Question 4 - Harris Corner Detection

November 23, 2018

1 Harris Corner Detector

We were tasked to find corners on an image of the college for interest point detection. The corners were to be superimposed on the original image and shown as the final result.

1.1 The code

We start by importing all libraries required for performing the filtering
1. cv2: OpenCV is a library of programming functions mainly aimed at real-time computer vision.
2. numpy: NumPy adds support for large, multi-dimensional arrays along with a large collection of functions to operate on these arrays.
3. matplotlib: This is the most widely used plotting library available for python.

```
In [7]: import cv2
        import numpy as np
        from matplotlib import pyplot as plt
        plt.rcParams['figure.figsize'] = [15, 15]
```

We will now read the input images using `imread` method. OpenCV's `imread` reads the image in the BGR color-space by default. We convert the color-space to RGB and GRAY, and save them as different images using `cvtColor` method. The grayscale image `img` will be used to detect corners.

```
In [2]: # read given image and convert to RGB color-space
        image = cv2.imread("/home/sarvesh/Projects/Image Processing/IITG.jpg")
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

        # save a grayscale version of the image to be used for corner detection
        img = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
        img = img.astype('float64')
```

At a corner there is change in the intensity in all directions. This information is captured in the structure tensor matrix which is computed using the gradient map of the image. We compute the corner response using the structure tensor matrix. We now define three functions below.

1. The function `calc_corner_response` which takes the grayscale image, and the sensitivity (alpha) as input to perform two tasks:
    - Calculate the gradient at each pixel.
    - Use the gradient to calculate the corner response and return the same.

```
In [3]: def calc_corner_response(image, alpha):

            h = len(image)
            w = len(image[0])

            # initialize a container to save the corner responses for every pixel
            corner_responses = np.zeros((h, w))

            for x in range(2, h - 2):
                for y in range(2, w - 2):

                    # square of the gradient in horizontal direction, vertical direction,
                    # and product of the two gradients
                    Ixx, Iyy, Ixy = (0, 0, 0)

                    # only considering 5x5 window for calculating the corner response
                    for i in range(5):
                        for j in range(5):
                            n_x = x + 2 - i
                            n_y = y + 2 - j

                            Ixx += (img[x][n_y] - img[x][y])**2
                            Iyy += (img[n_x][y] - img[x][y])**2
                            Ixy += (img[x][n_y] - img[x][y]) * (
                                img[n_x][y] - img[x][y])

                    det = Ixx * Iyy - Ixy**2
                    trace = Ixx + Iyy

                    r = det - alpha * (trace**2)

                    corner_responses[x][y] = r

            return corner_responses
```

2. The function `non_maximal_suppression` performs non maximal suppression on the corner responses given a threshold.
   - Non maximal suppression is performed over a 5x5 window.
   - If the corner response of the local maximas exceeds the threshold, it is marked as a corner and its coordinates are stored.

```
In [4]: def non_maximal_suppression(responses, threshold):
            h = len(responses)
            w = len(responses[0])

            # initialize containers to store the coordinates of the corners
            cornerListx = []
            cornerListy = []
```

```
        for y in range(2, h - 1):
            for x in range(2, w - 1):
                # the first condition checks whether the pixel is the
                # local maxima in a 5x5 window
                # the second condition checks if the corner response is above threshold
                if (responses[y][x] == np.amax(responses[y - 2:y + 3, x - 2:x + 3])
                        and responses[y][x] > threshold):
                    cornerListx.append(x)
                    cornerListy.append(y)

        return cornerListx, cornerListy
```

2. The function `harris_corner_detector` is a wrapper which takes a grayscale image, sensitivity (alpha) and the threshold as input to perform corner detection using the Harris corner algorithm.

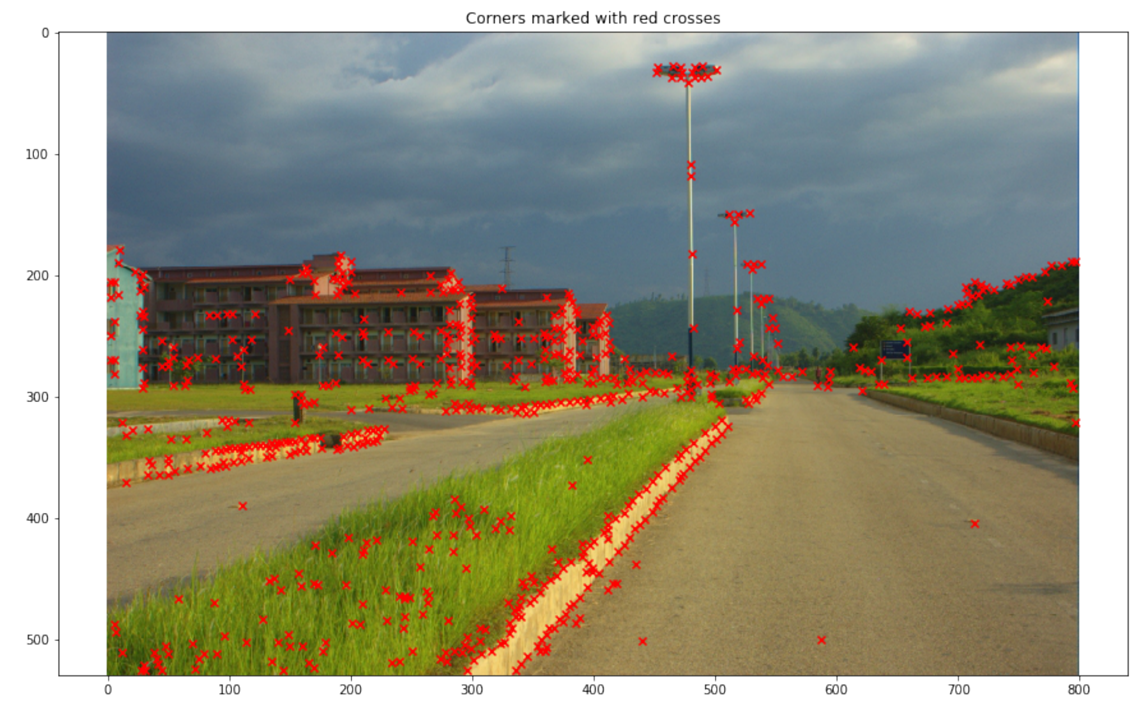
   - Call the `calc_corner_response` to get corner responses.
   - Call the `non_maximal_suppression` passing the corner respones and get the coordinates of the corners.
   - Mark the corners and superimpose them on to the original image and display it.

```
In [5]: def harris_corner_detector(source, alpha, thres):

            corner_scores = calc_corner_response(source, alpha)
            cX, cY = non_maximal_suppression(corner_scores, thres)

            # while displaying the image the corners are marked with a red cross
            plt.imshow(image)
            plt.scatter(cX, cY, c='red', marker='x')
            plt.title("Corners marked with red crosses")
            plt.show
```

We now perform corner detection using the harris corner algorithm over the given image. Note that we have converted the image to grayscale.

The **sensitivity** and **threshold** were set to 0.05 and 100000000 respectively. Multiple observations were made and the values which gave the best results were chosen.

```
In [8]: harris_corner_detector(img, 0.05, 100000000)
```



1.2 Conclusion

The corners were successfully detected and the result was shown. The corners can be used as interest points for various application like image alignment, motion tracking and object recognition to name a few.