MADRAS INSTITUTE OF TECHNOLOGY

ANNA UNIVERSITY

DEPARTMENT OF INFORMATION TECHNOLOGY

AD23401- COMPUTER VISION

ASSIGNMENT
HARRIS CORNER DETECTION

REGISTER NUMBER: 2023510005

NAME: VIJAYA MARTIN A R

SEMESTER: 4

Introduction:

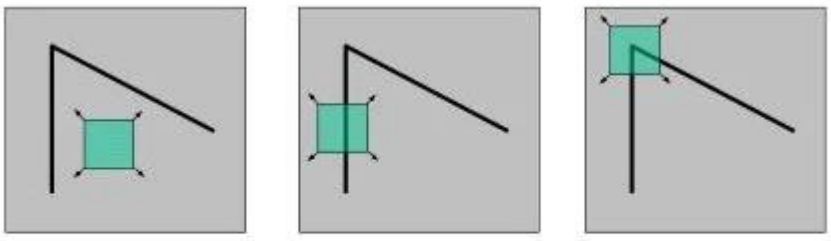
Harris Corner Detector is a corner detection operator that is commonly used in computer vision algorithms to extract corners and infer features of an image. It was first introduced by Chris Harris and Mike Stephens in 1988 upon the improvement of Moravec's corner detector. It is a method to extract the corners from the input image and to extract features from the input image.

In image processing, a corner is a point where the intensity changes significantly in multiple directions. Detecting these points is very important for various computer vision tasks, including image matching, motion tracking, and 3D reconstruction. Harris Corner detection algorithm was developed to identify the internal corners of an image.

Working of Harris Corner Detection Algorithm:

Corner:

A corner is a point whose local neighbourhood stands in two dominant and different edge directions. In other words, a corner can be interpreted as the junction of two edges, where an edge is a sudden change in image brightness. Corners are the important features in the image, and they are generally termed as interest points which are invariant to translation, rotation, and illumination.



Harris Corner Detection Algorithm:

Image gradients:

The first step involves calculating image gradients in the horizontal (x) and vertical (y) directions, essentially measuring how pixel intensity changes. To reduce noise, the image is typically smoothed using a small Gaussian filter before applying derivative filters like Sobel or Scharr. For instance, OpenCV's cv2.Sobel function can efficiently compute these gradients. These operations are performed via convolution using simple kernels, optionally combined with Gaussian blurring.

$$I_x(u,v) \approx \frac{\partial I}{\partial x}, \quad I_y(u,v) \approx \frac{\partial I}{\partial y}$$

Structure tensor (second moment matrix):

Next, we build what's called a structure tensor (or autocorrelation matrix) for each pixel by averaging the products of gradients within a local window—often using a Gaussian-weighted window. The result is a 2×2 matrix capturing how the intensity varies around a given point. This matrix is sometimes referred to as the Harris matrix or second moment matrix.

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

Eigenvalue interpretation:

This matrix has two eigenvalues that reflect how intensity changes in two perpendicular directions:

- If both eigenvalues are large → strong gradient in both directions → likely a corner.
- If one eigenvalue is large and the other small → strong gradient in one direction → an edge.
- If both eigenvalues are small → very little gradient change → a flat region.

Geometric interpretation:

These behaviors can be visualized as ellipses:

- Corners → ellipse with two large axes.

- ❖ Edges → elongated ellipse.
- ❖ Flat areas → tiny ellipse.

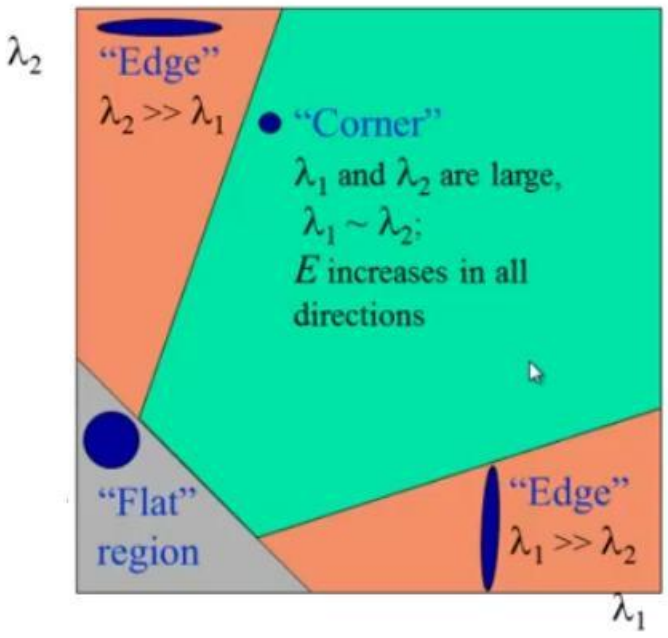
<u>Corner response function:</u>

To quantify "cornerness" from the eigenvalues, Harris and Stephens proposed a corner response function. It combines the determinant (which is large when both eigenvalues are large) and the trace (which helps penalize dominant single-direction gradients), controlled by a sensitivity constant. This avoids needing to calculate square roots, making it computationally efficient.

- ❖ Large, positive response → both eigenvalues large and similar → corner.
- ❖ Small or negative response → one eigenvalue dominates → edge.
- ❖ Small absolute response → both eigenvalues small → flat region.

$$R = \det(M) - k\,(\mathrm{trace}(M))^2 \;=\; \lambda_1\lambda_2 - k\,(\lambda_1 + \lambda_2)^2.$$

To pinpoint unique corners, the corner response is thresholded and non-maximum suppression is applied.



Simple Algorithm:

- Convert to Grayscale: If the input image is colored, convert it to grayscale to simplify processing.
- Compute Image Gradients: Calculate Ix and Iy using derivative operators like Sobel filters.
- Compute Products of Derivatives: Calculate Ix^2, Iy^2, and Ix * Iy.
- Apply Gaussian Filter: Smooth the derivative products with a Gaussian filter to reduce noise.
- Compute Corner Response: Use the corner response function R to evaluate each pixel.
- Thresholding: Identify pixels where R exceeds a certain threshold as potential corners.
- Non-Maximum Suppression: Refine corner detection by retaining only local maxima in the R response.

Implementation in Python With and Without In-Built Function:

Using In-Built Function:

```
img = cv2.imread('D:hsv.png')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
gray = np.float32(gray)

dst = cv2.cornerHarris(gray, blockSize=2, ksize=3, k=0.04)
dst = cv2.dilate(dst, None)

threshold = 0.01 * dst.max()
```

```
corners = dst > threshold

img_marked = img.copy()

corner_coords = np.argwhere(corners)

for y, x in corner_coords:
    cv2.circle(img_marked, (x, y), radius=7, color=(0, 0, 255), thickness=-1)

img_rgb = cv2.cvtColor(img_marked, cv2.COLOR_BGR2RGB)

plt.figure(figsize=(10,10))
plt.imshow(img_rgb)
plt.title('Harris Corners')
plt.axis('off')
plt.show()
```

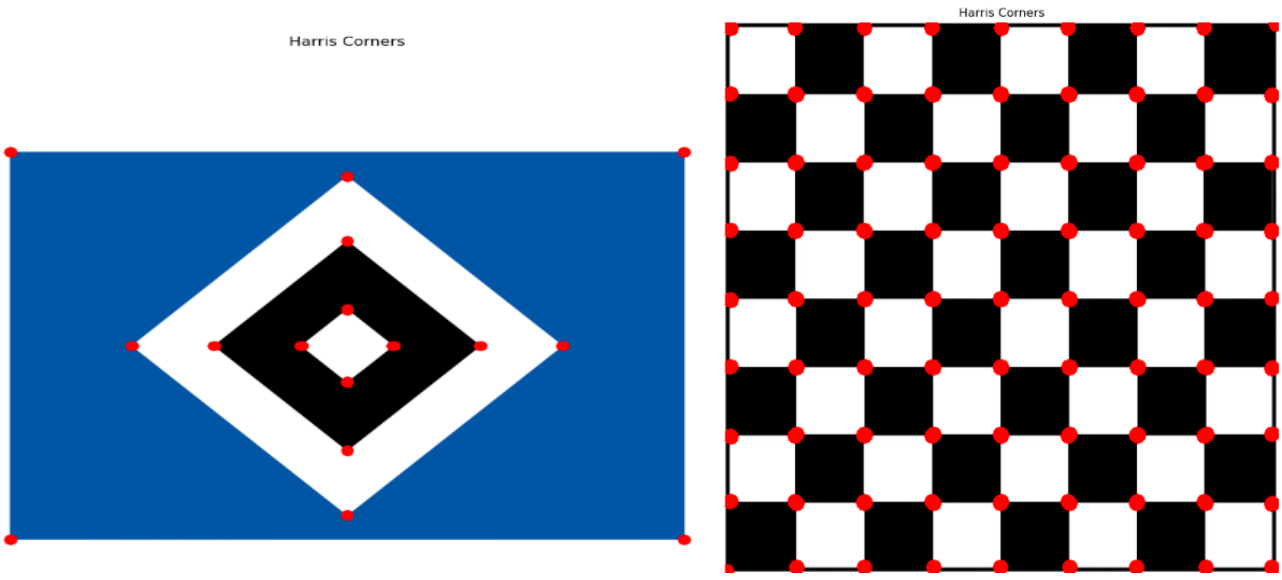
Without Using In-Built Function:

```
img_gray = cv2.imread('D:hsv.png', cv2.IMREAD_GRAYSCALE)
img_color = cv2.imread('D:hsv.png')

img_blur = cv2.GaussianBlur(img_gray, (3, 3), sigmaX=1)
Ix = cv2.Sobel(img_blur, cv2.CV_64F, 1, 0, ksize=3)
Iy = cv2.Sobel(img_blur, cv2.CV_64F, 0, 1, ksize=3)
Ixx = Ix * Ix
Iyy = Iy * Iy
Ixy = Ix * Iy
Sxx = cv2.GaussianBlur(Ixx, (3, 3), sigmaX=1)
Syy = cv2.GaussianBlur(Iyy, (3, 3), sigmaX=1)
Sxy = cv2.GaussianBlur(Ixy, (3, 3), sigmaX=1)

k = 0.04
R = (Sxx * Syy - Sxy**2) - k * (Sxx + Syy)**2

threshold = 0.01 * R.max()
corner_coords = np.argwhere(R > threshold)

for y, x in corner_coords:
```

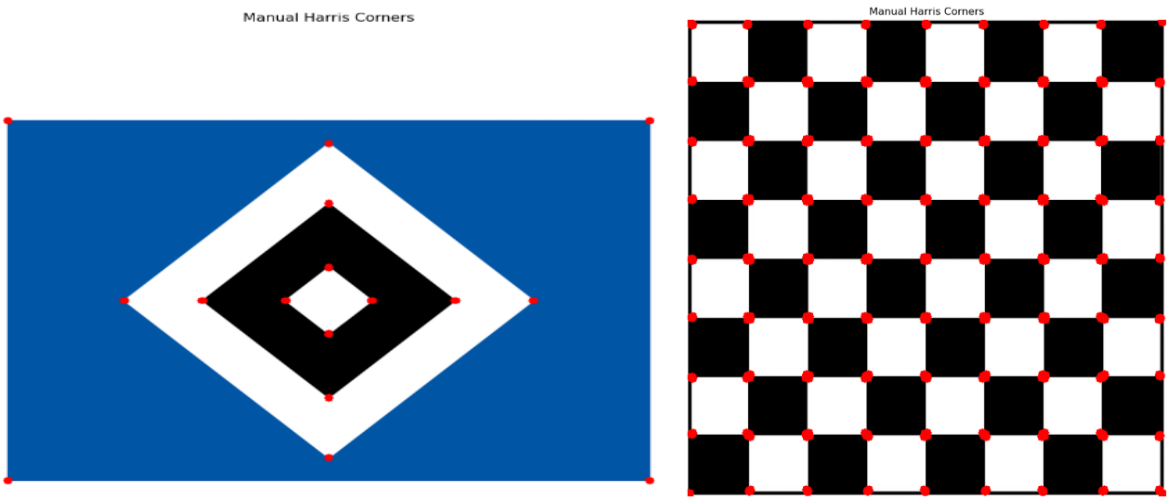
```
    cv2.circle(img_color, (x, y), radius=5, color=(0, 0, 255), thickness=-1)

img_rgb = cv2.cvtColor(img_color, cv2.COLOR_BGR2RGB)

# Display results
plt.figure(figsize=(10, 10))
plt.imshow(img_rgb)
plt.title('Manual Harris Corners')
plt.axis('off')
plt.show()
```

Output:

Using In-Built Function:



Without Using In-Built Function:



Applications:

- Image Matching and Stitching: Identifying corresponding points between images for alignment.
- Motion Tracking: Following features across video frames.
- 3D Reconstruction: Using corner points to infer 3D structure from multiple views.
- Object Recognition: Detecting and recognizing objects based on distinctive features.

Conclusion:

The Harris Corner Detector is still one of the most influential and broadly implemented computer vision techniques for the detection of interest points. Through the detection of local image gradients using the structure tensor, it is able to robustly detect corners—areas of high intensity variation in many different directions. Its rotation invariance and computational simplicity mean it is particularly well-suited to applications such as feature matching, motion tracking, and object recognition.