Yash Maniya
B20CS033

Computer Vision (CSL7360)
Programming Assignment-1

1. Harris Corner detection :

Harris Corner Detection is a method used to detect corners in images. It operates based on the intensity changes in different directions within a small window of the image.

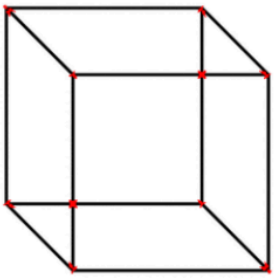
Principle : Harris Corner Detection identifies corners by looking for significant changes in intensity in all directions. It measures the variation of intensity when the image is translated by a small amount in any direction.

Steps :

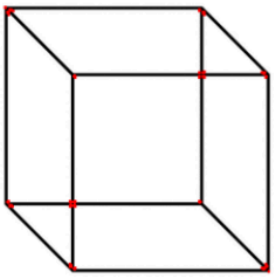
1. **Image Loading:** Images are loaded from the specified directory and resized to a standard size to ensure uniform processing.
2. **Edge Detection:** Sobel operators are applied to calculate gradients in the x and y directions for edge detection.
3. **Gaussian Smoothing:** Gaussian blur is applied to smooth the image and reduce noise using a specified kernel size.
4. **Harris Corner Detection:** Harris corner detection algorithm is implemented. This involves computing the Harris response function, which combines intensity gradients and covariance matrices of pixel neighborhoods.
5. **Thresholding:** Detected corners are thresholded based on the Harris response to identify significant corners.
6. **Comparison with OpenCV:** The results obtained from the custom implementation are compared with OpenCV's cornerHarris function to validate the accuracy of the detection.
7. **Visualization:** Detected corners are overlaid on the original images for visualization, both for the custom implementation and OpenCV's function.

Applications : It's widely used in feature detection and matching, which is fundamental in tasks like image stitching, object recognition, and visual tracking.

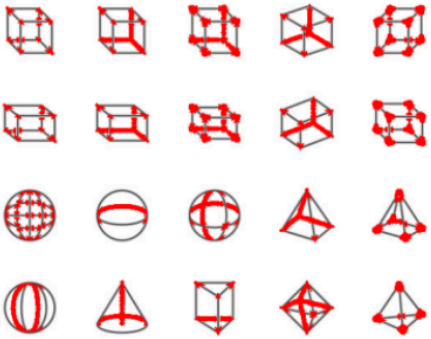
Custom Harris Corner Detection - Image 1



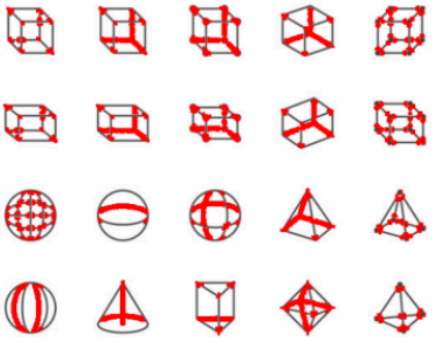
OpenCV Corner Detection - Image 1



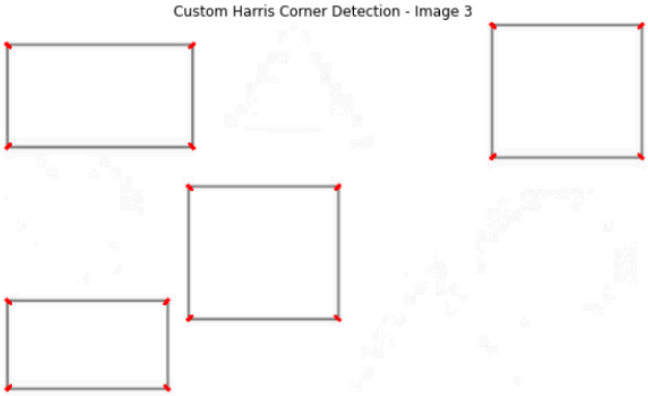
Custom Harris Corner Detection - Image 2



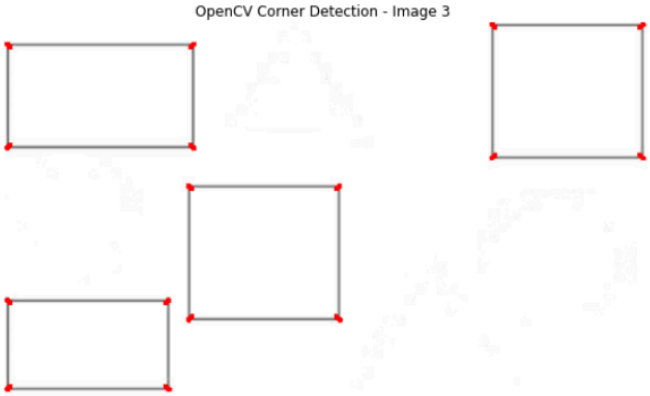
OpenCV Corner Detection - Image 2



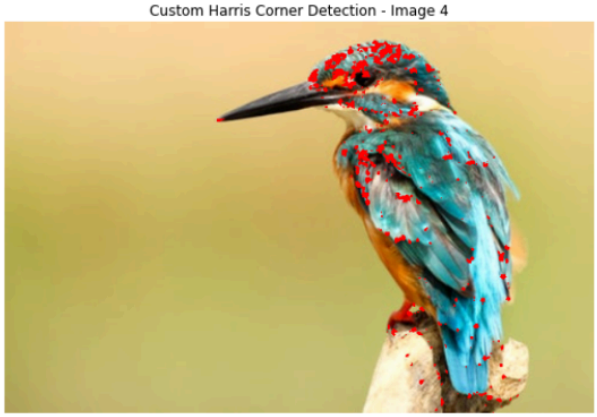
Custom Harris Corner Detection - Image 3



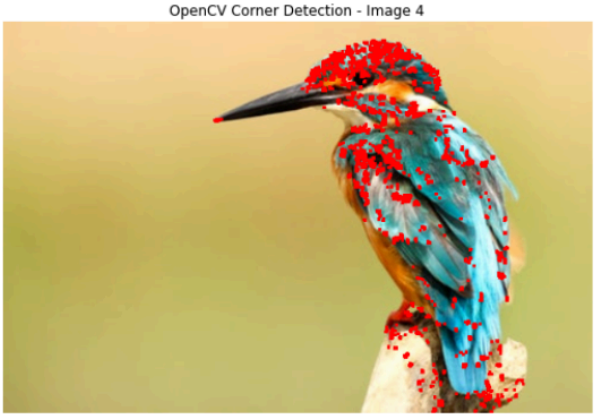
OpenCV Corner Detection - Image 3



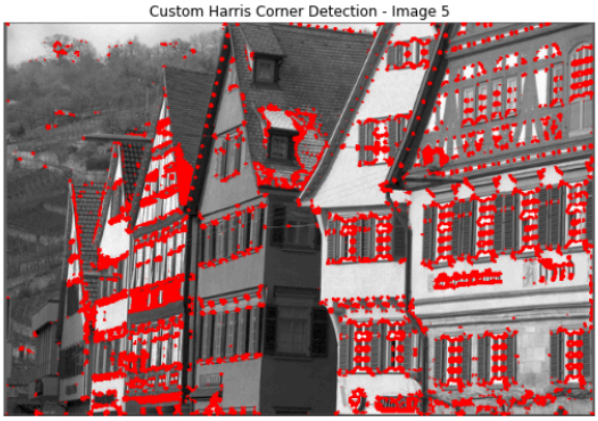
Custom Harris Corner Detection - Image 4



OpenCV Corner Detection - Image 4



Custom Harris Corner Detection - Image 5

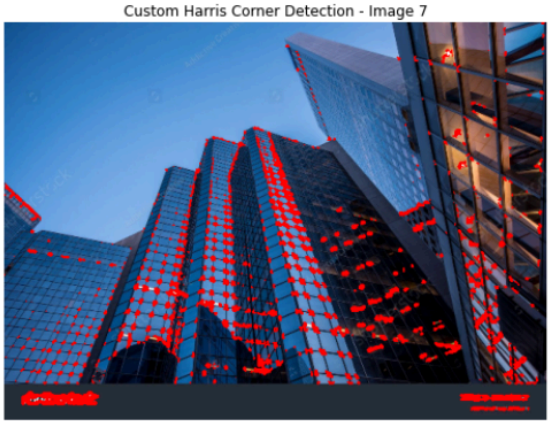


OpenCV Corner Detection - Image 5

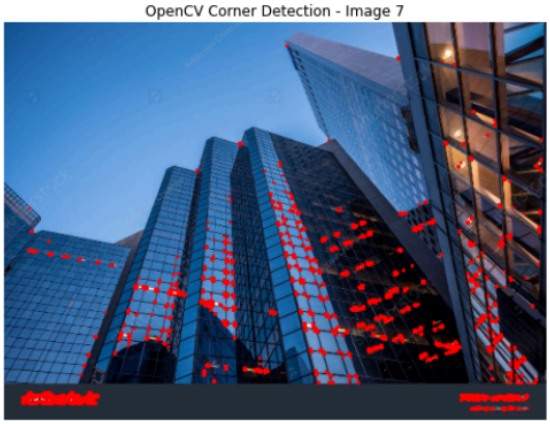
Custom Harris Corner Detection - Image 6

OpenCV Corner Detection - Image 6

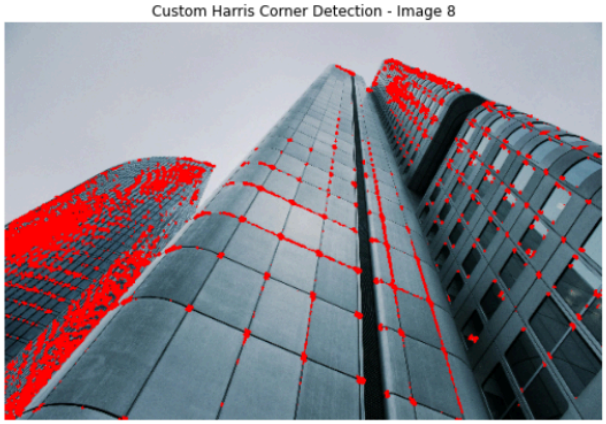
Custom Harris Corner Detection - Image 7



OpenCV Corner Detection - Image 7



Custom Harris Corner Detection - Image 8



OpenCV Corner Detection - Image 8

Regarding the input parameters and their effects:

- **window_size:** This parameter determines the size of the window used for Harris corner detection. Larger window sizes capture more local structure but may also introduce more noise.
- **threshold:** Threshold value controls the sensitivity of corner detection. Increasing the threshold will result in fewer corners being detected, while decreasing it will lead to more corners being detected. It affects the strictness of corner identification.

These parameters allow adjusting the trade-off between corner sensitivity and noise robustness in the corner detection process. Adjustments should be made based on the characteristics of the images and the specific requirements of the application.

2. **Stereo 3D Reconstruction** :

Stereo 3D Reconstruction involves creating a 3D model of a scene from multiple 2D images taken from different viewpoints. This process typically involves generating a disparity map, depth map, and eventually a 3D point cloud representation.

- **Disparity Map :** It measures pixel-wise differences between corresponding points in stereo images. It's computed by finding the horizontal shift needed to match pixels in the left and right images.

- **Depth Map :** This map assigns a depth value to each pixel in an image, representing the distance from the camera to the scene point corresponding to that pixel.

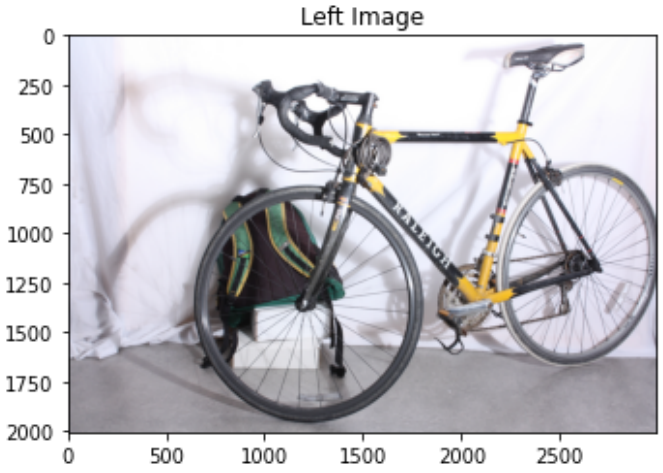
- **3D Point Cloud Representation :** It's a set of points in a 3D coordinate system where each point corresponds to a pixel in one of the stereo images, and its coordinates are determined by the disparity and depth information.

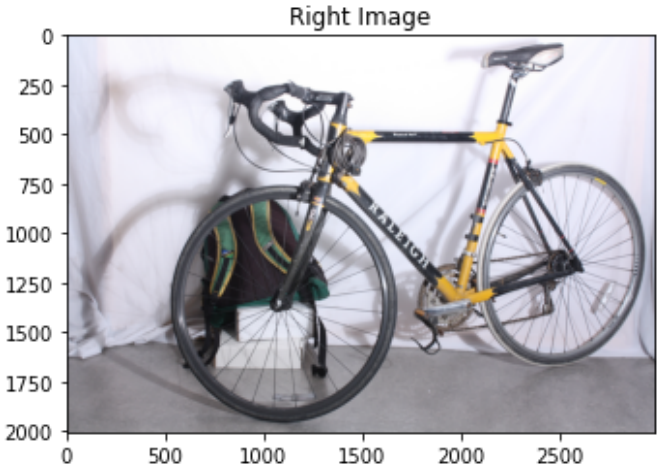
Steps :

1. **Load Stereo Images:** Two stereo images are loaded, namely 'bikeL.png' and 'bikeR.png'.
2. **Define Camera Parameters:** Intrinsic matrices for both cameras (cam0 and cam1) and the baseline distance between the cameras are defined.
3. **Convert Images to Grayscale:** The stereo images are converted to grayscale for processing.
4. **Compute Disparity Map:** Using OpenCV's StereoBM_create function, a disparity map is computed from the grayscale stereo images.
5. **Compute Depth Map:** The depth map is computed from the disparity map using the formula baseline / disparity.
6. **Compute 3D Point Cloud:** 3D coordinates of the scene points are computed using the disparity values and camera parameters.
7. **Mask out Invalid Depth Values:** Invalid depth values (where disparity is 0) are masked out from the 3D point cloud.

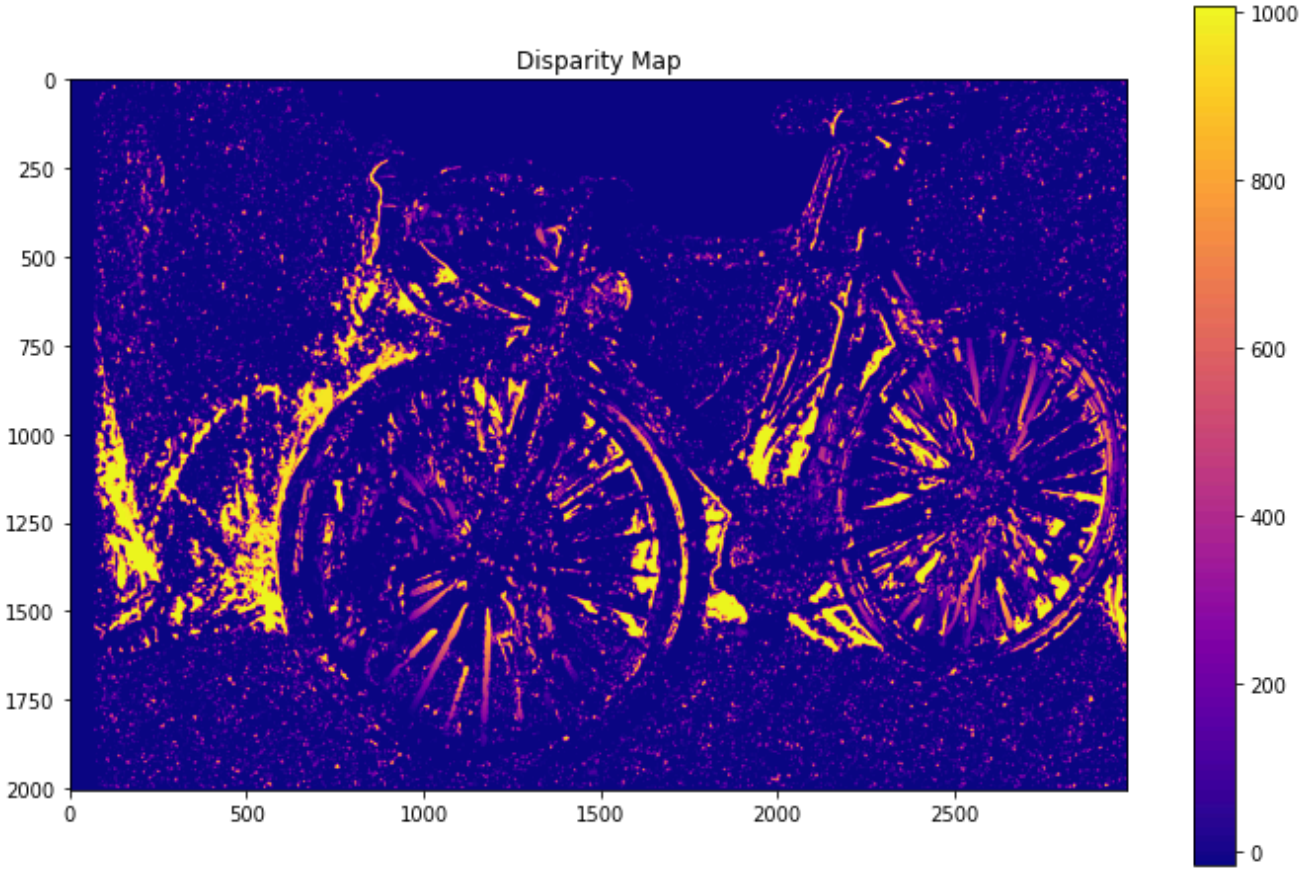
8. **Plotting:**

    - Left and right images are displayed side by side.

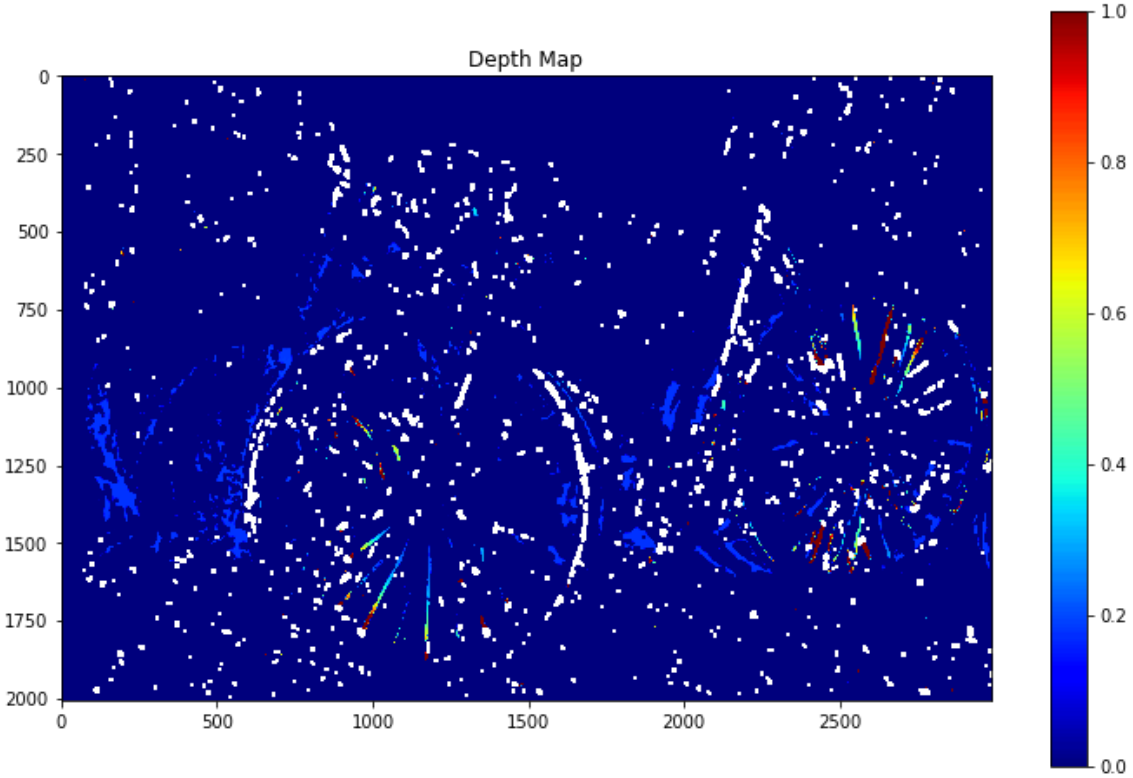




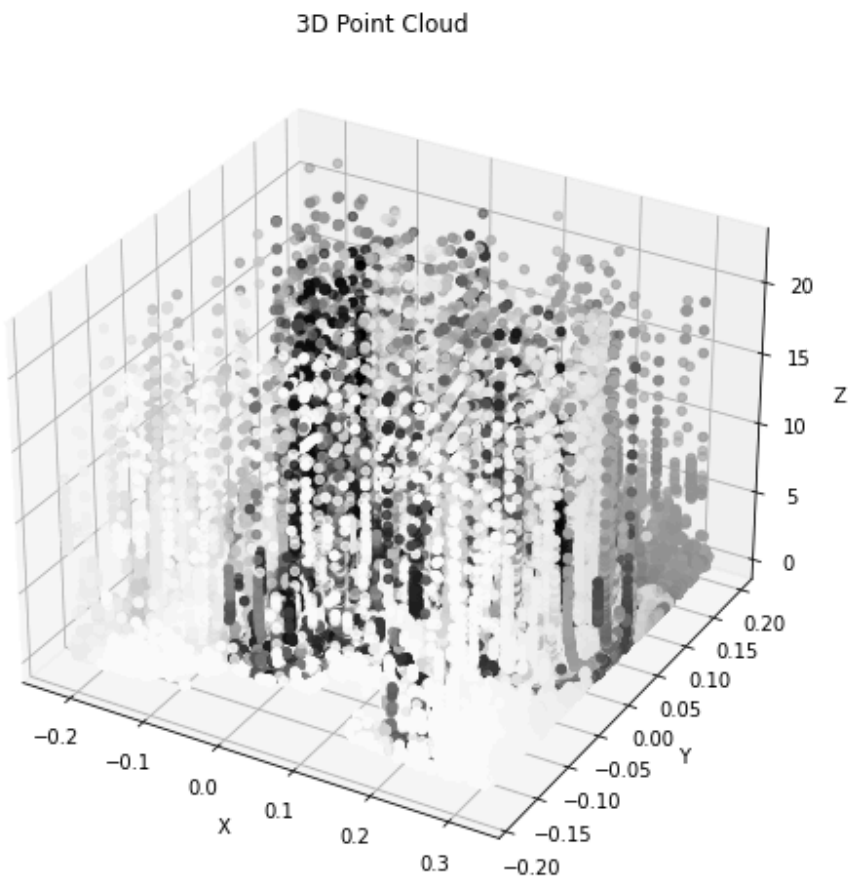
- Disparity map is plotted using a colormap.



- Depth map is plotted using a jet colormap to represent depths.



- 3D point cloud representation is visualized using a scatter plot in a 3D space with grayscale color intensity.



3. Epipolar Geometry :

Epipolar Geometry describes the geometric relationship between two images of the same scene taken from different viewpoints.

Fundamental Matrix : It encodes the epipolar geometry between two images. Given corresponding points in the two images, it allows the calculation of epipolar lines in one image corresponding to points in the other.

Epipolar Constraint : This constraint states that the epipolar lines in one image correspond to the projection of the corresponding point in the other image. It simplifies the problem of matching points between two images.

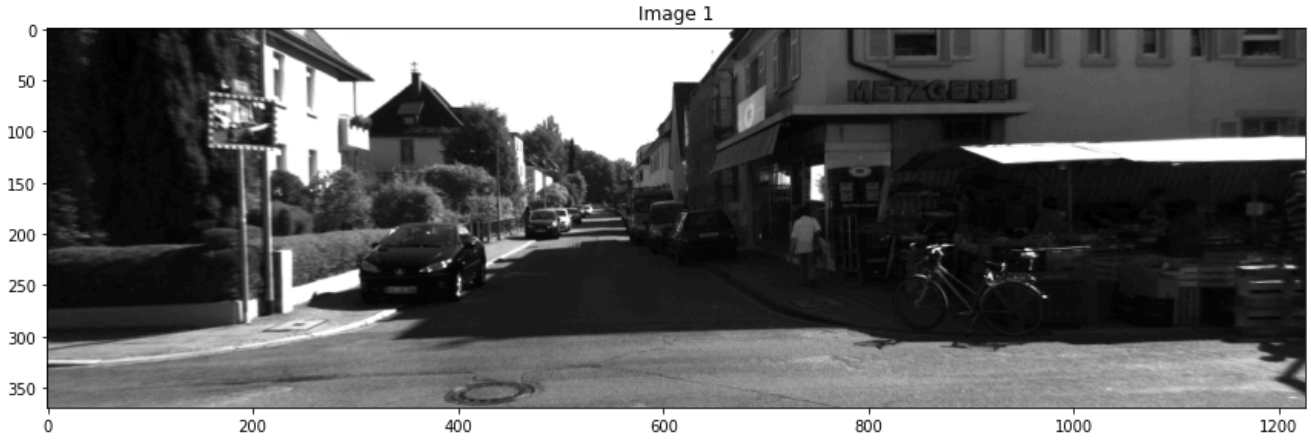
Epipolar Lines : These are lines in one image along which the corresponding point in the other image lies. They provide a search constraint for matching points in stereo images, reducing the computational complexity of stereo matching algorithms.

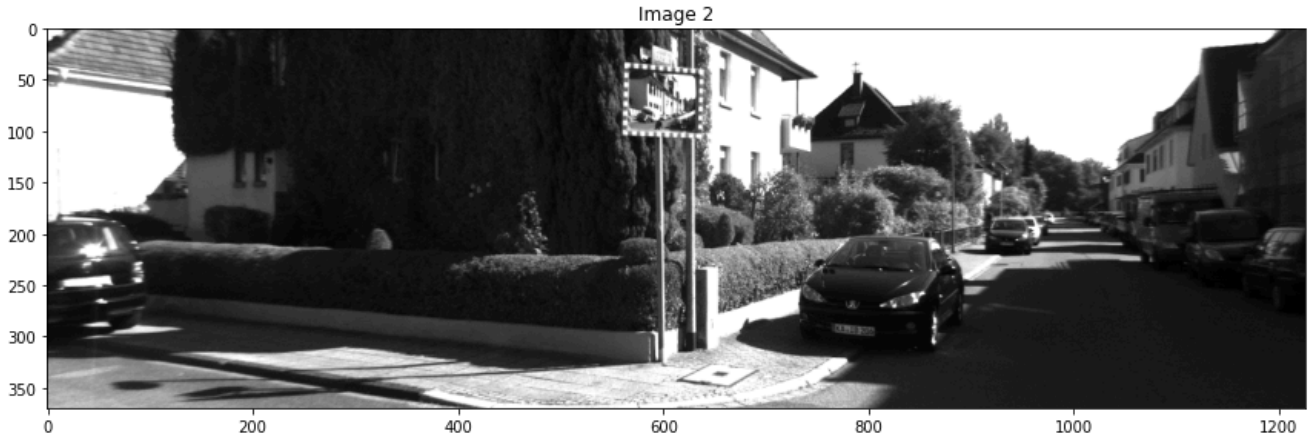
Applications : Epipolar geometry is fundamental in stereo vision, structure from motion, and visual odometry tasks. It's used for feature matching, camera calibration, and 3D reconstruction from multiple views.

Steps :

1. **Fundamental Matrix F:** Defines the Fundamental Matrix F, which encodes the epipolar geometry between two images.
2. **Load Images:** Loads two grayscale images, 'image1' and 'image2', which are taken from different viewpoints.
3. **Plot Images:** Displays 'image1' and 'image2' side by side using Matplotlib.
4. **Compute Epipolar Lines:** Defines a function 'compute_epipolar_lines' to compute epipolar lines given the Fundamental Matrix F and a set of points. Then, it computes epipolar lines for points provided in the 'points' array on 'image1' and 'image2'.
5. **Visualize Epipolar Lines:** Plots the epipolar lines corresponding to the points on 'image1' and 'image2' using Matplotlib.

Image1 and image2, which are taken from different viewpoints :





The yellow lines represent the epipolar lines corresponding to 2 different points in world.