COMPUTER VISION

Practical Session 1:
HARRIS CORNER DETECTION

Objectives

Learn how to detect corners in an image and understand the importance of the different steps involved in corner detection using the Harris algorithm.

Introduction

In this practical session, we are going to implement Harris corner detection. This method is based on the image gradient. A zone of interest such as a corner or an edge corresponds to a large change in appearance, impacting the value of the gradient. Harris corner detection applies a mathematical approach to this property in order to detect corners.

For each pixel, the "cornerness" can be estimated using the eigenvalues of the autocorrelation matrix M, defined by:

$$M = \begin{pmatrix} \sum_W \langle {I_x}^2 \rangle & \sum_W \langle I_x I_y \rangle \\ \sum_W \langle I_x I_y \rangle & \sum_W \langle {I_y}^2 \rangle \end{pmatrix} \quad (1)$$

$$\langle {I_x}^2 \rangle = g \otimes {I_x}^2 \quad (2)$$

Where W is a 3x3 neighbourhood around the pixel, g a Gaussian filter and $\otimes$ the convolution operator.

The eigenvalues of M $(\lambda_1,\ \lambda_2)$ gives a classification of the image points. If λ_1 and λ_2 are small, the area is flat. If one of the eigenvalues is very small compared to the other, then the area corresponds to an edge. Finally, if λ_1 and λ_2 are both large and of similar magnitude, the area is a corner region.

Harris and Stefens suggested using the determinant and trace of M to detect corners, which removes the need to compute eigenvalues:

$$R = \det(M) - k \cdot \mathrm{tr}(M)^2 \quad (3)$$

Where k is a constant (with a typical value of 0.04), $\det(M) = \lambda_1 \lambda_2$ the determinant and $\mathrm{tr}(M) = \lambda_1 + \lambda_2$ the trace of a square matrix.

R depends only on M's eigenvalues. R is large for a corner, R is negative with a large magnitude for an edge and |R| is small for a flat region.

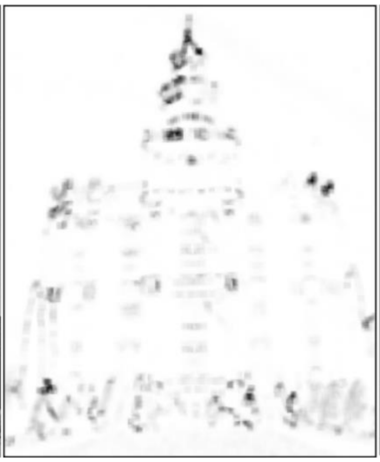
The implementation of the method is as follows:

- Compute the image gradient for each pixel in x and y directions: $I_x = \frac{\partial I}{\partial x}, I_y = \frac{\partial I}{\partial y}$
- Compute ${I_x}^2$, ${I_y}^2$ and $I_x I_y$.
- Smooth the square image derivative with a Gaussian filter: $\langle {I_x}^2 \rangle = g \otimes {I_x}^2$.
- For each pixel, compute the autocorrelation matrix M defined in equation (1).
- Finally, compute R using equation (3).

If the matrix R is directly used for corner detection, each zone of interest might be detected several times. Redundancies are generally corrected using a "non-maximal suppression" algorithm (NMS), which consists of setting a pixel to zero if there is a higher value in its neighbourhood. A threshold can be applied to improve the detection.

Example: (G.Brostow, UCL computer science, corner detection)

Original image



Harris detection: R matrix

Result after NMS

Practical session

Start with chessboard00.png

Part 1

Read the image, compute the image derivative Ix and Iy and apply the smoothing filter.

```
%% PART 1

% a. Read the image
I=imread('chessboard00.png');
% b. Compute the image derivative Ix and Iy
% c. Generate a Gaussian filter of size 9*9 and standard deviation 2
% d. Apply the Gaussian filter to smooth the images Ix*Ix, Iy*Iy and Ix*Iy
% e. Display results
```

Part 2

Compute E, the matrix that contains for each pixel the value of the smaller eigenvalue of M. Display the matrix E.

```
%% PART 2 - Compute Matrix E which contains for every point the value of the smaller
% eigenvalue of auto correlation matrix M.

% a. Compute E
% Initialize E. Then, for each pixel:
        % (1) build matrix M using a window of size 3*3
        % (2) Compute eigenvalues of the matrix
        % (3) save the smaller eigenvalue in E
% b. Display results
figure, imshow(mat2gray(E));
```

Part 3

Compute matrix R which contains for every pixel the result of equation (3). What is the difference with E? Use functions tic and toc to measure the time required for computing E and R.

```
%% PART 3 - Compute Matrix R which contains for every point the cornerness score
```

```
% a. Compute R
        % Initialize R. Then, for each pixel:
        % (1) build matrix M
        % (2) Compute the trace and the determinant of M
        % (3) save the result of equation 3 in R
% b. Display results
figure, imshow(mat2gray(R));
```

Part 4

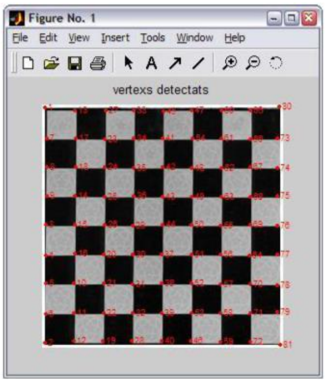
For E and R, select the 81 most salient points. Do you get the result that you expected?

```
%% PART 4 - Select for E and R the 81 most salient points.

% a. Write a function to obtain the 81 most salient points of E and R
% and their coordinates features(p_x, p_y)
% b. Display the selected points on top of the original image
figure; imshow(I); hold on; xlabel('Max 81 points');
for i=1:size(features,2),
        plot(features(i).p_x, features(i).p_y, 'r+');
end
hold off
```

Part 5

Apply a non-maximal suppression algorithm on E and R (choosing a neighbourhood of dimension 11*11), and display again the 81 most salient points. Is this result better than the previous one?

```
%% PART 5 - Build a function to carry out non-maximal suppression for E and R
% Select the 81 most salient points using a non-maximal suppression of 11×11 pixels.

% a. Apply non maximal suppression with a window of 11*11
% b. Get the 81 most salient points and their coordinates, in the same way as part 4
% c. Display the selected points on top of the original image
```



Try your code on the other chessboard images (chessboard0X.png) or on other images with corners that interest you. Comment and conclude.

Submission guidelines

Please submit your report with comments and illustrations as a pdf file named “Harris_YourName.pdf”. Please send you source code in a zip file containing your MatLab scripts and functions.