mplementing the Harris Corner Detection Algorithm

Bilal Ahmad

February 25, 2020

1 Introduction

I implemented elements of the Harris corner detection algorithm and applied it on two images: *CheckerBoard* and *Building1* The purpose of this exercise is to see how different parameters involved in implementing the Harris corner detection algorithm affect the final results.

The algorithm is implemented in three steps. The first step consists of applying a Gaussian filter to the image and calculating the gradient components. In the next step, an autocorrelation matrix is formed using the square and product of the gradient components. The eigenvalues calculated from the matrix are saved if they exceed a threshold indicating a potential corner. Finally, the third step filters out weaker corner detections existing in the neighborhood of stronger detections.

2 Harris Corner Detection Implementation

2.1 Apply a Gaussian Filter and Obtain the Gradient Components

A Gaussian filter is applied to the image to perform initial smoothing. Next, two convolutions are applied to the image using the Sobel operators G_x and G_y to obtain the gradient components. The Sobel operators, defined in Eq 1 and Eq 2, approximate the derivatives for the horizontal and vertical changes in the image.

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \tag{1}$$

$$G_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \tag{2}$$

The algorithm then calculates the square and product of the gradient components. The code used to implement these steps is shown in Listing 1.

Listing 1: MATLAB Code for Smoothing and Gradient Components

```
%% Smooth Image and Obtain Gradients
lanImage = imread(asImage);

% convert image to greyscale if necessary
if (size(lanImage, 3) ~= 1)
    lanImage = rgb2gray(lanImage);
end

% imgaussfilt performs a convolution with a 5x5 Gaussian kernel
larSmoothedImage = imgaussfilt(lanImage, arSigma);

% Compute the horizontal and vertical gradients
lanGradX = [-1, 0, 1; -1, 0, 1; -1, 0, 1];
lanGradY = lanGradX';
larGradImgX = conv2(larSmoothedImage, lanGradX, 'same');
larGradImgY = conv2(larSmoothedImage, lanGradY, 'same');

% Get the square and product of the components
larGradImgX_2 = larGradImgX.^2;
larGradImgY_2 = larGradImgY.^2;
lanGradImgXY = larGradImgX.*larGradImgY;
```

2.2 Calculating Corner Detections

Corners are detected by determining if both a horizontal and vertical edge exists within a window of pixels, Q. For each pixel, calculate the autocorrellation matrix, C, from all pixels within the neighborhood Q. The matrix C is defined by Eq 3. If the smaller eigenvalue calculated from C is larger than the threshold τ, save the eigenvalue and pixel that generated it as a corner detection. The MATLAB implementation for calculating the corner detections shown in Listing 2. MATLAB's *eig* function was used to calculate the eigenvalues for C.

$$C = \begin{bmatrix} \sum E_x^2 & \sum E_{xy} \\ \sum E_{xy} & \sum E_y^2 \end{bmatrix} \tag{3}$$

Listing 2: Obtaining Corner Detections

```
%% Detect All Potential Corners
% Matrix to hold the eigenvalues and coordinates
larL = [];

% Number of rows and columns
lnNumRows = size(larSmoothedImage,1);
lnNumCols = size(larSmoothedImage,2);

% Get eigenvalue for each point
for i=1+anWindow:lnNumRows-anWindow
    for j=1+anWindow:lnNumCols-anWindow
        lnSquaredX = ...
            sum(sum(larGradImgX_2(i-anWindow:i+anWindow,...
                j-anWindow:j+anWindow)));
        lnSquaredY = ...
            sum(sum(larGradImgY_2(i-anWindow:i+anWindow,...
                j-anWindow:j+anWindow)));
        lnSquaredProduct= ...
            sum(sum(lanGradImgXY(i-anWindow:i+anWindow,...
                j-anWindow:j+anWindow)));

        % Build matrix C
        larC =[lnSquaredX lnSquaredProduct;...
            lnSquaredProduct lnSquaredY];
        % Store eigenvalues in matrix L
        lnEig = min(eig(larC));
        if lnEig > arThresh
            larL = [larL; [lnEig i j]];
        end
    end
end
```

2.3 Filtering Weak Corner Detections

The final step of the algorithm is to filter out weaker corner detections that exist within the neighborhood of stronger corner detections. The corner detections are first sorted by the strength of their eigenvalues. Next, for each detection in descending order of strength, any weaker detection within the neighborhood Q is eliminated. The MATLAB implementation for eliminating the weaker corner detections can be seen in Listing 3

Listing 3: Eliminating Weak Corner Detections

```
%% Filter Weaker Adjacent Corner Detections
% sort the eigenvalues
larL = sortrows(larL, 'descend');

% Keep only the largest eigenvalues in each neighborhood
lnA = 1;
while (lnA ~= length(larL))
    lnB = lnA + 1;
    while (lnB ~= length(larL))
        if (abs(larL(lnA,2) - larL(lnB,2)) <= anWindow && ...
                abs(larL(lnA,3) - larL(lnB,3)) <= anWindow)
            larL(lnB,:) = []; % eliminate weaker detection
        else
            lnB = lnB + 1;
        end
    end
    lnA = lnA + 1;
end
```

3 Applying the Algorithm to Images

The Harris corner detection algorithm was applied to the images: *CheckerBoard* and *Building1*. Different neighborhood size, threshold, and sigma values were used in the applications to see how the parameters affected the algorithm. The resultant images can be seen in Fig 1, Fig 2, and Fig 3.

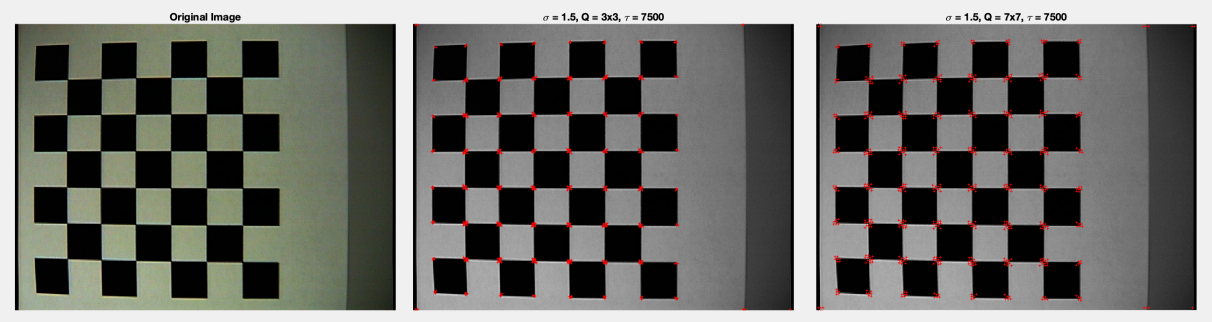

Figure 1: Applying Corner Detection with Different Neighborhood Sizes to *CheckerBoard*

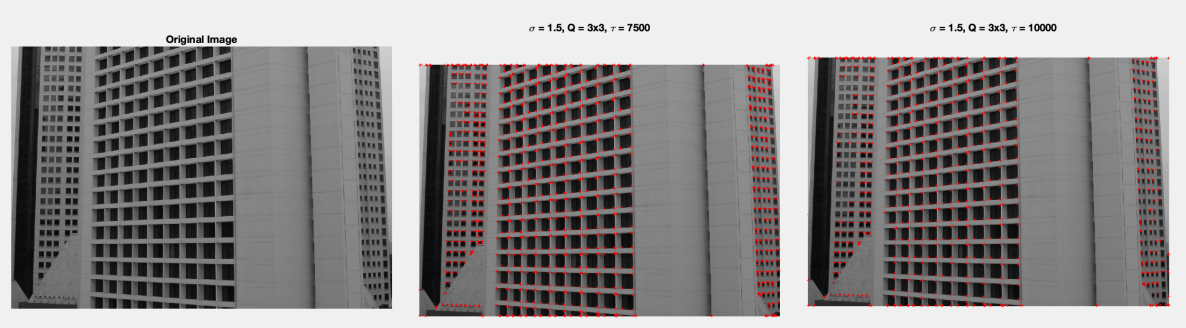

Figure 2: Applying Corner Detection with Different Threshold Values to *Building1*

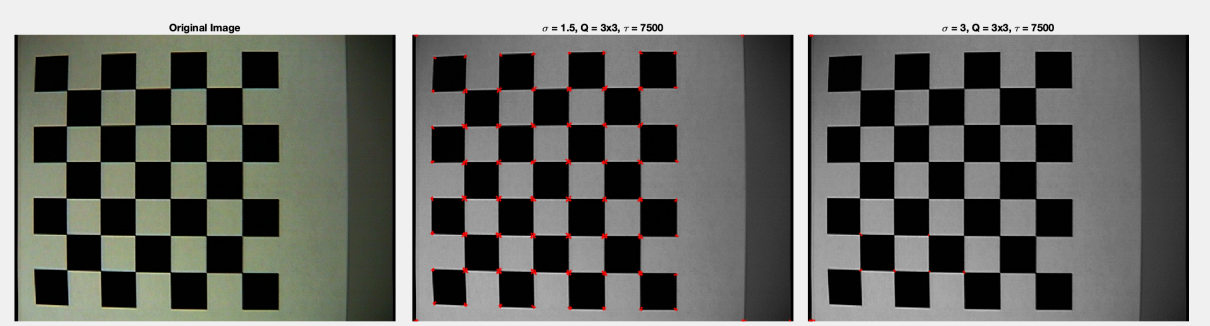

Figure 3: Applying Corner Detection with Different Sigma Values to *CheckerBoard*

4 Analysis

4.1 Adjusting Neighborhood Size Values

In Fig 1 different neighborhood sizes are used to calculate the autocorrelation matrix, C. Increasing the size of the neighborhood size spreads the detections further away from the actual corner. This is the expected result because using a larger neighborhood during the calculation of the autocorrellation matrix will incorporate the actual corner location for more pixels further away from the corner.

4.2 Adjusting Threshold Values

In Fig 2 different threshold values are used for the corner detection thresholding step of the Harris corner detection algorithm. Increasing the threshold value resulted in fewer image corner detections. This is the expected result because fewer eigenvalues calculated from

the autocorrellation matrix will meet the threshold requirement to get saved as a corner detection.

4.3 Adjusting Sigma Values

From Fig 3, you can see the results of using two different sigma values during the initial Gaussian filtering. There are fewer corners detected when using a larger sigma value. This is as expected since using a larger sigma value will smooth out and erase more edges in the image. This edge-erasure will make it difficult for the algorithm to detect corners after the initial smoothing.