Harris corner detector

Goal

In this tutorial you will learn:

- What features are and why they are important
- Use the function cornerHarris to detect corners using the Harris-Stephens method.

Theory

What is a feature?

- In computer vision, usually we need to find matching points between different frames of an environment. Why? If we know how two images relate to each other, we can use *both* images to extract information of them.
- When we say **matching points** we are referring, in a general sense, to *characteristics* in the scene that we can recognize easily. We call these characteristics **features**.
- **So, what characteristics should a feature have?**
    - It must be *uniquely recognizable*

Types of Image Features

To mention a few:

- Edges
- **Corners** (also known as interest points)
- Blobs (also known as regions of interest)

In this tutorial we will study the *corner* features, specifically.

Why is a corner so special?

- Because, since it is the intersection of two edges, it represents a point in which the directions of these two edges *change*. Hence, the gradient of the image (in both directions) have a high variation, which can be used to detect it.

How does it work?

- Let's look for corners. Since corners represents a variation in the gradient in the image, we will look for this "variation".
- Consider a grayscale image I. We are going to sweep a window $w(x, y)$ (with displacements u in the x direction and v in the right direction) I and will calculate the variation of intensity.

$$E(u, v) = \sum_{x,y} w(x, y)[I(x + u, y + v) - I(x, y)]^2$$

where:

  - $w(x, y)$ is the window at position (x, y)
  - $I(x, y)$ is the intensity at (x, y)
  - $I(x + u, y + v)$ is the intensity at the moved window $(x + u, y + v)$

- Since we are looking for windows with corners, we are looking for windows with a large variation in intensity. Hence, we have to maximize the equation above, specifically the term:

$$\sum_{x,y}[I(x + u, y + v) - I(x, y)]^2$$

- Using *Taylor expansion*:

$$E(u, v) \approx \sum_{x,y}[I(x, y) + uI_x + vI_y - I(x, y)]^2$$

- Expanding the equation and cancelling properly:

$$E(u, v) \approx \sum_{x,y} u^2 I_x^2 + 2uvI_x I_y + v^2 I_y^2$$

- Which can be expressed in a matrix form as:

$$E(u, v) \approx \begin{bmatrix} u & v \end{bmatrix} \left(\sum_{x,y} w(x, y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \right) \begin{bmatrix} u \\ v \end{bmatrix}$$

- Let's denote:

$$M = \sum_{x,y} w(x, y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

- So, our equation now is:

$$E(u, v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix}$$

- A score is calculated for each window, to determine if it can possibly contain a corner:

$$R = det(M) - k(trace(M))^2$$

where:

- det(M) = $\lambda_1 \lambda_2$
- trace(M) = $\lambda_1 + \lambda_2$

a window with a score R greater than a certain value is considered a “corner”

Code

This tutorial code’s is shown lines below. You can also download it from here

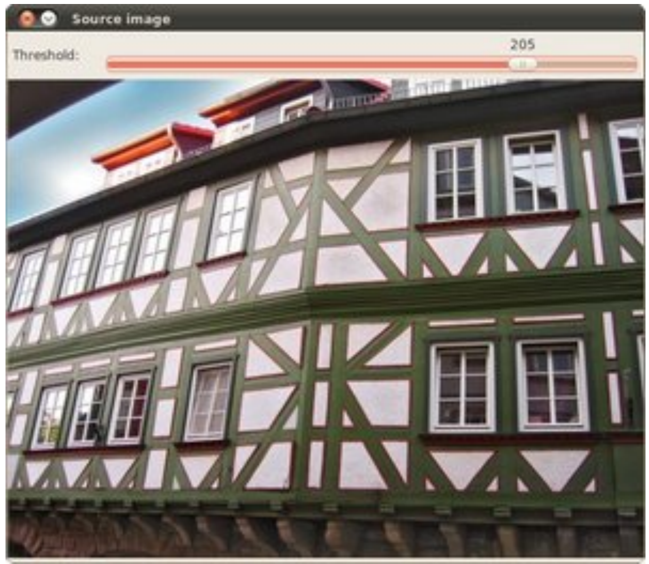
```
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <iostream>
#include <stdio.h>
#include <stdlib.h>

using namespace cv;
using namespace std;

/// Global variables
Mat src, src_gray;
int thresh = 200;
int max_thresh = 255;

char* source_window = "Source image";
char* corners_window = "Corners detected";

/// Function header
void cornerHarris_demo( int, void* );

/** @function main */
int main( int argc, char** argv )
{
  /// Load source image and convert it to gray
  src = imread( argv[1], 1 );
  cvtColor( src, src_gray, CV_BGR2GRAY );

  /// Create a window and a trackbar
  namedWindow( source_window, CV_WINDOW_AUTOSIZE );
  createTrackbar( "Threshold: ", source_window, &thresh, max_thresh, cornerHarris_demo
  imshow( source_window, src );

  cornerHarris_demo( 0, 0 );
```

```
  waitKey(0);
  return(0);
}

/** @function cornerHarris_demo */
void cornerHarris_demo( int, void* )
{

  Mat dst, dst_norm, dst_norm_scaled;
  dst = Mat::zeros( src.size(), CV_32FC1 );

  /// Detector parameters
  int blockSize = 2;
  int apertureSize = 3;
  double k = 0.04;

  /// Detecting corners
  cornerHarris( src_gray, dst, blockSize, apertureSize, k, BORDER_DEFAULT );

  /// Normalizing
  normalize( dst, dst_norm, 0, 255, NORM_MINMAX, CV_32FC1, Mat() );
  convertScaleAbs( dst_norm, dst_norm_scaled );

  /// Drawing a circle around corners
  for( int j = 0; j < dst_norm.rows ; j++ )
     { for( int i = 0; i < dst_norm.cols; i++ )
          {
            if( (int) dst_norm.at<float>(j,i) > thresh )
              {
               circle( dst_norm_scaled, Point( i, j ), 5,  Scalar(0), 2, 8, 0 );
              }
          }
     }
  /// Showing the result
  namedWindow( corners_window, CV_WINDOW_AUTOSIZE );
  imshow( corners_window, dst_norm_scaled );
}
```

Explanation

Result

The original image:



The detected corners are surrounded by a small black circle