Assignment 3
Harris Corner Detector & Optical Flow

Computer Vision 1
University of Amsterdam

Due 23:59 pm, Oct 5th, 2021 (Amsterdam time)

General guidelines

Your source code and report must be handed in together in a zip file (`ID1_ID2_ID3.zip`) before the deadline by submitting it to the Canvas Lab 3 Assignment. For full credit, make sure your report follows these guidelines:

- Include an introduction and a conclusion to your report.
- The maximum number of pages is 10 (single-column, including tables and figures). Please express your thoughts concisely. The number of words does not necessarily correlate with how well you understand the concepts.
- Answer all given questions (in green boxes). Briefly describe what you implemented. Blue boxes are there to give you hints to answer questions.
- Try to understand the problem as much as you can. When answering a question, give evidence (qualitative and/or quantitative results, references to papers, etc.) to support your arguments.
- Analyze your results and discuss them, e.g. why algorithm A works better than algorithm B in a certain problem.
- Tables and figures must be accompanied by a brief description. Do not forget to add a number, a title, and if applicable name and unit of variables in a table, name and unit of axes and legends in a figure.

Late submissions are not allowed. Assignments that are submitted after the strict deadline will not be graded. In case of submission conflicts, TAs' system clock is taken as reference. We strongly recommend submitting well in advance, to avoid last minute system failure issues.
Plagiarism note: Keep in mind that plagiarism (submitted materials which are not your work) is a serious crime and any misconduct shall be punished with the university regulations.

Contents

1 Harris Corner Detector (45pts)

In this section, a derivation of the Harris Corner Detector [1] is presented.

Given a shift $(\Delta x, \Delta y)$ at a point (x, y), the auto-correlation function is defined as:

$$c(\Delta x, \Delta y) = \sum_{(x,y)\in W(x,y)} w(x,y)(I(x+\Delta x, y+\Delta y) - I(x,y))^2\,, \tag{1}$$

where $W(x,y)$ is a window centered at point (x,y) and $w(x,y)$ is a Gaussian function. For simplicity, from now on, $\sum_{(x,y)\in W(x,y)}$ will be referred to as $\sum_W$.

Approximating the shifted function by the first-order Taylor expansion we get:

$$\begin{aligned} I(x+\Delta x, y+\Delta y) &\approx I(x,y) + I_x(x,y)\Delta x + I_y(x,y)\Delta y & (2)\\ &= I(x,y) + [I_x(x,y)\ I_y(x,y)] \begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix}, & (3)\end{aligned}$$

where I_x and I_y are partial derivatives of $I(x,y)$. The first gradients can be approximated by:

$$\begin{aligned} I_x &= \frac{\partial I}{\partial x} \approx I * G_x, \quad G_x = (-1,0,1) & (4)\\ I_y &= \frac{\partial I}{\partial y} \approx I * G_y, \quad G_y = (-1,0,1)^T & (5)\end{aligned}$$

Note that using the kernel $(-1,1)$ to approximate the gradients is also correct. The auto-correlation function can now be written as:

$$\begin{aligned} c(\Delta x, \Delta y) &= \sum_W w(x,y)(I(x+\Delta x, y+\Delta y) - I(x,y))^2 & (6)\\ &\approx \sum_W w(x,y)([I_x(x,y)\ I_y(x,y)] \begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix})^2 & (7)\\ &= [\Delta x\ \Delta y] Q(x,y) \begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix}, & (8)\end{aligned}$$

where $Q(x,y)$ is given by:

$$\begin{aligned} Q(x,y) &= \sum_W w(x,y) \begin{bmatrix} I_x(x,y)^2 & I_x(x,y)I_y(x,y)\\ I_x(x,y)I_y(x,y) & I_y(x,y)^2\end{bmatrix} & (9)\\ &= \begin{bmatrix} \sum_W I_x(x,y)^2 * w(x,y) & \sum_W I_x(x,y)I_y(x,y) * w(x,y)\\ \sum_W I_x(x,y)I_y(x,y) * w(x,y) & \sum_W I_y(x,y)^2 * w(x,y)\end{bmatrix} & (10)\\ &= \begin{bmatrix} A & B\\ B & C\end{bmatrix}. & (11)\end{aligned}$$

The *cornerness* $H(x, y)$ is defined by the two eigenvalues of $Q(x, y)$, e.g. λ_1 and λ_2:

$$\begin{aligned} H &= \lambda_1\lambda_2 - 0.04(\lambda_1 + \lambda_2)^2 && (12) \\ &= det(Q) - 0.04(trace(Q))^2 && (13) \\ &= (AC - B^2) - 0.04(A + C)^2. && (14) \end{aligned}$$

In this section, you are going to implement Equation 12 to calculate H and use it to detect the corners in an image.

Hint

For that purpose, you need to compute the elements of $\mathbf{Q}$, i.e. A, B and C. To do that, you need to calculate I_x, which is the smoothed derivative of the image. That can be obtained by convolving the first order Gaussian derivative, G_d, with the image I along the x-direction. Then, $\mathbf{A}$ can be obtained by squaring I_x, and then convolving it with a Gaussian, G. Similarly, $\mathbf{B}$ and $\mathbf{C}$ can be obtained. For example, to get $\mathbf{C}$, you need to convolve the image with G_d along the y-direction (to obtain I_y), raise it to the square, then convolve it with G.

Hint

The corner points are the local maxima of $\mathbf{H}$. Therefore, you should check for every point in H, (1) if it is greater than all its neighbours (in an $n \times n$ window centered around this point) and (2) if it is greater than the user-defined threshold. If both conditions are met, then the point is labeled as a corner point.

Question - 1 (35-*pts*)

1. Create a function to implement the Harris Corner Detector. Your function should return matrix H, the indices of rows of the detected corner points **r**, and the indices of columns of those points **c**, where the first corner is given by $(r[0], c[0])$. Name your script as **harris_corner_detector.py**.

2. Implement another function that plots three figures: The computed image derivatives I_x and I_y, and the original image with the corner points plotted on it. Show your results on example images **toy/0001.jpg** and **doll/0200.jpeg** in your report separately. Remember to experiment with different threshold values to see the impact on which corners are found.

3. Is the algorithm rotation-invariant? How about your implementation? Rotate **toy/0001.jpg** image 45 and 90 degrees and run the Harris Corner Detector algorithm on the rotated images. Explain your answer and support it with your observations.

Note: You are allowed to use `scipy.signal.convolve2d` to perform convolution, and `scipy.ndimage.gaussian_filter` to obtain your image derivatives. Include a demo function to run your code.

Question - 2 (10-*pts*)

Now you have seen the cornerness definition of Harris on Equation 12. Another relevant definition of cornerness is defined by Shi and Tomasi [2], after the original definition of Harris. Check their algorithm and answer the following questions:

1. How do they define cornerness? Write down their definition using the notations of Equation 12.

2. Does the Shi-Tomasi Corner Detector satisfy the following properties: translation invariance, rotation invariance, scale invariance? Explain your reasoning.

3. In the following scenarios, what could be the relative cornerness values assigned by Shi and Tomasi? Explain your reasoning.

   (a) Both eigenvalues are near 0.

   (b) One eigenvalue is big and the other is near zero.

   (c) Both eigenvalues are big.

2 Optical Flow - Lucas-Kanade Algorithm (35pts)

Optical flow is the apparent motion of image pixels or regions from one frame to the next, which results from moving objects in the image or from camera motion. Underlying optical flow is typically an assumption of *brightness constancy*. That is the image values (brightness, color, etc) remain constant over time, though their 2D position in the image may change. Algorithms for estimating optical flow exploit this assumption in various ways to compute a velocity field that describes the horizontal and vertical motion of every pixel in the image. For a 2D+t dimensional case a voxel at location (x, y, t) with intensity $I(x, y, t)$ will have moved by δ_x, δ_y and δ_t between the two image frames, and the following image constraint equation can be given:

$$I(x, y, t) = I(x + \delta_x, y + \delta_y, t + \delta_t). \tag{15}$$

Assuming the movement to be small, the image constraint at I(x, y, t) can be extended using Taylor series, truncated to first-order terms:

$$I(x + \delta_x, y + \delta_y, t + \delta_t) = I(x, y, t) + \frac{\partial I}{\partial x}\delta_x + \frac{\partial I}{\partial y}\delta_y + \frac{\partial I}{\partial t}\delta_t \tag{16}$$

Since we assume changes in the image can purely be attributed to movement, we will get:

$$\frac{\partial I}{\partial x}\frac{\delta_x}{\delta_t} + \frac{\partial I}{\partial y}\frac{\delta_y}{\delta_t} + \frac{\partial I}{\partial t}\frac{\delta_t}{\delta_t} = 0 \tag{17}$$

or

$$I_x V_x + I_y V_y = -I_t, \tag{18}$$

where V_x and V_y are the x and y components of the velocity or optical flow of $I(x, y, t)$. Further, I_x, I_y and I_t are the derivatives of the image at (x, y, t) in the corresponding directions, which defines the main equation of optical flow.

Optical flow is difficult to compute for two main reasons. First, in image regions that are roughly homogeneous, the optical flow is ambiguous, because the brightness constancy assumption is satisfied by many different motions. Second, in real scenes, the assumption is violated at motion boundaries and by miscellaneous lighting, non-rigid motions, shadows, transparency, reflections, etc. To address the former, all optical flow methods make some sort of assumption about the spatial variation of the optical flow that is used to resolve the ambiguity. Those are just assumptions about the world which are approximate and consequently may lead to errors in the flow estimates. The latter problem can be addressed by making much richer but more complicated assumptions about the changing image brightness or, more commonly, using robust statistical methods which can deal with 'violations' of the brightness constancy assumption.

Lucas-Kanade Algorithm

We will be implementing the Lucas-Kanade method [3] for Optical Flow estimation. This method assumes that the optical flow is essentially constant in a local neighborhood of the pixel under consideration. Therefore, the main equation of the

optical flow can be assumed to hold for all pixels within a window centered at the pixel under consideration. Let's consider pixel p. Then, for all pixels around p, the local image flow vector (V_x, V_y) must satisfy:

$$\begin{array}{c} I_x(q_1)V_x + I_y(q_1)V_y = -I_t(q_1) \\ I_x(q_2)V_x + I_y(q_2)V_y = -I_t(q_2) \\ \vdots \\ I_x(q_n)V_x + I_y(q_n)V_y = -I_t(q_n), \end{array} \tag{19}$$

where q_1, q_2, $\ldots q_n$ are the pixels inside the window around p. $I_x(q_i)$, $I_y(q_i)$, $I_t(q_i)$ are the partial derivatives of the image I with respect to position x, y and time t, evaluated at the point q_i and at the current time.

These equations can be written in matrix to form $Av = b$, where

$$A = \begin{bmatrix} I_x(q_1) & I_y(q_1) \\ I_x(q_2) & I_y(q_2) \\ \vdots & \vdots \\ I_x(q_n) & I_y(q_n) \end{bmatrix}, v = \begin{bmatrix} V_x \\ V_y \end{bmatrix}, \text{ and } b = \begin{bmatrix} -I_t(q_1) \\ -I_t(q_2) \\ \vdots \\ -I_t(q_n) \end{bmatrix}. \tag{20}$$

This system has more equations than unknowns and thus it is usually over-determined. The Lucas-Kanade method obtains a compromise solution by the weighted-least-squares principle. Namely, it solves the 2×2 system as

$$A^T A v = A^T b \tag{21}$$

or

$$v = (A^T A)^{-1} A^T b. \tag{22}$$

Question - 1 (30-pts)

For this assignment, you will be given two pairs of images: *Car1.jpg*, *Car2.jpg*; and *Coke1.jpg*, *Coke2.jpg*. You should estimate the optical flow between these two pairs. That is, you will get optical flow for sphere images, and for synth images separately. Implement the Lucas-Kanade algorithm using the following steps. Name your script **lucas_kanade.py**.

1. Divide input images on non-overlapping regions, each region being 15×15.
2. For each region compute A, A^T and b. Then, estimate optical flow as given in Equation 22.
3. When you have estimation for optical flow (V_x, V_y) of each region, you should display the results. There is a **matplotlib** function `quiver` which plots a set of two-dimensional vectors as arrows on the screen. Try to figure out how to use this to show your optical flow results.

Note: You are allowed to use `scipy.signal.convolve2d` to perform convolution.
Include a demo function to run your code.

Hint

You can use regions that are 15×15 pixels that are non-overlapping. That is, if input images are 256×256, you should have an array of 17×17 optical flow vectors at the end of your procedure. As we consider 15×15 regions, your matrix $\mathbf{A}$ will have the following size 225×2, and the vector $\mathbf{b}$ will be 225×1.

Hint

Carefully read the documentation of **matplotlib**'s `quiver`. By default, the angles of the arrows are 45 degrees counter-clockwise from the horizontal axis. This means your arrows might point in the wrong direction! Also, play around with the arrow scaling.

Question - 2 (5-*pts*)

Now you have seen one of the optical flow estimation methods developed by Lucas and Kanade. There are several more methods in the literature. The Horn-Schunck method [4] is one of them. Check their method, compare it to Lucas-Kanade and answer the following questions:

1. At what scale do the algorithms operate; i.e local or global? Explain your answer.
2. How do the algorithms behave at flat regions?

3 Feature Tracking (20pts)

In this part of the assignment, you will implement a simple feature-tracking algorithm. The aim is to extract visual features, like corners, and track them over multiple frames.

Question - 1 (18-pts)

1. Implement a simple feature-tracking algorithm by following below steps. Name your script **tracking.py**.
   (a) Locate feature points on the first frame by using the Harris Corner Detector, that you implemented in section 1.
   (b) Track these points using the Lucas-Kanade algorithm for optical flow estimation, that you implemented in section 2.
2. Prepare a video for each sample image sequences. These videos should visualize the initial feature points and the optical flow. Test your implementation and prepare visualization videos for *doll* and *toy* samples.

Include a demo function to run your code.

Question - 2 (2-pts)

Why do we need feature tracking even though we can detect features for each and every frame?

Tips: required Python packages: Pillow, Numpy, Scipy, Matplotlib, Opencv. To install Opencv in an anaconda environment, this command is highly recommended: "conda install -c menpo opencv".

References

[1] C. G. Harris and M. Stephens, "A combined corner and edge detector.," in *Alvey vision conference*, vol. 15, pp. 10–5244, Citeseer, 1988.

[2] S. Jianbo and C. Tomasi, "Good features to track," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 593–600, 1994.

[3] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," 1981.

[4] B. K. Horn and B. G. Schunck, "Determining optical flow," in *Techniques and Applications of Image Understanding*, vol. 281, pp. 319–331, International Society for Optics and Photonics, 1981.