**CV Assignment-1**

**Group:** G Sai Nikhil, D Veera Harsha Vardhan Reddy, N Karthik Raja

**Question 1:** Visualize the depth map and provide an analysis of its accuracy. Discuss potential challenges in disparity estimation, such as occlusion, repetitive patterns, and noise in real-world applications.

**Deliverables:**

- Implement the stereo matching and depth estimation algorithms in Python (or MATLAB).
- Submit the code, disparity map, depth map, and a brief report explaining the results and any challenges faced.

**Input:**

- Left image and Right image (stereo pair of images taken from a stereo camera). You can use datasets like the KITTI dataset for stereo vision or capture stereo images with a stereo camera.

**Expected Output:**

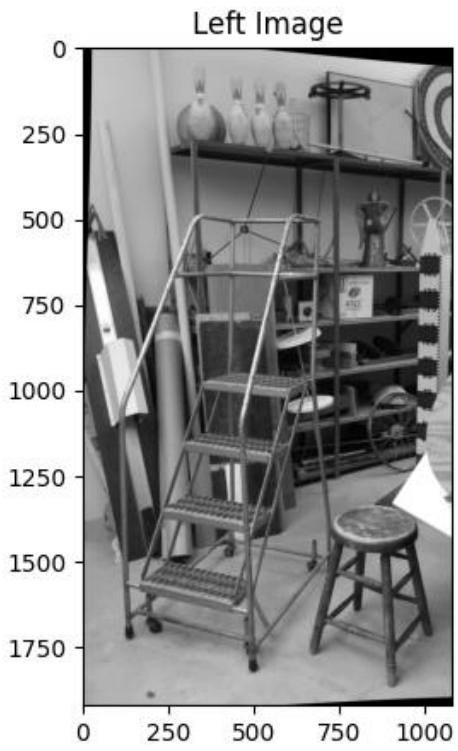
- Disparity Map: A grayscale image where lighter pixels represent greater disparity (closer objects).
- Depth Map: A grayscale image where darker regions represent distant objects, and lighter regions represent closer objects.

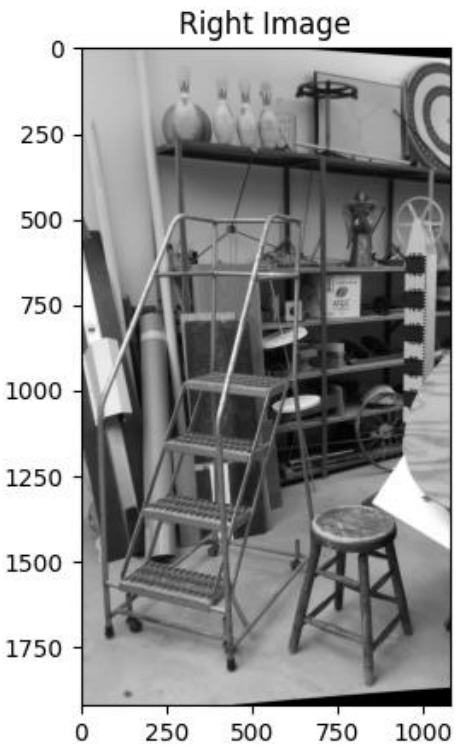
**Disparity Map**

- A disparity map represents the apparent pixel shifts (disparity) between two corresponding points in stereo images, which are captured from slightly different viewpoints (like left and right cameras).
- **Block Matching (BM)**:
    - BM is a traditional algorithm for computing a disparity map by comparing small blocks or patches in stereo images. It shifts these blocks horizontally between the two images to find matches and minimize disparity errors.
- **Semi-Global Block Matching (SGBM)**
    - SGBM is a more sophisticated variant of BM that minimizes the disparity error not only locally (within blocks) but also across neighbouring pixels in multiple directions, leading to a smoother disparity map.

**Depth Map**

- A depth map is a representation of the distances of objects in a scene from the camera, calculated based on the disparity. Each pixel in a depth map corresponds to the estimated depth or distance of the corresponding point in the 3D scene.




**Objective:** The goal is to compute and analyze disparity and depth maps from stereo images using two different stereo matching algorithms Block Matching (BM) and Semi-Global Block Matching (SGBM). The quality of the disparity and depth maps produced by each algorithm was evaluated, highlighting the challenges encountered in real-world applications of disparity estimation.

CODE:
https://colab.research.google.com/drive/1kLw6pb8WjH_kyCGHxIvLyhi_PAxRbsoZ?usp=sharing

**Disparity Map Results:**

1. **Disparity Calculation Formula**

$$d(x,y) = \arg\min_{d} Cost(I_L(x,y), I_R(x-d,y))$$

**Where**

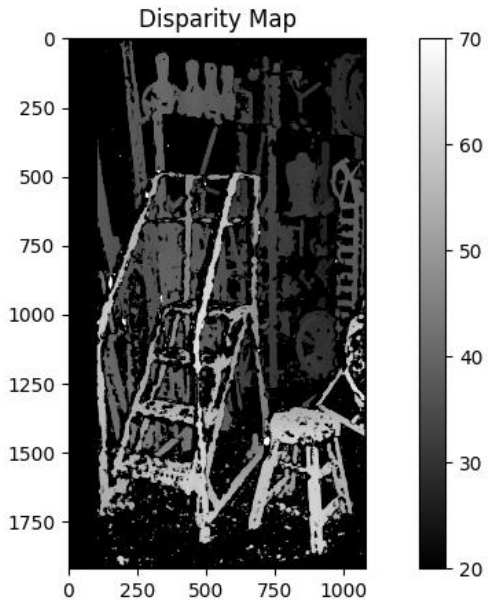
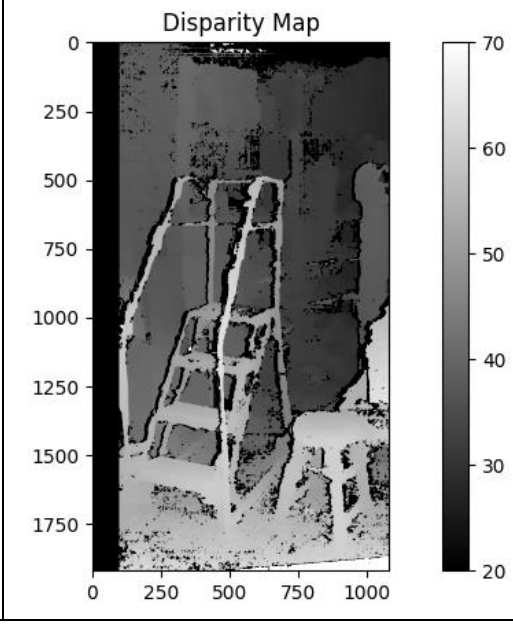
- $I_L(x,y)$ & $I_R(x-d,y)$ are the pixel intensities at location (x,y) in the left image and (x – d,y) in the right image.
- d is the disparity value (shift) for the pixel at (x,y).
- Cost(.) is a function that measures the similarity between pixels or blocks, such as Sum of Absolute Differences (SAD) or Sum of Squared Differences (SSD).

**Block Matching (BM)**: The disparity map generated using BM showed clear depth boundaries, particularly useful in regions with good contrast and non-repetitive patterns. The BM algorithm, due to its local approach, effectively estimated disparity for these simpler scenes. However, BM has limitations in handling occlusions, resulting in less accurate depth information in certain regions.

**Semi-Global Block Matching (SGBM)**: In contrast, the disparity map from SGBM provided smoother transitions and finer details across the image. SGBM's semi-global optimization approach improves depth consistency across large areas, especially in regions with repetitive patterns. This produced an overall better depth map, though SGBM is computationally more intensive and may be affected by noise and image texture inconsistencies.

**Observation**: In general, the disparity map produced by Block Matching appeared more effective, whereas SGBM offered a more robust solution for challenging scenes with repetitive patterns or complex textures.

**Results:**

| Block Matching | Semi Global Block Matching |
|---|---|
|  |  |



**Depth Map Results**

Using the calculated disparity maps, depth maps were generated. The resulting depth maps were clipped to a reasonable range ('max_display_depth' set to 10,000–12,000 mm) and normalized to fit within a grayscale scale of 0–255 for visualization.

$$d = (focal\ length * baseline)\ /(disparity + doffs)$$

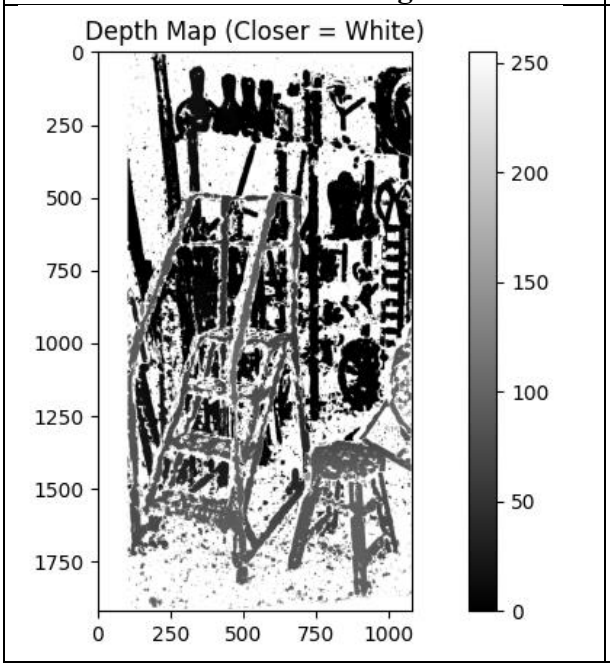
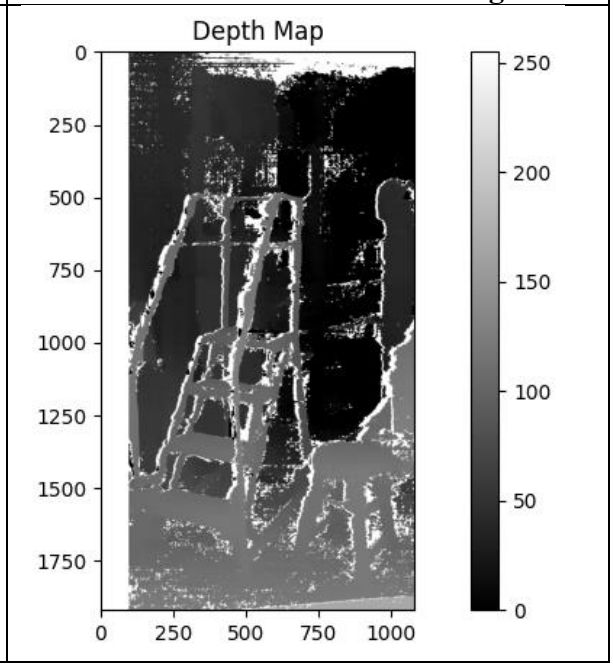
*Focal length = camera focal length*
*Baseline = Distance b/w two camera centres*
*Doffs = disparity offset*

**Depth Map from BM**: The BM-derived depth map maintained high contrast in clear regions but struggled in areas of occlusion and repetitive patterns, where disparity estimates were less reliable. This led to some noisy or blurred regions in the depth map where depth values were inaccurate.

**Depth Map from SGBM:** The SGBM-derived depth map was smoother and displayed more consistent depth information across the scene. SGBM better handled repetitive patterns and provided more detail in low-contrast regions. This produced a more visually accurate depth map but also suffered from potential artifacts in areas with low disparity or excessive noise.

**Observation**: The SGBM depth map was overall superior in maintaining depth consistency and detail, compared to BM.

**Results:**

| Block Matching | Semi Global Block Matching |
| --- | --- |
|  |  |





**Challenges Faced in Disparity Estimation**

**1. Occlusion:**

**Issue:** In areas where one image shows content that is occluded in the other, matching becomes inaccurate or impossible, resulting in gaps or noise in the disparity map.

**Solution:** The SGBM algorithm includes parameters (‘speckleWindowSize’ and ‘speckleRange’) to filter out inconsistent disparity values in occluded areas, reducing noise but not entirely eliminating inaccuracies.

**2. Repetitive Patterns:**

**Issue:** Repetitive textures cause ambiguity in disparity matching, as similar patterns appear at multiple potential depths.

**Solution:** The SGBM algorithm performs better in these cases by incorporating global consistency checks, although some ambiguity still remains, particularly in highly repetitive regions.

**3. Noise:**

**Issue:** Noise in images (e.g., due to lighting variations or camera artifacts) can create false depth estimates, especially for BM, which is highly sensitive to image texture.

**Solution:** Using parameters such as `blockSize` (in BM) and smoothing settings in SGBM helps, though completely removing noise effects remains challenging in real-world scenes.

**Question 2:** Use the extracted features to match keypoints between two images and evaluate the performance of both algorithms. Discuss scenarios where one method might outperform the other.

**Deliverables:**

- Implement SIFT and SURF feature extraction and matching in Python (use OpenCV or any preferred library).
- Submit the code, images with keypoints visualized, and a report comparing the performance of SIFT and SURF based on different metrics (e.g., speed, number of keypoints, accuracy of matching).

**Input:**

- Two input images where feature matching is needed (e.g., different views of the same object or scene).

**Expected Output:**

- SIFT/SURF Keypoints: Visualized keypoints on the input images.
- Matched Points: Matched keypoints visualized on a combined image where lines show which points match between the two images.

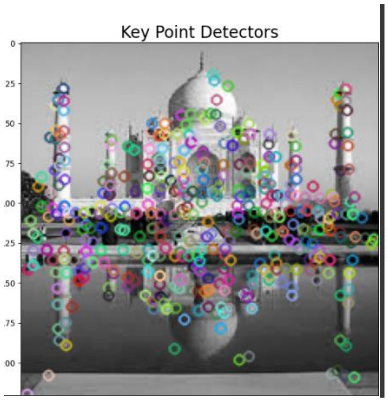
**Feature Extraction Using SIFT, ORB, FAST and BRIEF : Comparison for Images**

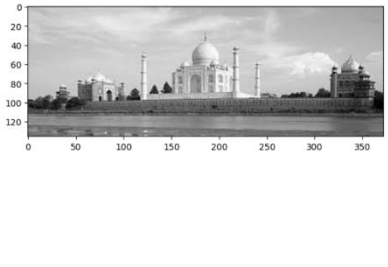
**Code :**
https://colab.research.google.com/drive/1PvpqzGc8Vwu_YZ9TLu8kpDKA3A1TtTIQ?usp=chrome_ntp

Overview of Feature Extraction Techniques:

1. **SIFT (Scale Invariant Feature Extraction):**

SIFT proposed by Lowe solves the image rotation, affine transformations, intensity, and viewpoint change in matching features. The SIFT algorithm has 4 basic steps. The first is to estimate a scale space extremum using the Difference of Gaussian (DoG). Secondly, a key point localization where the key point candidates are localized and refined by eliminating the low contrast points. Thirdly, a key point orientation assignment based on local image gradient and lastly a descriptor generator to compute the local image descriptor for each key point based on image gradient magnitude and orientation.



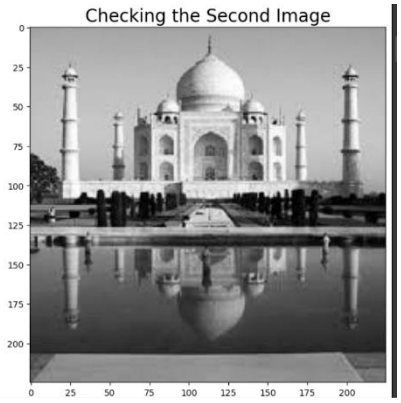



Fig 1 Fig 2

Fig 1: Representing the Key point detectors of the image

Fig 2: Displaying the both views of the Image

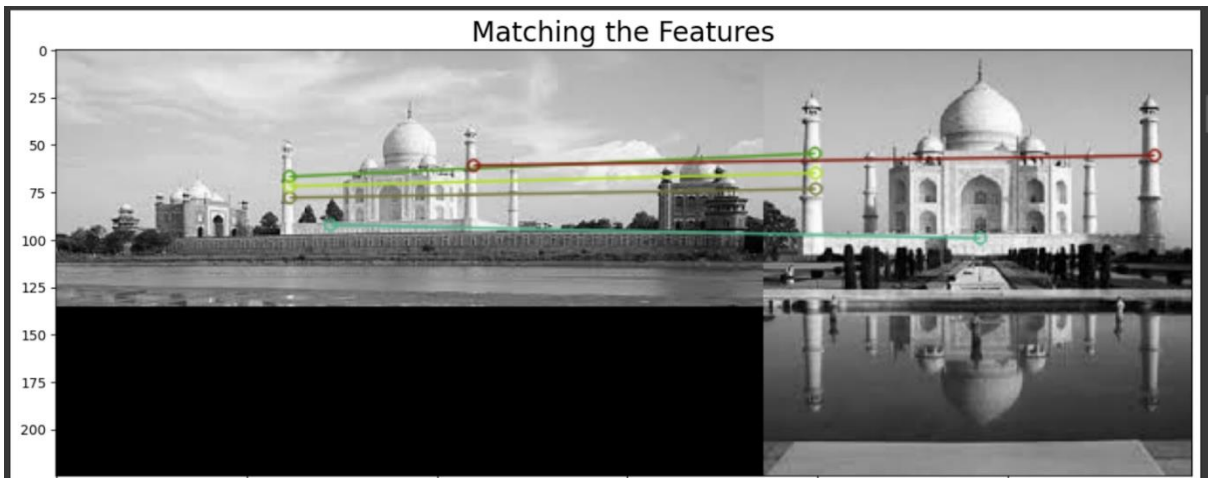



**Fig 3: Representing the Matching features for the two images**

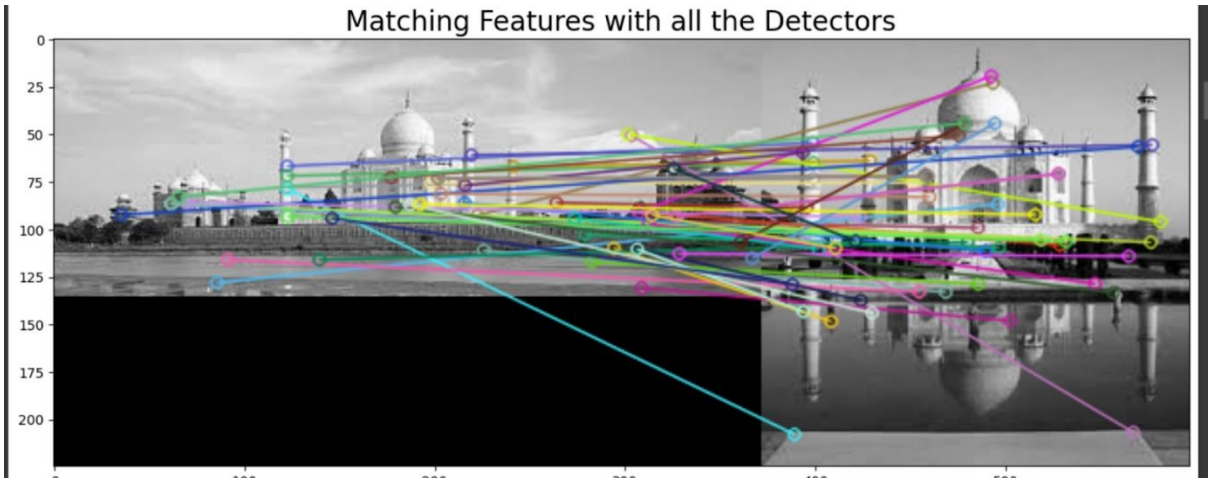



**Fig 4: Representing the Matching Features with all the detectors**

2. **ORB (Oriented FAST and Rotated Brief Algorithm)**

ORB is a fusion of the FAST key point detector and BRIEF descriptor with some modifications. Initially to determine the key points, it uses FAST. Then a Harris corner measure is applied to find top N points. FAST does not compute the orientation and is rotation variant. It computes the intensity weighted centroid of the patch with located corner at center. The direction of the vector from this corner point to centroid gives the orientation. Moments are computed to improve the rotation invariance. The descriptor BRIEF poorly performs if there is an in-plane rotation. In ORB, a rotation matrix is computed using the orientation of patch and then the BRIEF descriptors are steered according to the orientation.

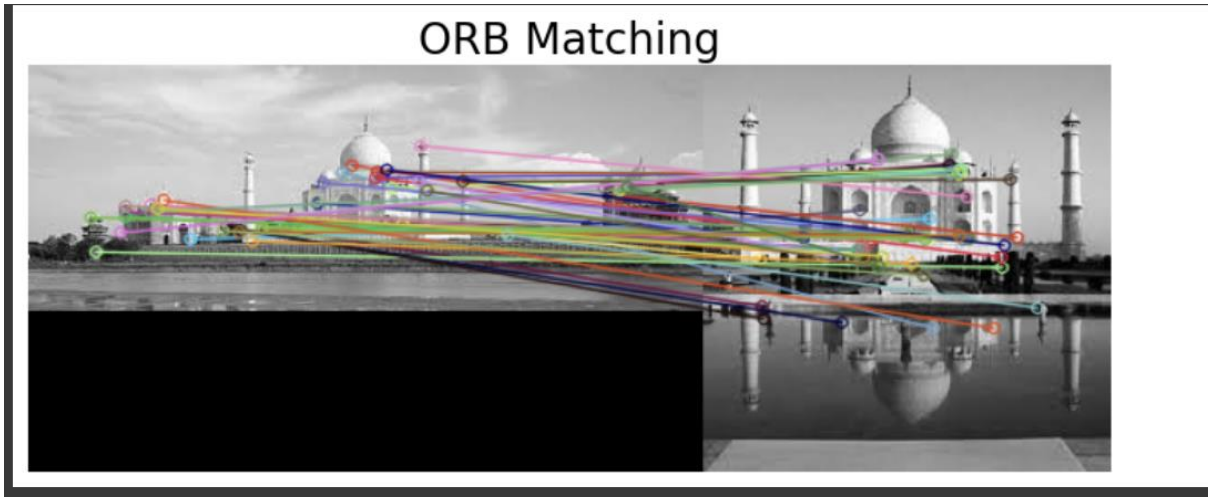



**Figure 5 : Displaying the Matching features with all the detectors**

3. **BRIEF (Binary Robust Independent Elementary Features):**

The Binary Robust Independent Elementary Features (BRIEF) descriptor is a feature descriptor designed for computational efficiency and robustness in matching tasks. Instead of computing descriptors based on gradients or other complex image properties, BRIEF employs simple binary tests between pairs of pixels within a small image patch. This approach significantly reduces the computational load, making it well-suited for real-time applications on resource-limited devices.

BRIEF's binary nature allows for rapid matching using the Hamming distance, a straightforward comparison method ideal for binary descriptors. However, BRIEF is not inherently scale or rotation-invariant; it performs best in controlled environments with minimal changes in scale and rotation. When combined with detectors like FAST, BRIEF provides a lightweight yet effective solution for applications requiring high-speed feature matching, such as SLAM and object tracking**.**

4. **FAST (Features from Accelerated Segment Test) :**

**Features from Accelerated Segment Test (FAST)** is a high-speed corner detection algorithm designed to be computationally efficient, making it ideal for real-time computer vision tasks. FAST determines if a pixel is a corner by examining a circle of 16 pixels around

it. If a contiguous subset of pixels in this circle is consistently brighter or darker than the central pixel by a set threshold, the pixel is classified as a corner.

FAST's efficiency comes from a short-circuit mechanism that only checks a subset of pixels first; if these pixels don't meet the corner criteria, it skips the rest, saving processing time. This allows FAST to process images significantly faster than traditional corner detectors like Harris or SIFT, making it suitable for applications such as object tracking and visual SLAM.

However, FAST is not scale- or rotation-invariant on its own. Despite this, its speed and effectiveness in detecting corners make it a popular choice for feature detection in embedded and real-time systems.

In this scenario, I am using FAST for keypoint detection and BRIEF for descriptor extraction to analyze two images from different perspectives.

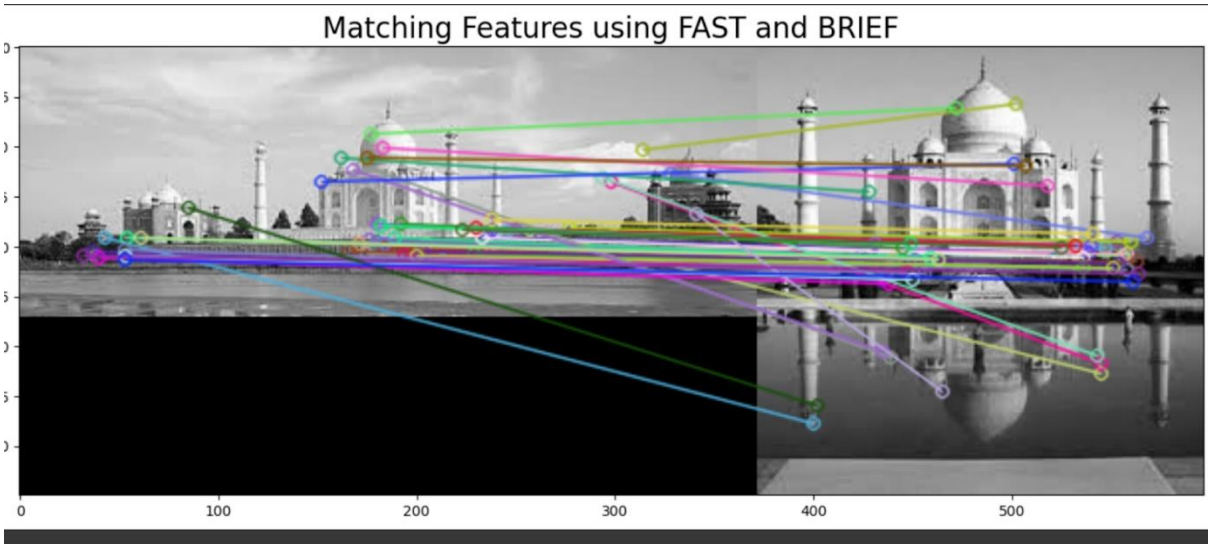



**Fig 6: Displaying the Matching features using FAST and BRIEF**

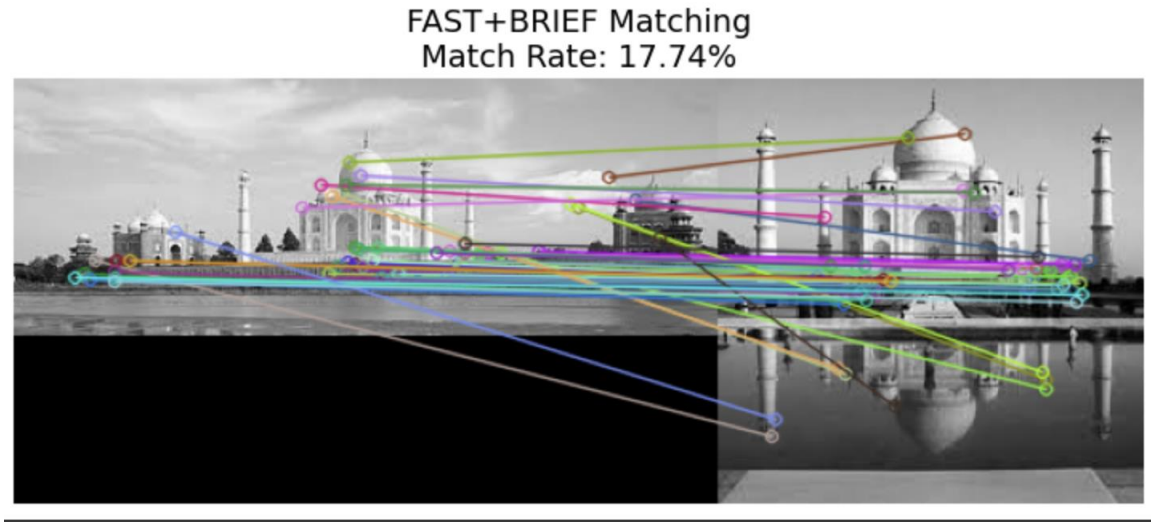



**Fig 7: Displaying the matching rate for FAST + BRIEF method**

**Results:**

| Method | Time (in seconds) | K points 1 | K points 2 | Matches | Matching rate (%) |
|---|---|---|---|---|---|
| **SIFT** | 0.13 | 238 | 478 | 80 | 33.61 |
| **ORB** | 0.21 | 325 | 424 | 100 | 30.77 |
| **FAST + BRIEF** | 0.16 | 840 | 1095 | 149 | 17.77 |

**Table 1: Representing the results for different methods and their respective rates**

**Table 1** shows the results of three different methods used for keypoint detection and matching: SIFT, ORB, and FAST + BRIEF. Each method is tested based on how long it takes to process the images, how many keypoints are detected, how many matches are found between the images, and the overall matching rate (the percentage of matches compared to the total number of possible matches).

- **SIFT** takes 0.13 seconds to process the images. It detects 238 keypoints in the first image and 478 in the second image. It finds 80 matches, giving it a matching rate of 33.61%. This means about one-third of the keypoints match between the two images. SIFT offers a good balance of speed and accuracy in detecting and matching keypoints.
- **ORB** takes a bit longer (0.21 seconds) than SIFT but detects more keypoints, with 325 in the first image and 424 in the second image. ORB finds 100 matches, but the matching rate is 30.77%, which is slightly lower than SIFT's matching rate. ORB is known for being faster than methods like SIFT but still provides decent results.
- **FAST + BRIEF** is the fastest method, taking just 0.16 seconds. It detects a large number of keypoints—840 in the first image and 1095 in the second. However, it only finds 149 matches, which results in a matching rate of 17.77%. Despite detecting a high number of keypoints, it has the lowest matching rate, which suggests that while many keypoints are found, fewer are actually useful for matching.

In short, each method has its strengths and weaknesses. SIFT gives the best matching rate, ORB offers a faster and still effective solution, and FAST + BRIEF detects the most key points but provides fewer accurate matches.

**Question 3:** After calibration, visualize the reprojection of the chessboard corners on one of the original images. Measure the reprojection error and provide an analysis of its significance.

**Deliverables:**

- Provide the camera calibration code.
- Submit the intrinsic matrix, distortion coefficients, and a report explaining the significance of reprojection error and how calibration affects real-world applications like 3D reconstruction.

**Input:** A series of images containing a calibration pattern (chessboard grid, for example).

**Expected Output:**

- Undistorted Image: You'll get an undistorted version of the input image showing how the calibration corrected the image distortion.
- Reprojection Error: A printed value indicating how well the calibration process worked.

**Solution:**

**Code:** https://colab.research.google.com/drive/1VmIMTF5IfmBLz3pdcbP3M8-wdAg2iLo-?usp=sharing

**Camera Calibration:**

It is the process of determining a camera's internal settings (like focal length and lens distortion) to map real-world objects onto images accurately.

**Objectives of Camera Calibration:**

The objective of camera calibration is to determine the **intrinsic parameters** of a camera, which include:

1) **Focal Lengths** $(f_x \text{ and } f_y)$: Measure how strongly the camera focuses light to create an image.
2) **Principal Point** $(c_x \text{ and } c_y)$: The optical center of the image, which is ideally the point where light rays converge on the sensor.
3) **Distortion Coefficients**: Parameters that account for lens distortion, which can alter the image geometry.

These parameters allow us to undistort images, making them geometrically accurate for real-world measurements, 3D reconstruction, or object recognition.

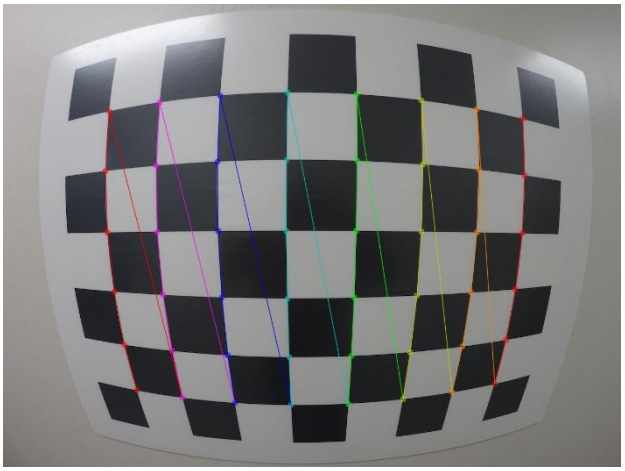
**Reprojection Error Analysis:**

It measures how well this calibration matches reality by comparing actual image points with projected points after calibration.

**Input Images**: The calibration images can be accessed https://drive.google.com/drive/folders/11KOjbKwJ2VO_-qtMoZwYGfDtlitIHocH?usp=sharing

**Sample Image of Corner Detection**

The sample image displays the detected corners on a chessboard, a critical step for calibration accuracy.



**Intrinsic Parameters**

1. **Intrinsic Matrix K:**
   The **Intrinsic Matrix** K encodes the focal lengths and principal point, relating the 3D coordinates in the camera frame to 2D coordinates in the image.

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

$Where\ f_x, f_y\ are\ Focal\ lengths\ in\ the\ x\ and\ y\ directions.$

$c_x, c_y\ are\ Coordinates\ of\ the\ principal\ point.$

In this case:

intrinsic_matrix: [[560.27092951 0 651.26167835] [0 561.32074335 499.05657532] [0 0 1]]

2. **Distortion Coefficients:**
   Distortion from the lens can be **radial** or **tangential**:
   - **Radial distortion** causes circular pattern deformation.
   - **Tangential distortion** results from slight misalignment of the lens.

   The complete set of distortion coefficients is typically represented as [k1,k2,p1,p2,k3]

   In this case:

   distortion_coeffs = [−0.2329 0.0617 −0.000014 0.000038 −0.00754]

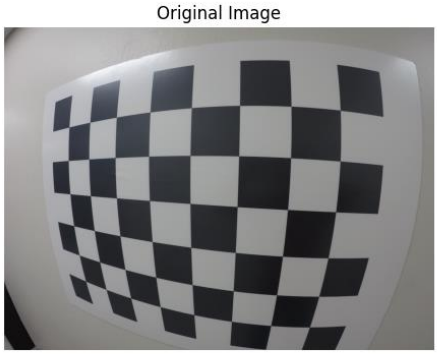
3. **Reprojection Error:**
   Reprojection error quantifies the calibration accuracy by comparing observed corner points with projected corner points after calibration.

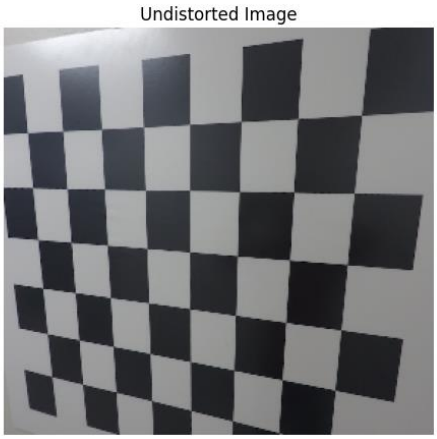
$$Reprojection\ Error = \frac{1}{N}\sum_{i=1}^{N} \|Observed\ points - projected\ points\|$$

This error serves as a key indicator of calibration quality. A lower reprojection error indicates more accurate intrinsic parameters. Generally, an error below 1 pixel is considered a good calibration, a higher error suggests potential calibration issues.

In this case: Mean Reprojection Error: 0.07733785696583968

**Results:** Sample output for original image and its corresponding undistorted image.




**Real-World Application:**

- **3D Reconstruction**: Accurate camera calibration ensures precise 3D scene reconstruction by transforming image coordinates to real-world coordinates, essential for fields like robotics and augmented reality (AR).
- **Object Detection and Recognition**: Correcting distortion and calibrating the camera improves feature extraction and recognition accuracy in applications such as facial recognition, autonomous driving, and surveillance.
- **Measurement Accuracy**: In fields like metrology or medical imaging, calibration is essential for accurately converting pixel measurements to real-world units.

**Question 4:** Uncalibrated Stereo and Fundamental Matrix Estimation.

When working with uncalibrated stereo cameras, the relationship between corresponding points in two images is captured using the fundamental matrix.

**Task:**

1. Given an uncalibrated stereo pair, estimate the fundamental matrix using matched feature points (use the 8-point or 7-point algorithm).
2. Use RANSAC to filter out outliers and improve the accuracy of the estimated fundamental matrix.
3. Visualize the epipolar lines on both images and discuss the relationship between corresponding points on the lines.

**Deliverables:**

- Implement fundamental matrix estimation and epipolar line visualization in Python.
- Provide the code, visualizations of the epipolar lines, and a short report explaining the process and the significance of the fundamental matrix in uncalibrated stereo.

**Input:** Two uncalibrated images of the same scene (stereo pair).

**Expected Output:**

- Fundamental Matrix: A matrix printed as output, representing the relationship between the two images.
- Epipolar Lines: Visualization of the corresponding epipolar lines drawn on both images. These lines represent the geometric constraints between the corresponding points in the images.

**Solution:**

**Code:**
https://colab.research.google.com/drive/1ZMZdkCjz7WBWYKASEABSSuC5Kh07qaGH?usp=sharing

In uncalibrated stereo vision, we work with stereo images where intrinsic camera parameters are unknown, relying purely on point correspondences to establish the relationship between the images. The fundamental matrix is crucial in this context, as it allows us to relate image pairs without needing detailed camera calibration. This task involves estimating the fundamental matrix for an uncalibrated stereo pair, filtering outliers using RANSAC (Random Sample Consensus), and visualizing the epipolar lines on both images. The fundamental matrix encapsulates the geometric relationship between corresponding points in the two images, enabling us to define constraints on where points in one image map to lines in the other. This relationship is essential for applications such as 3D reconstruction and stereo vision.

**1. Fundamental Matrix:**

The **fundamental matrix F** is a 3x3 matrix that encapsulates the epipolar geometry of two views. If we have a point x in image 1 and a corresponding point x′ in image 2, they are related by:

$$x'^{T}.F.x = 0$$

Where:

- x and x' are homogeneous coordinates of points in the two images.
- F encapsulates the intrinsic and extrinsic parameters of the camera setup, even if uncalibrated.

**2. Epipolar Lines:**

The **epipolar line** for a point in one image is the line in the other image where the corresponding point must lie. Given the fundamental matrix F and a point x in image 1, the epipolar line in image 2 can be computed as:

$$l' = F.x$$

Similarly, the epipolar line for a point in image 2, corresponding to a point in image 1, is:

$$l = F^T.x'$$

**3. RANSAC Algorithm**

RANSAC is an iterative algorithm used to estimate a model from data that contains outliers. It works by repeatedly selecting a random subset of points, computing a model based on this subset, counting the inliers that fit the model, and keeping the model with the highest inlier count. This approach helps refine the fundamental matrix by eliminating outlier correspondences.

This epipolar geometry is essential in applications like 3D reconstruction, where the spatial layout of a scene is derived from multiple viewpoints, robotics, where stereo vision aids in navigation and obstacle avoidance, and object tracking, enabling the identification and tracking of moving objects across frames.

**Input Images**: The images can be accessed

https://drive.google.com/drive/folders/1nXYfeObecBH2l5Wo1N7eywFYf6tIT_tH?usp=sharing

**Results:**

1) **Fundamental Matrix Estimation**: In the initial estimation, the fundamental matrix was computed as:

$$\begin{bmatrix} 9.81 \times 10^{-7} & 5.36 \times 10^{-6} & -0.0017 \\ 2 \times 10^{-6} & 1.39 \times 10^{-5} & -0.0045 \\ -0.00044 & -0.003 & 1 \end{bmatrix}$$

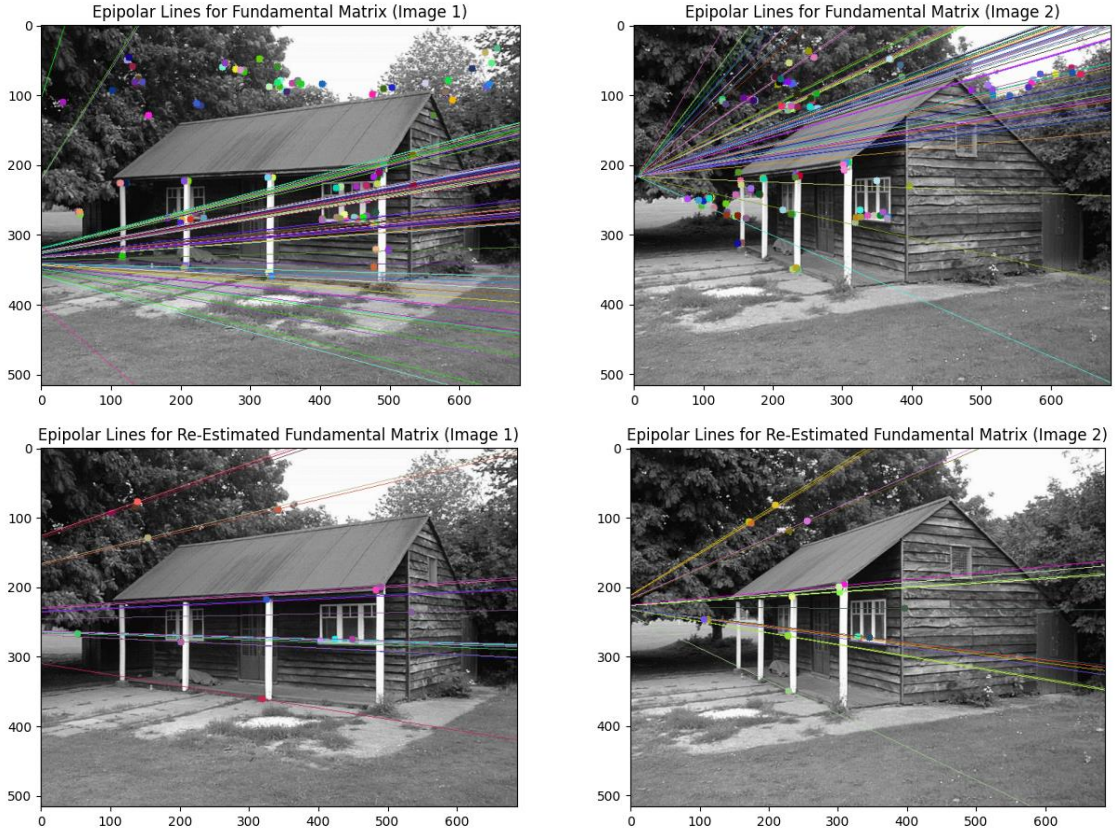
   The values close to zero in the first two rows suggest that the matrix may not have a good fit to the data, likely due to noise and outliers in the matched points.

2) **Re-Estimation of Fundamental Matrix:** After applying RANSAC, the fundamental matrix was re-estimated as:

$$\begin{bmatrix} 4.58 \times 10^{-7} & 2.19 \times 10^{-5} & -0.0053 \\ -9.35 \times 10^{-6} & 6.38 \times 10^{-6} & -0.0049 \\ 0.0021 & -0.00089 & 1 \end{bmatrix}$$

This matrix is more stable and accurate, with larger off-diagonal values indicating better alignment between the two images. The RANSAC algorithm effectively filtered out outliers, resulting in a more reliable fundamental matrix estimation.

3) **Epipolar Lines Visualization**:




The epipolar lines for both the initial and RANSAC-refined fundamental matrices were visualized on the images. The epipolar lines aligned closely with the actual point correspondences for the RANSAC-filtered matrix, confirming an improved fit.

4) **Reprojection Error**:

The reprojection error for the initial fundamental matrix was **170.38**, indicating a poor fit, likely due to noise and outliers. After applying RANSAC, the reprojection error dropped significantly to **0.90**, showing a much better fit and accurate alignment with the true epipolar geometry. This improvement demonstrates the effectiveness of RANSAC in refining the fundamental matrix estimation and making the model more reliable in predicting point positions and epipolar lines.