorized.

e) Repeat (d) but this time use a Laplacian of a Gaussian filter with $\sigma = 6$ and width 40. To construct this filter you can directly use the Matlab function `fspecial`. *Briefly discuss how your results in e) compare with the results in d), using also your insights from question 1.*

Question 3 Line finding using RANSAC [40 points]

For this question, you should use the Manhattan skyscraper image. You can also use an additional image for illustrative purposes if you like, but make sure that it contains a certain distribution of long linear structures. Don't forget to submit this image as well.

a) Your task is to implement the RANSAC method to find prominent lines in your input image. The pseudo-code for the RANSAC algorithm is in the slides and lecture notes provided in lecture 6, and it is presented in slightly simplified form here. In this simplified form, initially we are not using least squares to refine a fit, and we are not finding the `best' model.

```
Repeat M times{
      Randomly sample an edge element a list of N edge elements;
          Examine all remaining (N-1) edge elements and count
          the number C that are within some threshold distance τ
          from the line associated with the edge element, and
          whose orientation is within Δθ degrees of the orienta-
          tion of the edge element.(This is the 'consensus set'
          for that line model.);

          If C is sufficiently large (greater than Cmin), then
          use the average location of all these edge elements,
          and the average orientation, to produce a new line
          model. Store this line model. Remove the edge elements
          in the present consensus set from further considera-
          tion, i.e., mark them as 'used'.;
      CONTINUE;
}
```

You are given a script `makeEdgeList.m` (starter code) that reads in an image and generates a list of image edges. For your convenience it generates a list of 4-tuples, namely $(x_i, y_i, cos\theta_i, sin\theta_i)$, where $(cos\theta_i, sin\theta_i)$ points in the gradient direction. This code uses Matlab's Canny edge detector and Matlab's image gradient estimator.

Write and submit a script `Q3_Ransac.m` *that does the following.* It first reads an image and then computes the edges, using the script above. It then implements the pseudocode above, to return a set of line models, given a choice of values for $\tau, \Delta\theta$ and `Cmin` and the number of times M you chose to repeat the loop.

When you are done you will have a certain number of line models. Superimpose these line models by drawing the associated lines on the original image in a chosen colour. They should coincide with linear features in the image. By playing with `Cmin` you should be able to control the number of line segments found.

Present and discuss your results and the effect of the parameter choices.

Do not use any public domain implementations of the RANSAC algorithm since we want you to write your own!

b) [*Attempt this part only once you have a solution to part a), since this part carries only a little weight*]. Modify your solution in part a) slightly as follows. Rather than using the average position and orientation to return a line model, produce one via a least squares fit to the (x, y) data points in the consensus set. To implement least squares fitting you can use built in Matlab functions for eigenvector/eigenvalue computation. The difference now will be that your line models will involve actual fits to the data points, and each line model will come with an error of fit. Discuss your results.

Get started early! Good luck, and have fun!