Assignment 2 : Canny Edge & Harris Corner Detector

Submission Deadline: Sunday Nov 3, 2019 (11:59 PM)

Deliverables: **Project Report** in **BMVC format** and **code** in Python/Matlab/whateverprogramminglanguageyouknow.

All script/code files should be saved in a folder named 'Code', and a **report** needs to be submitted in **doc/pdf** format. A zip file containing both the code and report should be named as '**yourrollno_Ass2.zip**'. **Failure to adhere to these instructions will result in reduced marks.**

You are strongly encouraged to code in Python or Matlab to save time, as they already include high-level images for image processing!

Objective: Your goal is to implement a **Canny edge detector**, and a **Harris corner detector**, from ***first principles***. **Solutions including library functions (e.g., *cv2.canny*) will be summarily rejected.** *However, you are free to use library functions to complete constituent tasks* (original codes are nevertheless entitled to receive higher scores). You will be required to run and explain the working of both detectors during your evaluation.

You may refer to https://iitmcvg.github.io/summer_school/Session3/ for information that could be useful for completing the assignment.

Specifics:

1) **Implement a Canny Edge Detector (65 Marks).** You could look at class notes, and pointers on the web including this Wikipedia article. This process would entail **three phases**:

Phase 1: Compute smoothed gradients:

a) Load an image, convert it to float format, and extract its luminance as a 2D array. Alternatively, you may also convert the input image from color to gray-scale if required.

b) Find the x and y components Fx and Fy of the image gradient after smoothing with a Gaussian (for the Gaussian, you can use $\sigma = 1$). There are two ways to go about doing this: either (A) smooth with a Gaussian using a convolution, followed by computation of the gradient; or (B) convolve with the x and y derivatives of the Gaussian.

c) At each pixel, compute the edge strength F (gradient magnitude), and the edge orientation $D = \text{atan}(Fy/Fx)$. **Include the gradient magnitude and direction images in your report.**

Phase 2: Non-maximal Suppression

Create a "thinned edge image" $I[y, x]$ as follows:

a) For each pixel, find the direction D^* in $(0, \pi/4, \pi/2, 3\pi/4)$ that is closest to the orientation D at that pixel.

b) If the edge strength $F[y, x]$ is smaller than at least one of its neighbors along the direction D^*, set $I[y, x]$ to zero, otherwise, set $I[y, x]$ to $F[y, x]$. **Note:** Make a copy of the edge strength array before thinning, and perform comparisons on the copy, so that you are not writing to the same array that you are making comparisons on.

After thinning, your "thinned edge image" should not have thick edges any more (so edges should not be more than 1 pixel wide). **Include the thinned edge output in your report.**

Phase 3: Hysteresis thresholding:

a) Assume two thresholds: T_low, and T_high (they can be manually set or determined automatically).

b) Mark pixels as "definitely not edge" if less than T_low.

c) Mark pixels as "strong edge" if greater than T_high.

d) Mark pixels as "weak edge" if within [T_low, T_high].

e) Strong pixels are definitely part of the edge. Whether weak pixels should be included needs to be examined.

f) Only include weak pixels that are connected in a chain to a strong pixel. How to do this?

Visit pixels in chains starting from the strong pixels. For each strong pixel, recursively visit the weak pixels that are in the 8 connected neighborhood around the strong pixel, and label those also as strong (and as edge). Label as "not edge" any weak pixels that are not visited by this process. **Hint:** This is really a connected components algorithm, which can be solved by depth first search. Make sure to not revisit pixels when performing your depth first search!

Please run your edge detector on images in the **Data** folder, and include your intermediate plus edge-image outputs for each image in the Report. Experiment with the choice of thresholds T_low and T_high to obtain an edge image that corresponds to human intuition for what are edges. **You should find** that any strong edge forms a connected (linked) chain of pixels in the output.

2) **Implement a Corner Detector (35 Marks)**

Implement a Harris Corner Detector from first principles based on material presented in Class, and available on the Web. This involves the following steps:

A) **Filtered gradient:** Compute *x* and *y* gradients *Fx* and *Fy*, the same as in the Canny edge detector.

B) **Find corners:** For each pixel (*x*, *y*), look in a window of size $2m+1 \times 2m+1$ around the pixel (you can use $m = 4$). Accumulate over this window the covariance matrix *C*, which contains the average of the products of *x* and *y* gradients:

$$C = 1/(2m+1)^2 \sum_u \sum_v \begin{bmatrix} F_x^2 & F_x F_y \\ F_x F_y & F_y^2 \end{bmatrix} = \begin{bmatrix} \langle F_x^2 \rangle & \langle F_x F_y \rangle \\ \langle F_x F_y \rangle & \langle F_y^2 \rangle \end{bmatrix}$$

Here (*u*, *v*) are the coordinates within the window: $u = -m, ..., m$, and $v = -m, ..., m$, the brackets (< and >) denote a dot product operation, and the gradients *Fx* and *Fy* on the right hand side of the above equation are read from the locations $(x + u, y + v)$.

Compute the smaller eigenvalue, *e*, of *C*, which is the "corner response" (note that the original Harris corner detector paper uses a different and faster formula of e = Determinant(C)-k(Trace(C))^2, where k is a small constant such as 0.04, so you can alternatively use that formulation instead if you like). In Python, see the command ***numpy.linalg.eig***, which computes the eigenvalues. In MATLAB, use the ***eig*** command. Save all pixels at which the corner response ***e*** is greater than a corner threshold ***T*** into a list ***L***. Note that different images may require different *T* values.

C) **Non-maximum suppression:** Sort *L* in decreasing order of the corner response *e*. See ***numpy.sort*** and ***numpy.argsort***, or the Python built-in function ***sorted***. For each point *p*, remove all points in the 8-connected neighborhood of *p* that occur later in the list *L*.

Please run your corner detector on the 'Data' folder images, and include the results in your report. Experiment with the choice of thresholds to obtain a reasonable number of corners, between say 10 and 100.

What the report should contain:

(1) Your assumptions and methodology written in the form of an **Algorithm** (see https://www.youtube.com/watch?v=l7Z7tvCkQrg for guidance on how to write an Algorithm on Latex) for the two tasks.

(2) All the intermediate and final result images.

(3) Your observations and takeaways from the assignment.