Computer Vision 1 - AI Assignment3
Harris Corner Detector and Optical Flow

February 22, 2016

As this assignment includes more engineering work and is also weighted with more points (20%), we decide to postpone the due time a little bit late as scheduled. But this should not affect the submission of assignment 4. Please arrange your time properly.

All the files should be zipped (with file name "{stu1_name}, {stu2_name}.zip") and sent to **computervision1.uva(at)gmail.com** before **06-03-2016** , **23.59** (Amsterdam Time).

1 Harris Corner Detector

In this section, a derivation of the Harris Corner Detector is presented:

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

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

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

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

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

where I_x and I_y are partial derivatives of $I(x,y)$. Therefore, the auto-correlation function can be written as:

$$
\begin{aligned}
c(x,y) &= \sum_W (I(u,v) - I(u+\Delta x, v+\Delta y))^2 && (4)\\
&\approx \sum_W ([I_x(u,v)\ I_y(u,v)] \begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix})^2 && (5)\\
&= [\Delta x \Delta y] Q(x,y) \begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix} && (6)
\end{aligned}
$$

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

$$
\begin{aligned}
Q(x,y) &= \sum_W \begin{bmatrix} I_x(x,y)^2 & I_x(x,y)I_y(x,y) \\ I_x(x,y)I_y(x,y) & I_y(x,y)^2 \end{bmatrix} && (7)\\
&= \begin{bmatrix} \sum_W I_x(x,y)^2 & \sum_W I_x(x,y)I_y(x,y) \\ \sum_W I_x(x,y)I_y(x,y) & \sum_W I_y(x,y)^2 \end{bmatrix} && (8)\\
&= \begin{bmatrix} A & B \\ B & C \end{bmatrix} && (9)
\end{aligned}
$$

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

$$
\begin{aligned}
H &= \lambda_1\lambda_2 - 0.04(\lambda_1+\lambda_2)^2 && (10)\\
&= det(Q) - 0.04(trace(Q))^2 && (11)\\
&= (AC - B^2) - 0.04(A+C)^2 && (12)
\end{aligned}
$$

In this section, you are going to implement the last equation to measure H and use it to get the corners in an image. For that purpose, you need to compute the elements of Q, i.e. A, B and C. To do that, you need to know that I_x is the smoothed derivative of the image, it can be obtained by convolving the first order Gaussian derivative, Gd , with the image I along the x-direction. Later, $A = \sum_W I_x(x,y)^2$, can be obtained by squaring I_x then convolving it with a Gaussian, G. Similarly, B and C can be obtained. For example, to get C, you need to convolve the image with Gd along the y-direction (to obtain I_y), raise it to the square, then convolve it with G. At this point, you can compute H. The corners points are the local maxima of H. Therefore, in your function you should check for every point in H, if it is greater than all its neighbours (in an $n \times n$ window centered around this point) and if it is greater than the user-defined threshold, then it is a corner point. Your function should return the H matrix, the rows of the detected corner points r, and the columns of those points c - so the first corner is given by $(r(1), c(1))$. Your function should also plot three figures, the computed image derivatives I_x and I_y, and the original image with the corner points plotted on it.

2 Introduction: Optical Flow

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:

$$I(x, y, t) = I(x + \delta_x, y + \delta_y, t + \delta_t). \tag{13}$$

Assuming the movement to be small, the image constraint at I(x, y, t) with Taylor series can be developed to get:

$$I(x + \delta_x, y + \delta_y, t + \delta_t) = I(x, y, t) + \frac{\partial I}{\partial x}\delta_x + \frac{\partial I}{\partial y}\delta_y + \frac{\partial I}{\partial t}\delta_t + h.o.t. \tag{14}$$

As we assume the image values remain constant over time, we will get:

$$\frac{\partial I}{\partial x}\frac{\delta_x}{\delta_t} + \frac{\partial I}{\partial y}\frac{\delta_y}{\delta_t} + \frac{\partial I}{\partial t}\frac{\delta_t}{\delta_t} = 0 \tag{15}$$

or

$$I_x V_x + I_y V_y = -I_t, \tag{16}$$

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

Optical flow is hard 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 changing 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; these are just assumptions about the world which will be 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.

3 Lucas-Kanade Algorithm

We will be implementing the Lucas-Kanade method 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. Lets consider pixel p then for all pixels around p the local image flow vector (V_x, V_y) must satisfy

$$\begin{aligned} I_x(q_1)V_x + I_y(q_1)V_y &= -I_t(q_1) \\ I_x(q_2)V_x + I_y(q_2)V_y &= -I_t(q_2) \\ &\vdots \\ I_x(q_n)V_x + I_y(q_n)V_y &= -I_t(q_n), \end{aligned} \tag{17}$$

where q_1, q_2, $\dots$ q_n are the pixels inside the window around p, and $I_x(q_i)$, $I_y(q_i)$, $I_t(q_i)$ 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 form $Av = b$, where

$$A = \begin{bmatrix} I_x(q_1) & I_y(q_1) \\ I_x(q_2) & I_y(q_2) \\ \vdots & \vdots \\ I_x(q_n) & I_y(q_n) \end{bmatrix}, v = \begin{bmatrix} V_x \\ V_y \end{bmatrix}, \text{ and } b = \begin{bmatrix} -I_t(q_1) \\ -I_t(q_2) \\ \vdots \\ -I_t(q_n) \end{bmatrix}. \tag{18}$$

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

$$A^T A v = A^T b \tag{19}$$

or

$$v = (A^T A)^{-1} A^T b. \tag{20}$$

For this assignment, you will be given two pairs of images: *synth1.pgm*, *synth2.pgm*; and *sphere1.ppm*, *sphere2.ppm*. You should estimate the optical flow between these two pairs, it means you will get optical flow for sphere images, and for synth images separately.

You can use regions that are 15×15 pixels, and 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 A will have the following size 225×2, and the vector b will be 225×1. The algorithm can be summarized as follows:

1. Divide input images on non-overlapping regions, each region being 15×15 pixels.
2. For each region compute A, A^T and b; then estimate optical flow as given in equation (20).
3. When you have estimation for optical flow (V_x, V_y) of each region, you should display the results. There is a matlab function `quiver` which plots a set of two-dimensional vectors as arrows on the screen. Try to figure out how to use this to plot your optical flow results.

4 Tracking

In this part you will implement a simple tracking algorithm as follows:

1. Locate feature points on the first frame by using Harris Corner Detector which you implemented in previous section. Then, track these points with Lucas-Kanade method for optical flow estimation.

2. Prepare a video for each sample image sequences. These videos should visualize the initial feature points and the flow.

Test your implementation and prepare visualization videos for *pingpong* and *person_toy* samples.

5 Deliverables

1. **[6 pts]** For section 1:

   - [3 *pts*] A demo function should return the H matrix, the rows of the detected corner points r, and the columns of those points c - so the first corner is given by $(r(1), c(1))$.
   - [3 *pts*] For visualization, the function should also plot three figures, the computed image derivatives I_x and I_y, and the original image with the corner points plotted on it. (Note: please show your results on example images "person_toy/00000001.jpg" and "pingpong/0000.jpeg" in your report separately.)

2. **[6 pts]** For section 3:

   - [3 *pts*] A demo function which runs the whole routine with all other functions you have implemented.
   - [3 *pts*] Visualizations of two optical flows for sphere and synth images should be submitted.

3. **[8 pts]** For section 4:

   - [3 *pts*] Demo functions which run the trackers (Harris Corner Detector and the combination with Lucas-Kanade algorithm) with all other supporting functions you have implemented.
   - [5 *pts*] Visualization videos of two implemented trackers for *pingpong* and *person_toy* should be submitted.