CORNER DETECTION

NOVEMBER 9, 2021

TA's: Muhammed Zemzemoğlu and Mehmet Emin Mumcuoğlu

Name of Student: Moses Chuka Ebere
Student Number: 31491
Course: Computer Vision (EE 417)

TABLE OF CONTENTS

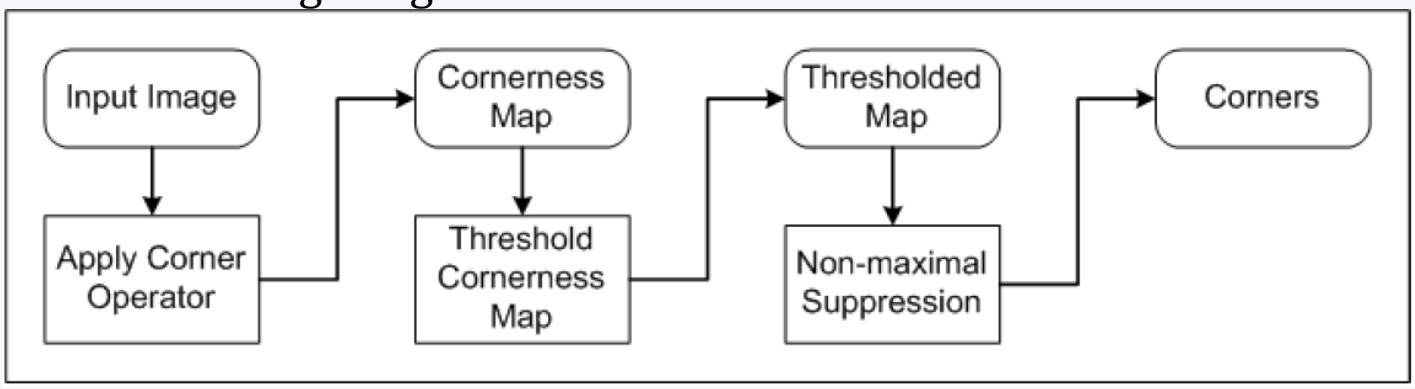
INTRODUCTION

Corner Detection

This is an image processing technique used to highlight points of intersection edges in images. Corner detection is widely researched because of its usefulness in a plethora of applications today like object detection, motion tracking, robot navigation, 3D reconstruction, etc. In the most trivial case, corner detection could be used to guide a robot along a predetermined path. Several researchers have come up with multiple approaches to developing corner detection algorithms. Som of these researchers include, but are not limited to, Chris Harris, Mike Stephens, Moravec, Kanade, Shi, Tomasi, Förstner, etc.

The corner detectors implemented in this lab make use of a second order moment matrix whose eigenvalues are compared with a threshold to determine whether the pixels qualify as corners, as in the case of the Kanade-Tomasi Algorithm. In the Harris algorithm, the same matrix is used; however, a parameter that approximates the minimum eigenvalue of the corner matrix is compared to the threshold.

The general procedure following during corner detection is outlined in the following image:



P.S. It is imperative that the input image be a smoothed one to avoid ending up with spurious corners.

EXPLANATION OF METHODS

Kanade-Tomasi Corner Detection

The inbuilt "imgradient" function is used to compute the x- and y-gradients of a smoothed image. Nested for-loops are used to scan the gradient images and create (2k+1)x(2k+1) windows of the x- and y-gradients. Within each window, image gradients are used to create the corner matrix using the following formula:

$$H = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix}$$

After this, within the same nested loops, the eigenvalues of the 2x2 H-matrix are found using MATLAB's inbuilt "eig" function. The minimum of these eigenvalues is now compared with the user-defined threshold such that all values above the threshold for qualify as corners. The pixel locations are taken to form a list. This will be used in plotting the corners in the final image.

In MATLAB, the following function was written for the Kanade-Tomasi corner detection using f. The function accepts an image and a threshold and returns the detected points and a smoothed image.

```
function [output1, output2] = lab4ktcorners(img, t)

% First of all, smoothen the image using Gaussian
filtering.
img = lab3gaussfilt(img);

% The row, column, and channels of the image are obtained
along with the cardinality of the image.
[r, c, ch] = size(img);
Card = r*c;

% This is added in case the image introduced is an RGB
image.
% It functions to convert it to a gray-scale image.
if (ch == 3)
    img = rgb2gray(img);
```

```
end

%Convert the image to double before performing any
%mathematical operation
I = double(img);

% Compute the image gradients
[Gx, Gy] = imgradientxy(I);
% Initialize the second order moment matrix and the list.
H = zeros(2,2); list = [];

% The window size for the gaussian filtering operation is
specified.
k = 2;

% Use nested for-loops to create a window for scanning the
image.
for i=(k+1):1:r-k
    for j=(k+1):1:c-k
        % Create a window of the image gradients (from -k
to +k)
        wpx = Gx(i-k:i+k, j-k:j+k);
        wpy = Gy(i-k:i+k, j-k:j+k);
        % Create the corner matrix, H.
        H(1,1) = sum(sum(wpx.*wpx));
        H(1,2) = sum(sum(wpx.*wpy));
        H(2,1) = H(1,2);
        H(2,2) = sum(sum(wpy.*wpy));
        % Find the eigenvalues of the corner matrix.
        L1 = eig(H);
        % Threshold the minimum eigenvalue of H.
        if min(L1) > t
            list = [list; [i,j]];
        end
    end
end
% Convert the resulting image to unsigned 8-bit image and
return the
% result and the list.
output1 = uint8(I);
output2 = list;
end
```

In the main script, the following code calls the Kanade-Tomasi function and applies it on the input image, before displaying the smoothed image with the detected corners plotted.

```
%% Kanade-Tomasi Corner Detection
% Read the image to be preprocessed
a = imread('blocks.png');
a1 = imread('ct.png');
a2 = imread('Monastry.bmp');
% The threshold is a user-defined variable to obtain the
corners.
thr = 5000000; th = 500000; th1 = 200000;

% We call the Kanade-Tomasi function and obtain a smoothed
image and a list
% of detected corners.
[x, pp] = lab4ktcorners_checker (a1, th1);
[x1, y1] = lab4ktcorners (a2, thr);
[x2,y2] = lab4ktcorners (a, th);

% We plot the detected corners on the smoothed image.
figure
subplot(1,3,1)
imshow(x)
hold on
plot(pp(:,2), pp(:,1), 'r*', 'Markersize', 1, 'Linewidth',
1)
title('Kanade-Tomasi Corner Detection')

subplot(1,3,2)
imshow(x1)
hold on
plot(y1(:,2), y1(:,1), 'r*', 'Markersize', 2, 'Linewidth',
1)
title('Kanade-Tomasi Corner Detection')

subplot(1,3,3)
imshow(x2)
hold on
```

```
plot(y2(:,2), y2(:,1), 'r*', 'Markersize', 2, 'Linewidth',
1)
title('Kanade-Tomasi Corner Detection')
```

Harris Corner Detection

The inbuilt "imgradient" function is used to compute the x- and y-gradients of a smoothed image. Nested for-loops are used to scan the gradient images and create (2k+1)x(2k+1) windows of the x- and y-gradients. Within each window, image gradients are used to create the corner matrix using the following formula:

$$H = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix}$$

After this, within the same nested loops, f is computed as follows:

$$f = det(H)/trace(H)$$

The value of f is now compared against a user-defined threshold such that all f's greater than the threshold qualify as corners. The pixel coordinates in the qualifying cases are indexed in a list and used to plot the corners in the final image.

In MATLAB, the following function was written for the Harris operator using f. The function accepts an image and a threshold and returns the detected points and a smoothed image.

```
function [output1, output2] = lab4Harriscorners(img, t)

% First of all, smoothen the image using Gaussian
filtering.
img = lab3gaussfilt(img);

% The row, column, and channels of the image are obtained
along with the cardinality of the image.
[r, c, ch] = size(img);
Card = r*c;
```

```
% This is added in case the image introduced is an RGB
image.
% It functions to convert it to a gray-scale image.
if (ch == 3)
    img = rgb2gray(img);
end

%Convert the image to double before performing any
%mathematical operation
I = double(img);

% Compute the image gradients
[Gx, Gy] = imgradientxy(I);
% Initialize the second order moment matrix and the list
with empty arrays.
H = zeros(2,2); list = [];

% The window size for the gaussian filtering operation is
specified.
k = 2;

% Use nested for-loops to create a window for scanning the
image.
for i=(k+1):1:r-k
    for j=(k+1):1:c-k
        % Create a window of the image gradients (from -k
to +k)
        wpx = Gx(i-k:i+k, j-k:j+k);
        wpy = Gy(i-k:i+k, j-k:j+k);
        % Create the corner matrix, H.
        H(1,1) = sum(sum(wpx.*wpx));
        H(1,2) = sum(sum(wpx.*wpy));
        H(2,1) = H(1,2);
        H(2,2) = sum(sum(wpy.*wpy));
        % f ~ the minimum eigenvalue of the corner matrix.
        f = det(H)/trace(H);
        % Threshold f to obtain the corners.
        if f > t
            list = [list; [i,j]];
        end
    end
end
```

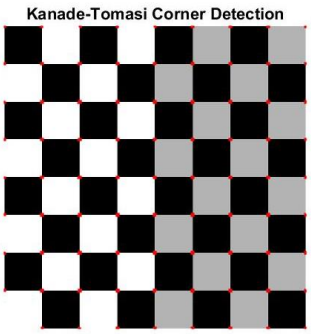
```
% Convert the resulting image to unsigned 8-bit image and
return the
% result and the list.
output1 = uint8(I);
output2 = list;
end
```

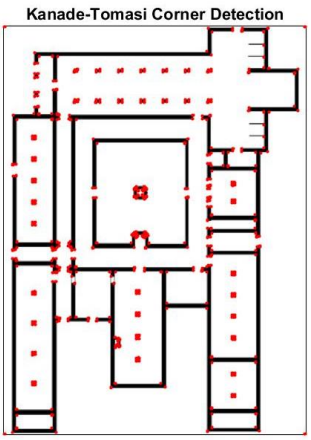
The following code calls the Harris function and applies it on the input image, before displaying the smoothed image with the detected corners plotted.

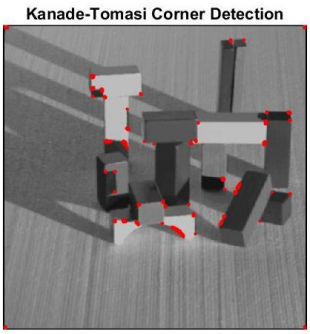
```
%% Harris Corner Detection
% Read the image to be preprocessed
b = imread('blocks.png');
b1 = imread('Monastry.bmp');
% The threshold is a user-defined variable to obtain the
corners.
thr1 = 400000; thr2 = 40000;

% We call the Harris function and obtain a smoothed image
and a list
% of detected corners.
[r, q] = lab4Harriscorners (b, thr2);
[r1, q1] = lab4Harriscorners (b1, thr1);

% We plot the detected corners on the smoothed image.
figure
subplot(1,2,1)
imshow(r1)
hold on
plot(q1(:,2), q1(:,1), 'r*', 'Markersize', 1, 'Linewidth',
1)
title('Harris Corner Detection')

subplot(1,2,2)
imshow(r)
hold on
plot(q(:,2), q(:,1), 'r*', 'Markersize', 1, 'Linewidth',
1)
title('Harris Corner Detection')
```

RESULTS

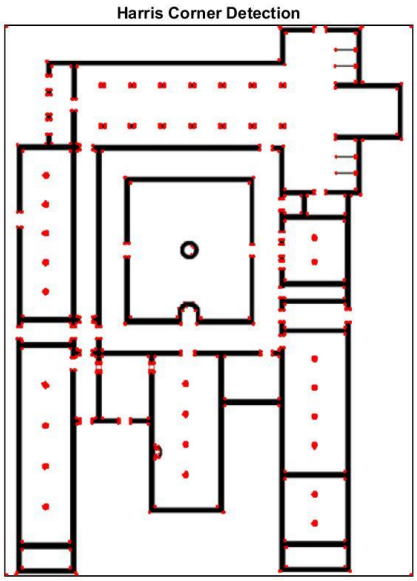
Kanade-Tomasi Algorithm

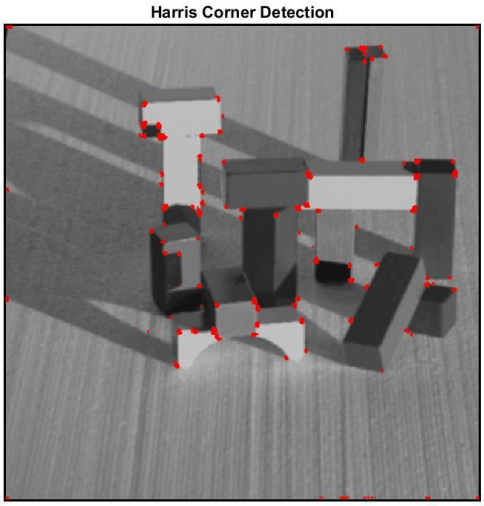






Harris Algorithm



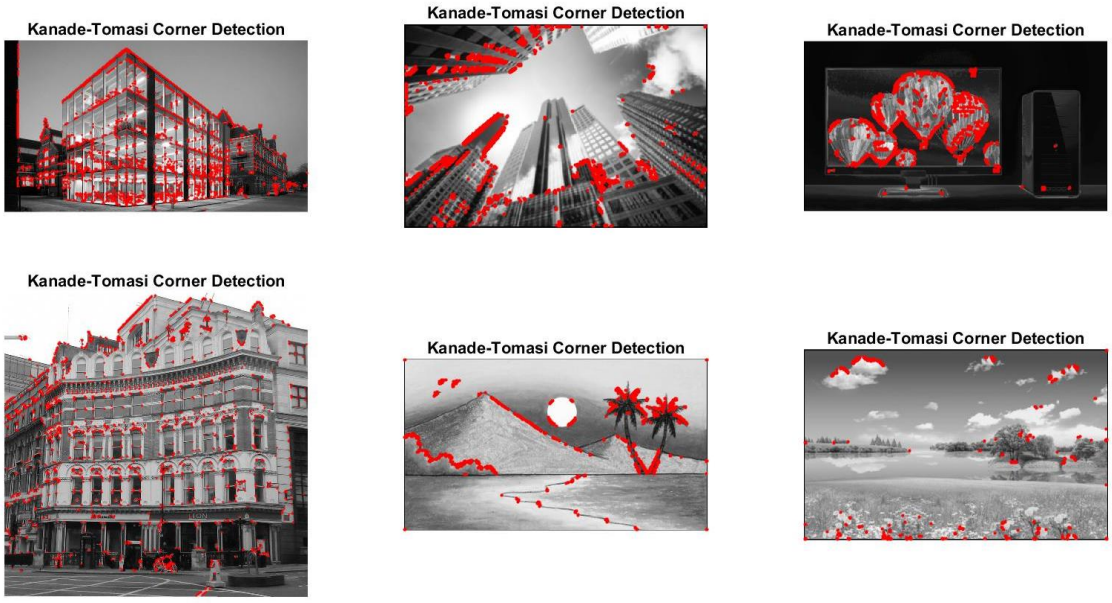


Notice: some points close to the bottom right corner are picked up. In the real sense, most of these don't qualify as corners.

MULTIPLE RESULTS

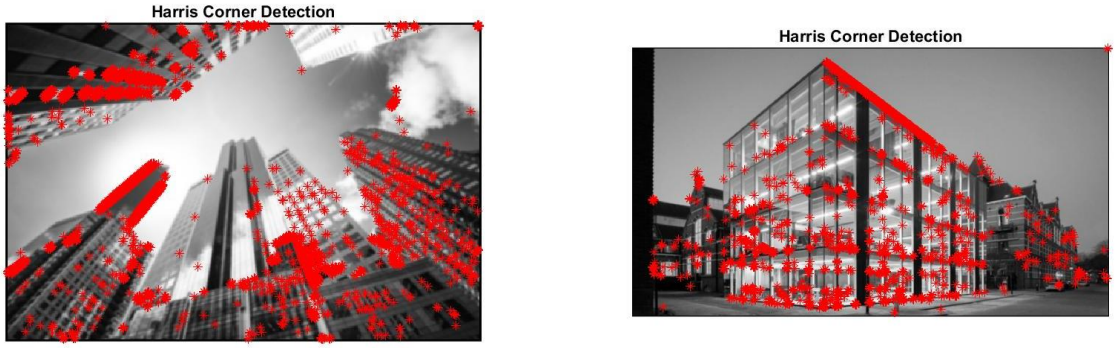
Kanade-Tomasi Algorithm

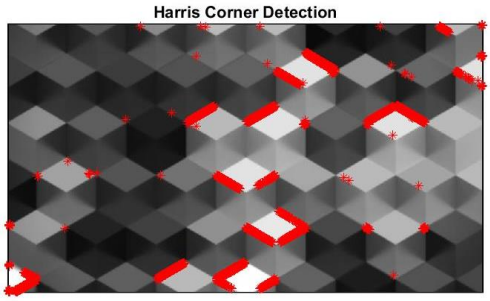
Let's apply the Kanade-Tomasi algorithm on RGB images:

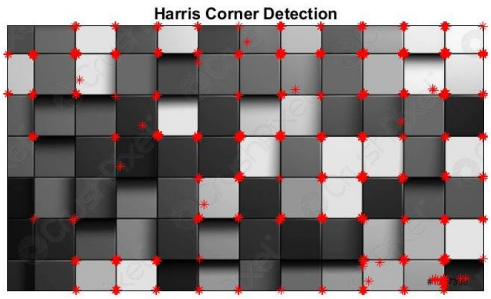


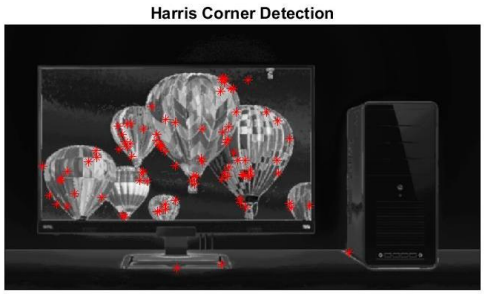
Harris Algorithm (with f)

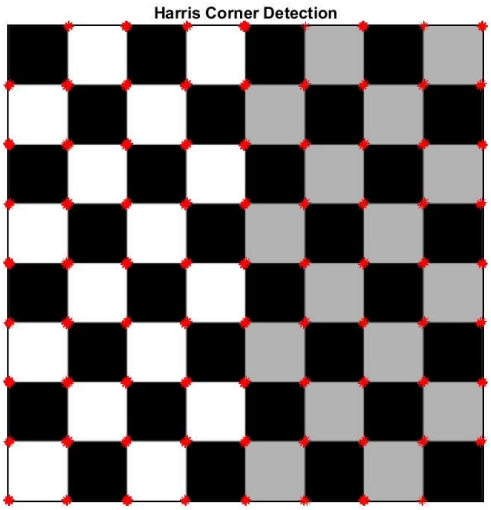
Applying this on some color images:




Harris Corner Detection


Harris Corner Detection


Harris Corner Detection


Harris Corner Detection

QUESTIONS

Harris Algorithm Using the Measure of Corner Response, R

Here, in place of f, a parameter, R, known as the measure of corner response, is used. R is defined as follows:

$$R = \det H - k(\text{trace}\, H)^2$$

Where: $\det H = \lambda_1 \lambda_2$, $\text{trace}\, H = \lambda_1 + \lambda_2$, and
(k – empirical constant, k = 0.04-0.06)

In MATLAB, the following function was written for the Harris operator using R. The function accepts an image and a threshold and returns the detected points and a smoothed image.

```
function [output1, output2] = lab4Harris_R_corners(img, t)

% First of all, smoothen the image using Gaussian
filtering.
img = lab3gaussfilt(img);

% The row, column, and channels of the image are obtained
along with the cardinality of the image.
[r, c, ch] = size(img);
Card = r*c;

% This is added in case the image introduced is an RGB
image.
```

```
% It functions to convert it to a gray-scale image.
if (ch == 3)
    img = rgb2gray(img);
end

%Convert the image to double before performing any
%mathematical operation
I = double(img);

% Compute the image gradients
[Gx, Gy] = imgradientxy(I);
% Initialize the second order moment matrix and the list
with empty arrays.
H = []; list = [];

% The window size for the gaussian filtering operation is
specified.
k = 2;
% The Empirical Constant for the Cornerness Function is
defined
kk = 0.05;

% Use nested for-loops to create a window for scanning the
image.
for i=(k+1):1:r-k
    for j=(k+1):1:c-k
        % Create a window of the image gradients (from -k
to +k)
        wpx = Gx(i-k:i+k, j-k:j+k);
        wpy = Gy(i-k:i+k, j-k:j+k);
        % Create the corner matrix, H.
        H(1,1) = sum(sum(wpx.*wpx));
        H(1,2) = sum(sum(wpx.*wpy));
        H(2,1) = H(1,2);
        H(2,2) = sum(sum(wpy.*wpy));
        % R is the cornerness function
        R = det(H) - kk*(trace(H))^2;
        % Threshold R to obtain the corners.
        if R > t
            list = [list; [i,j]];
        end
    end
```

```
end
% Convert the resulting image to unsigned 8-bit image and
return the
% result and the list.
output1 = uint8(I);
output2 = list;
end
```

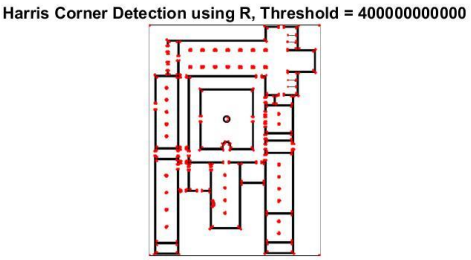
The following code calls the Harris (using R) function and applies it on the input image, before plotting the detected corners on the smoothed image.

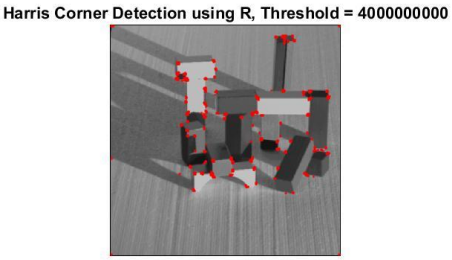
```
%% Harris Corner Detection using the Measure of Corner
Response
% Read the image to be preprocessed
c = imread('blocks.png');
c1 = imread('Monastry.bmp');
% The threshold is a user-defined variable to obtain the
corners.
thr3 = 1e+13; thr4 = 1e+10;

% We call the Harris function and obtain a smoothed image
and a list
% of detected corners.
[m, n] = lab4Harris_R_corners (c, thr4);
[m1, n1] = lab4Harris_R_corners (c1, thr3);

% We plot the detected corners on the smoothed image.
figure
subplot(1,2,1)
imshow(m1)
hold on
plot(n1(:,2), n1(:,1), 'r*')
title('Harris Corner Detection')

subplot(1,2,2)
imshow(m)
hold on
plot(n(:,2), n(:,1), 'r*')
```

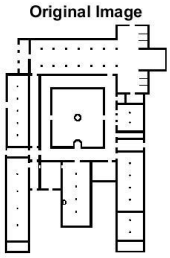
```
title('Harris Corner Detection')
```

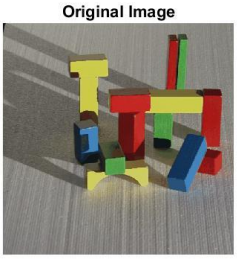
Results

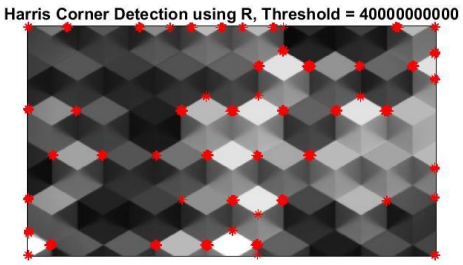
The result is as follows:

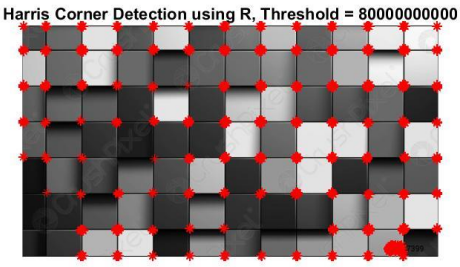


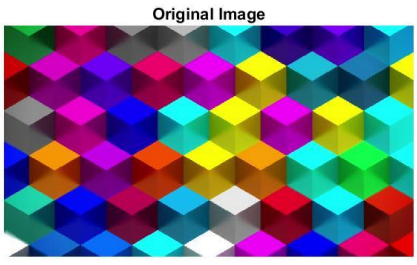


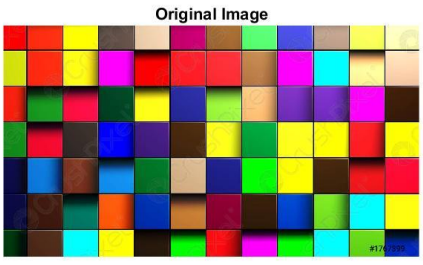












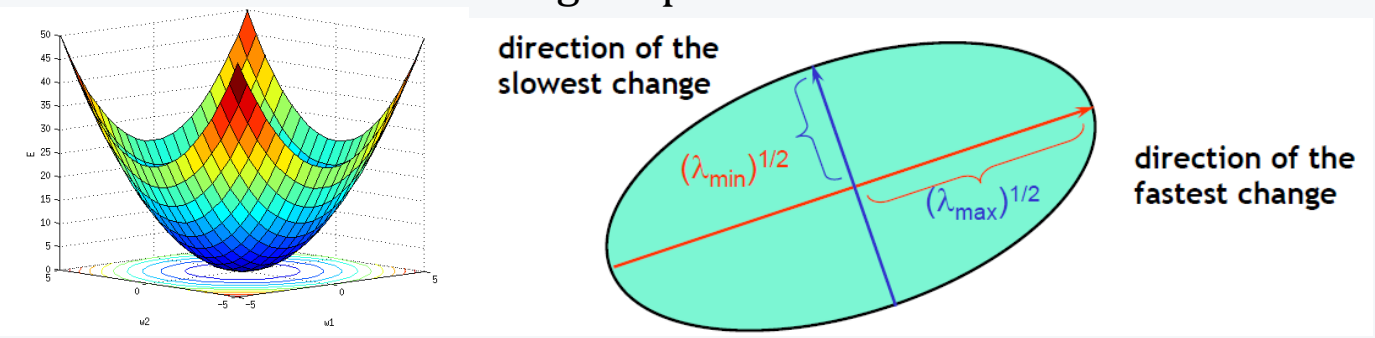
DISCUSSION

General Comments on Corner Detection

- The image must be smoothed before applying the algorithm to avoid obtaining **spurious corners.**
- The corner matrix is a positive semi-definite matrix.

Kanade-Tomasi Algorithm

- Since the H-matrix is 2x2 matrix, we end up with two eigenvalues.
- H-matrix is a symmetric matrix; hence, the eigenvalues are always real and non-negative.
- **Why do we choose the minimum eigenvalue as opposed to choosing the maximum one?**
- The error function of the gradients of an image in a region with a corner takes the following shape.



- As we can observe, it's in our best interest to heighten the response of our algorithm to corners by ensuring that we pinpoint area where the slightest move away result in massive variation in intensity. This can be achieved by thresholding the minimum eigenvalue since it represents the direction of slowest change.
- The threshold used is dependent both on the image and the corner detector used.

Harris Algorithm

- The Harris operator is a variant of the minimum eigenvalue of the corner matrix.

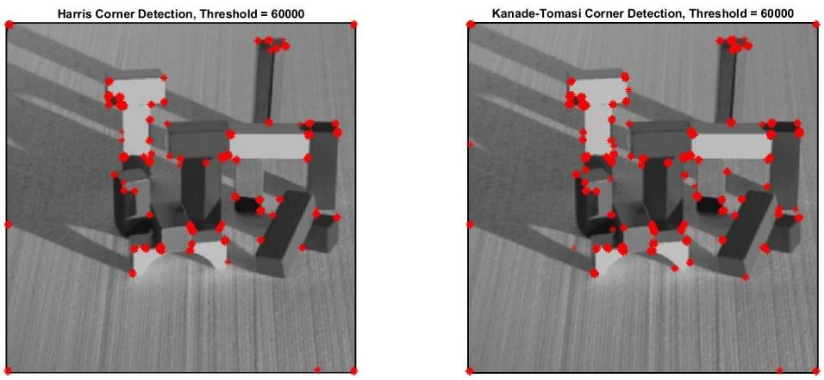
- Why do we use when we already have the Kanade-Tomasi Algorithm that fully calculates the eigenvalues of the H-matrix?
- The Harris operator computationally more advantageous than the Kanade-Tomasi method. This is predicated on the fact that in the calculation of f, there's no need to find the square root of any parameter. This greatly reduces the complexity of problems and puts significantly less strain on computational resources.

$$\lambda_{\pm} = \frac{1}{2}\left[(h_{11} + h_{22}) \pm \sqrt{4h_{12}h_{21} + (h_{11} - h_{22})^2}\right]$$

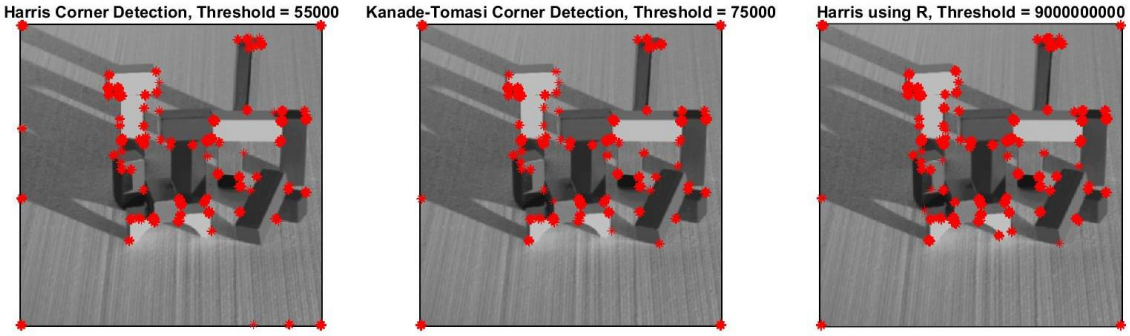
- This was particularly important around the 80's when computers with high computational power were particularly hard to come by.
- The threshold used is dependent both on the image and the corner detector used.

Comparison – Kanade-Tomasi and Harris

- The Kanade-Tomasi algorithm is computationally more demanding than the Harris algorithm. This is because of the need to find square roots, as opposed simple matrix parameter calculation in the Harris algorithm.
- Based on the above, *it might not be crass to assume that the Harris operator is better for real-time applications.*
- In both cases, the threshold used is image specific and method specific.
- In the following image, we see that the same threshold was used both the outcome was slightly different with the two algorithms:



- Notice that the Kanade-Tomasi detects more corners compared to the Harris operator.
- What's the reason behind this? The Harris operator computes an approximate minimum eigenvalue.
- Also, I don't think we can conclusively say that one method is better than the other. I believe that our results are heavily dependent on the threshold applied.
- Check out the following results. Different thresholds were used to try to obtain approximately similar results.



- Here, the Harris operator picks up some points close to the bottom right corner which ordinarily don't qualify as corners.

REFERENCES

Reinhard Klette. 2014. Concise Computer Vision: An Introduction into Theory and Algorithms. Springer Publishing Company, Incorporated.

https://commons.wikimedia.org/wiki/File:Error_surface_of_a_linear_neuron_with_two_input_weights.png

https://en.wikipedia.org/wiki/Harris_corner_detector

https://www.shutterstock.com/video/clip-4710293-colorful-cubes-rotation-seamless-loop

https://www.drupal.org/project/colours

https://unsplash.com/s/photos/building

https://www.ubm-development.com/magazin/en/without-a-trace/

https://www.wallpaperbetter.com/en/search?q=Summer+Scenery

https://www.londonremembers.com/memorials/king-lud-entrance