Computing the fundamental matrix between 2 images

H3Art
International School, Jinan University
Computer Science & Technology

Abstract

This project explores the computation of the fundamental matrix, a key component in understanding the geometric relationship between two views of the same scene. The implemented pipeline includes Harris corner detection, feature matching using SSD, fundamental matrix estimation via the 7-point and normalized 8-point algorithms, and robust refinement techniques such as RANSAC and non-linear optimization. Additional matches were identified along epipolar lines to enhance geometric consistency. Experimental evaluations on diverse image pairs demonstrated the effectiveness of the pipeline, achieving low residual errors and robust results across varying scene complexities. An interactive viewer was also developed, allowing users to validate epipolar geometry by dynamically rendering epipolar lines. The results emphasize the importance of robust estimation techniques and parameter tuning in achieving accurate and reliable epipolar geometry computation.

1 Introduction

In computer vision, the fundamental matrix plays a central role in understanding the geometric relationship between two images of the same scene captured from different viewpoints. It encapsulates the epipolar geometry, a powerful constraint that governs the correspondence of points across two images. The fundamental matrix is widely used in tasks such as 3D reconstruction, camera calibration, stereo vision, and motion estimation.

The accurate computation of the fundamental matrix is a challenging task due to noise, outliers, and variations in scene geometry. Traditional methods rely on matching points across images and solving systems of equations to estimate the matrix. However, real-world scenarios introduce complexities, such as repeated patterns, occlusions, and perspective distortions, which make robust estimation techniques crucial.

This project focuses on implementing and analyzing algorithms for the automatic computation of the fundamental matrix. The approach involves detecting interest points in both images using the Harris corner detector, matching these points based on a similarity measure (Sum of Squared Differences, SSD), and using various algorithms to estimate and refine the fundamental matrix. Key steps include:

- Feature detection and matching using Harris corner detection and SSD.
- Estimation of the fundamental matrix using the 7-point and normalized 8-point algorithms.

- Robust estimation via RANSAC to eliminate outliers.
- Non-linear optimization to minimize symmetrical epipolar distance.
- Searching for additional matches along epipolar lines to enhance accuracy.

The primary objectives of this work are implementing fundamental matrix computation and evaluating the performance of the implemented algorithms through visualizations and residual error analysis. Experimental results are presented using a diverse set of image pairs, including public data and personally captured scenes, ensuring comprehensive validation of the approach. Finally I implement a small viewer that allows clicking a point in one of the images and having the corresponding epipolar lines appear in both images.

2 Methods

This section describes the approach used to compute the fundamental matrix, including feature detection, feature matching, and the various algorithms implemented to estimate and refine the fundamental matrix. The methods used aim to ensure robustness and accuracy in the presence of noise, outliers, and varying scene geometry.

2.1 Overview of Approach

The process for computing the fundamental matrix consists of the following steps:

1. **Harris Corner Detection**: Detect interest points in both images to ensure distinct, reliable features.
2. **Feature Matching with SSD**: Match features between the two images using Sum of Squared Differences (SSD) as the similarity measure.
3. **Fundamental Matrix Estimation**:
    - Use the **7-point algorithm** when exactly 7 matching points are available.
    - Use the **Normalized 8-point algorithm** when 8 or more points are available to estimate the initial fundamental matrix.
4. **RANSAC**: Improve robustness by eliminating outliers.
5. **Non-linear Optimization**: Refine the fundamental matrix to minimize symmetrical epipolar distance.
6. **Matching Along Epipolar Lines**: Search for additional matching points along epipolar lines to validate and improve the results.

The following subsections describe each step of the process in detail.

2.2 Harris Corner Detection

Harris corner detection was used to identify interest points in the images. The detector identifies points with strong gradients in both directions, making them well-suited for matching.

The Harris response R is computed as:

$$R = \det(M) - k(\text{trace}(M))^2,$$

where M is the second-moment matrix:

$$M = \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix},$$

with I_x and I_y being image gradients in the x and y directions. k is a sensitivity parameter, typically set to 0.04.

Interest points are selected by thresholding R values and performing non-maximum suppression to ensure distinct keypoints.

2.3 Feature Matching with SSD

To establish correspondences between the two images, the features extracted from the Harris corner points are matched using Sum of Squared Differences (SSD). Given two feature vectors f_1 and f_2 of length N, the SSD is defined as:

$$\text{SSD}(f_1, f_2) = \sum_{i=1}^{N} (f_1[i] - f_2[i])^2.$$

Lower SSD values indicate higher similarity between feature vectors. Matches are further filtered using a ratio test, rejecting matches where the ratio of the nearest neighbor distance to the second-nearest neighbor distance exceeds a threshold.

2.4 7-point Algorithm

The 7-point algorithm is used to compute the fundamental matrix F when exactly 7 point correspondences are available. The algorithm solves the homogeneous system:

$$A\mathbf{f} = 0,$$

where A is a 7×9 matrix formed from the point correspondences, and $\mathbf{f}$ is the vectorized form of F.

The algorithm produces two candidate matrices F_1 and F_2, and the final solution is a linear combination:

$$F = \lambda F_1 + (1 - \lambda) F_2,$$

where λ is determined by enforcing $\det(F) = 0$.

2.5 Normalized 8-point Algorithm

The normalized 8-point algorithm improves numerical stability by normalizing the coordinates of the matching points. Normalization ensures the centroid of the points is at the origin and the average distance from the origin is $\sqrt{2}$. The

normalization transformation is:

$$T = \begin{bmatrix} \frac{1}{s_x} & 0 & -\frac{\mu_x}{s_x} \\ 0 & \frac{1}{s_y} & -\frac{\mu_y}{s_y} \\ 0 & 0 & 1 \end{bmatrix},$$

where μ_x, μ_y are the centroid coordinates, and s_x, s_y are scale factors.

After normalization, the 8-point algorithm solves:

$$A\mathbf{f} = 0,$$

where A is an $N \times 9$ matrix (with $N \geq 8$) constructed from the normalized coordinates. The solution is obtained via Singular Value Decomposition (SVD), and the rank-2 constraint is enforced by setting the smallest singular value of F to zero.

2.6 RANSAC

RANSAC (Random Sample Consensus) is used to robustly estimate the fundamental matrix by iteratively:

1. Randomly selecting a minimal subset of points (7 or 8 points).
2. Computing the fundamental matrix using the selected points.
3. Evaluating the inliers, defined as points where the symmetrical epipolar distance is below a threshold.

The symmetrical epipolar distance for a point correspondence $(\mathbf{x}_1, \mathbf{x}_2)$ is given by:

$$d = \frac{\mathbf{x}_2^\top F \mathbf{x}_1}{\sqrt{(F\mathbf{x}_1)_1^2 + (F\mathbf{x}_1)_2^2} + \sqrt{(F^\top \mathbf{x}_2)_1^2 + (F^\top \mathbf{x}_2)_2^2}}.$$

The model with the highest number of inliers is selected as the best estimate.

2.7 Non-linear Optimization

To further refine the fundamental matrix, a non-linear optimization process minimizes the symmetrical epipolar distance over all inliers. The optimization problem is formulated as:

$$\min_F \sum_{i=1}^{N} (d_i)^2,$$

where d_i is the symmetrical epipolar distance for the i-th correspondence. The optimization is performed using MATLAB's `lsqnonlin` function.

2.8 Matching Along Epipolar Lines

Additional matches are identified by exploiting the epipolar geometry. For a point $\mathbf{x}_1$ in the first image, its corresponding epipolar line in the second image is:

$$l_2 = F\mathbf{x}_1.$$

Similarly, for a point $\mathbf{x}_2$ in the second image, the epipolar line in the first image is:

$$l_1 = F^\top \mathbf{x}_2.$$

For every unmatched point in the first image, a search is performed along its corresponding epipolar line in the second image. Matches are identified by comparing image patches using SSD.

2.9 Interactive Epipolar Line Viewer

To facilitate result validation and geometric consistency analysis, an interactive viewer was implemented. This tool allows users to click on a point in one image and view the corresponding epipolar lines in both images. The viewer highlights the clicked point and dynamically renders the corresponding epipolar lines. Figure 1 shows an example of the viewer in action, demonstrating the epipolar lines for a selected point in Image Pair.

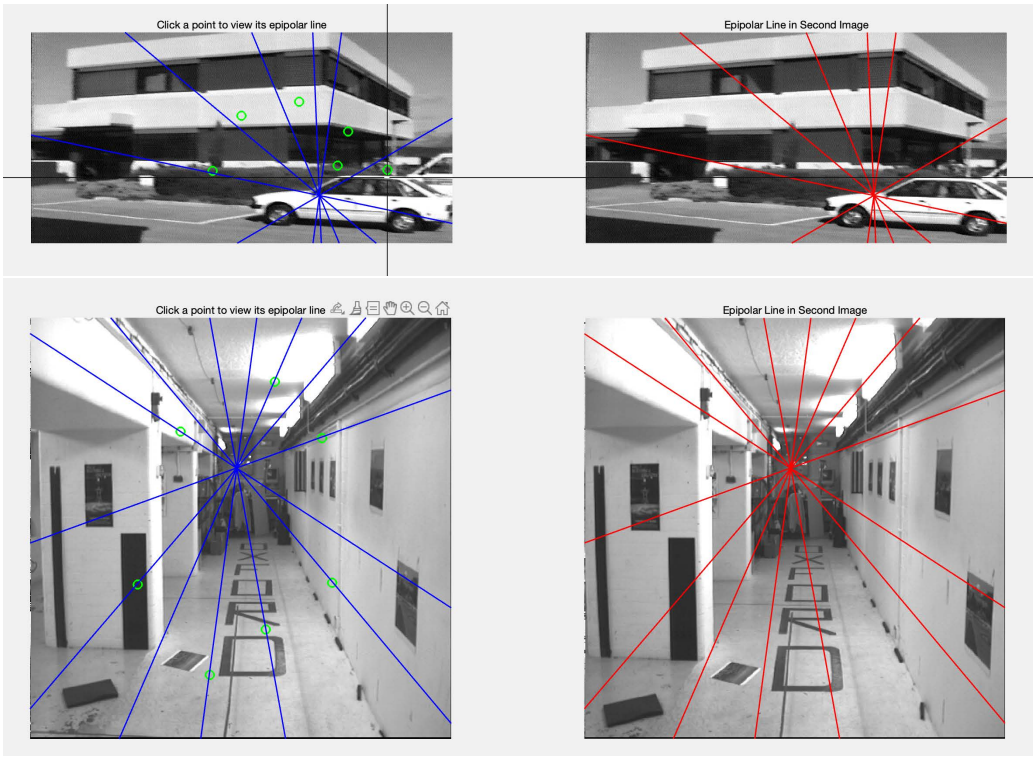


Figure 1: Interactive epipolar line viewer demonstrating line alignment for a clicked point in Image Pair.

3 Data

In this project, four pairs of images were used to evaluate the performance of the implemented algorithms for fundamental matrix computation. These image pairs were selected to ensure diversity in content and structure, the details of the data are as follows:

Three pairs of images were sourced from the UNC data for fundamental matrix computation. The data includes:

(a) First view of street scene (b) Second view of street scene

(c) First view of corridor scene (d) Second view of corridor scene

(e) First view of building scene (f) Second view of building scene

(g) First view of teaching building (h) Second view of teaching building

Figure 2: Experimental image pairing.

- **Street Scene**: A modern street view captured at INRIA Sophia Antipolis, featuring a building and a parked car. The scene contains both structural edges and textured surfaces, providing varied feature points for matching.
- **Corridor Scene**: Captured by the Oxford Visual Geometry Group, this scene features repetitive patterns along a corridor, offering distinct vanishing points and depth variations to challenge the robustness of feature detection.
- **Building Scene**: Another capture by the Oxford Visual Geometry Group, showcasing the facade of a historical building with intricate architectural details, such as edges, windows, and textures.

The fourth pair of images was captured at the Panyu Campus of Jinan University. The scene features the facade of a teaching building, characterized by geometric regularities such as windows, walls, and vegetation. These images provide a real-world context for testing the implemented algorithms.

The dataset combines indoor and outdoor scenes, including urban environments, structured architectural elements, and natural textures. The inclusion of scenes with varying depths, lighting conditions, and repetitive patterns ensures a comprehensive assessment of feature detection, matching, and geometric computation. For convenience, I will simply refer to these four image pairs as image pair 1, 2, 3 and 4 in the following sections.

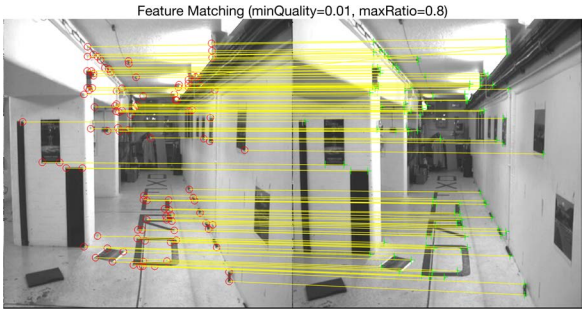
4 Experimental Results

This section presents the experimental results for computing the fundamental matrix using various algorithms and parameter experiments. The results are evaluated using the symmetrical epipolar residual error, and visualizations are provided to illustrate the matching points, epipolar lines, and the impact of different algorithms and parameters.

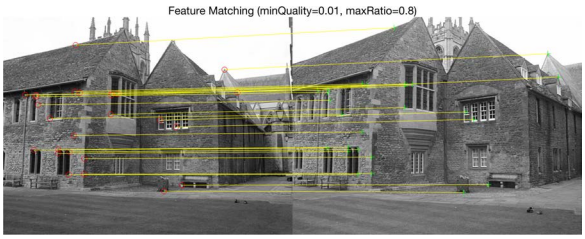
4.1 Feature Matching and Results

The initial step involves feature detection and matching using Harris corner detection and the SSD (Sum of Squared Differences) metric. Figure 3 shows the matched points for all four images, highlighting detected features and their corresponding matches.

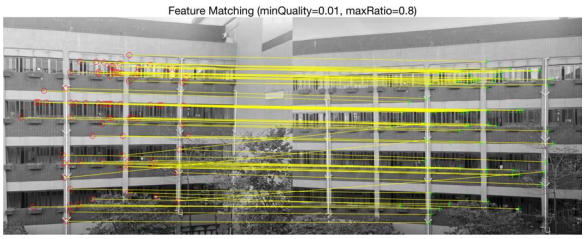
(a) Image Pair 1



(b) Image Pair 2



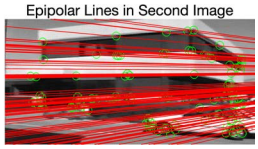
(c) Image Pair 3



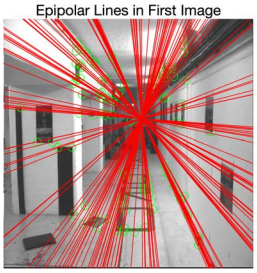
(d) Image Pair 4

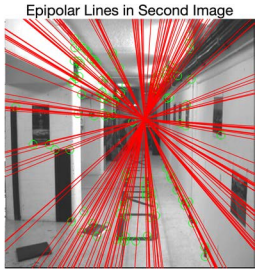
Figure 3: Feature matching results for all images (MinQuality = 0.01, MaxRatio = 0.8).

The normalized 8-point algorithm was applied to compute the initial fundamental matrix when sufficient matches (≥ 8) were available. For cases with exactly 7 matches, the 7-point algorithm was used. Figure 4 shows the epipolar lines computed using the initial fundamental matrix for all images.

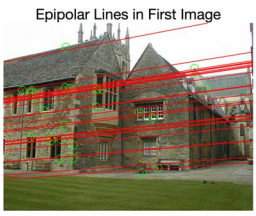


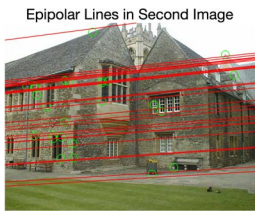
(a) Image Pair 1



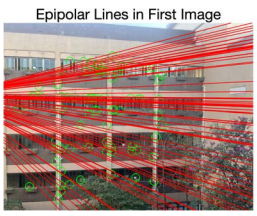


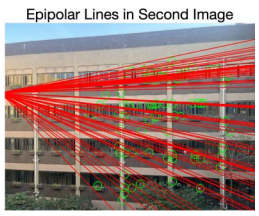
(b) Image Pair 2





(c) Image Pair 3

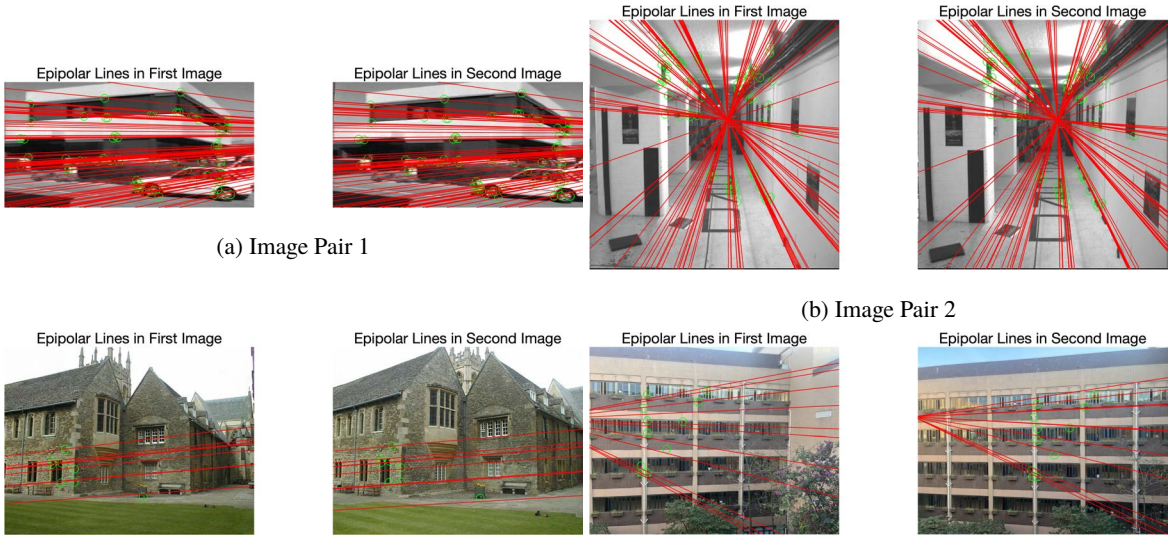




(d) Image Pair 4

Figure 4: Epipolar lines after fundamental matrix computation using the normalized 8-point algorithm.

Then RANSAC was applied to eliminate outliers from the matched points and recompute the fundamental matrix. Figure 5 shows the epipolar lines and inliers after applying RANSAC.



(a) Image Pair 1

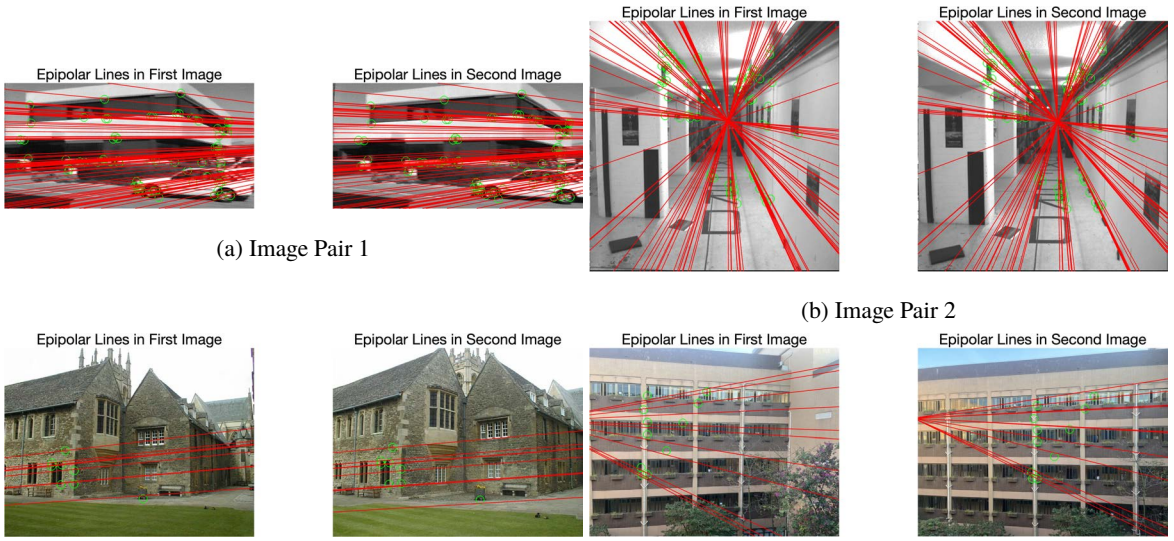
(b) Image Pair 2

(c) Image Pair 3

(d) Image Pair 4

Figure 5: Epipolar lines after RANSAC for outlier removal.

After RANSAC, the non-linear optimization was used to refine the fundamental matrix further. Figure 6 shows the epipolar lines after optimization.



(a) Image Pair 1

(b) Image Pair 2

(c) Image Pair 3

(d) Image Pair 4

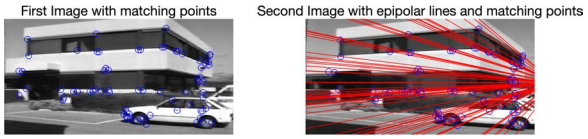
Figure 6: Epipolar lines after non-linear optimization of the fundamental matrix.

The residual errors for all images across different stages are summarized in Table 1. This includes errors from the initial computation using the normalized 8-point algorithm, after RANSAC outlier removal, and after non-linear optimization.

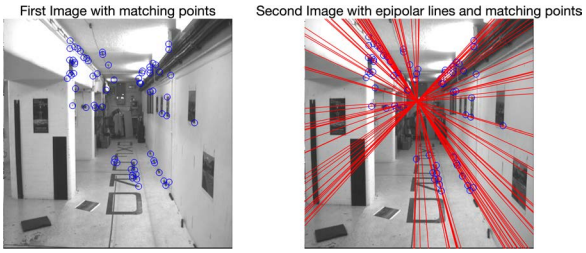
Table 1: Residual Errors for Different Images Across Stages.

Image Pair	Initial Error	RANSAC Error	Optimized Error
Image 1	2.403	0.45151	0.45151
Image 2	1.9937	0.39688	0.28828
Image 3	15.675	0.42992	0.379
Image 4	21.8654	0.41651	0.22299

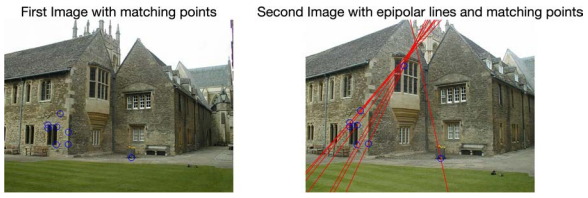
Finally, additional matches were identified along the computed epipolar lines. Figure 7 demonstrates the new matches found along the epipolar lines.



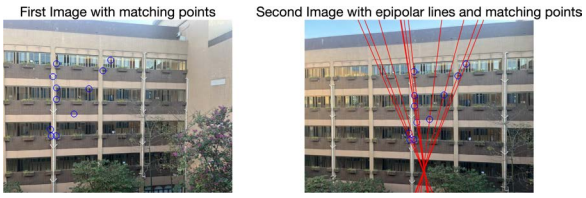
(a) Image Pair 1



(b) Image Pair 2



(c) Image Pair 3



(d) Image Pair 4

Figure 7: Matching points identified along the epipolar lines for all image pairs.

The results demonstrate that the implemented algorithms perform well across different types of image pairs. The normalized 8-point algorithm provides a reliable initial estimate of the fundamental matrix, though scenes with higher complexity, such as Image 3 and 4, exhibit higher initial residual errors. The application of RANSAC significantly improves the results by effectively removing outliers, especially in images with noisy or repetitive patterns. Non-linear optimization further refines the fundamental matrix, achieving the lowest residual errors across all images. Additional matches identified along the epipolar lines validate the geometric consistency of the computed matrices, enhancing the overall robustness of the pipeline.

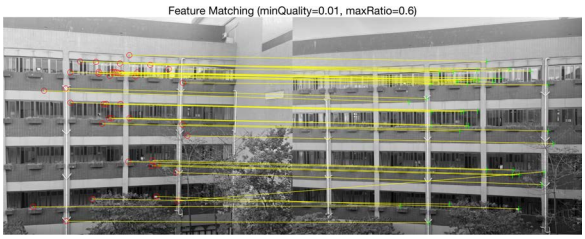
4.2 Parameter Experiments

To analyze the impact of feature detection (`MinQuality`) and matching (`MaxRatio`) parameters on the fundamental matrix computation, experiments were conducted across four image pairs. Table 2 summarizes the symmetrical epipolar residual errors for different parameter combinations across all images.

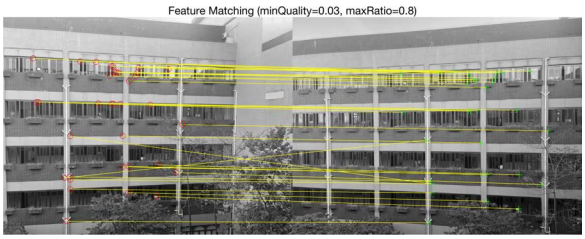
Table 2: Residual Errors for Parameter Experiments Across Four Image Pairs.

Parameters	Image Pair 1	Image Pair 2	Image Pair 3	Image Pair 4
MinQuality = 0.01, MaxRatio = 0.6	1.6973	2.1939	14.7897	22.6602
MinQuality = 0.01, MaxRatio = 0.8	2.403	1.9937	15.675	21.8654
MinQuality = 0.01, MaxRatio = 1.0	18.5277	38.6132	15.4171	77.1161
MinQuality = 0.03, MaxRatio = 0.6	2.0166	2.7301	11.8347	17.864
MinQuality = 0.03, MaxRatio = 0.8	7.129	2.6742	17.9535	28.0323
MinQuality = 0.03, MaxRatio = 1.0	7.0984	2.2684	17.9535	103.287
MinQuality = 0.05, MaxRatio = 0.6	1.8018	2.7211	N/A	54.3515
MinQuality = 0.05, MaxRatio = 0.8	1.7985	2.6446	35.1261	74.3608
MinQuality = 0.05, MaxRatio = 1.0	1.7966	2.7176	35.1261	59.991

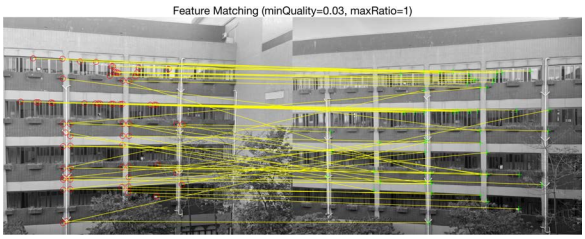
Due to space constraints, only results for Image Pair 4 are visualized in Figure 8. These results demonstrate the effect of parameter variations on feature matching.



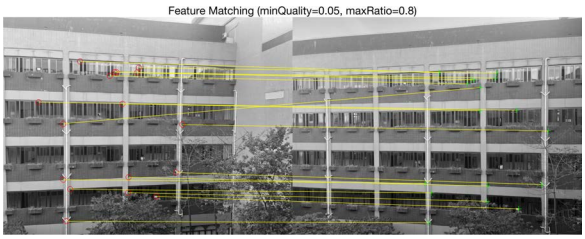
(a) MinQuality = 0.01, MaxRatio = 0.6



(b) MinQuality = 0.03, MaxRatio = 0.8



(c) MinQuality = 0.03, MaxRatio = 1.0



(d) MinQuality = 0.05, MaxRatio = 0.8

Figure 8: Feature matching results for Image Pair 4 under different parameter settings.

The parameter experiments highlight the sensitivity of feature matching and fundamental matrix estimation to parameter settings. Lower values of `MinQuality` and `MaxRatio` generally lead to fewer but more robust matches, resulting in lower residual errors. Conversely, higher values increase the number of matches but often introduce more outliers, leading to higher residual errors. The results also demonstrate that complex scenes (e.g., Image Pair 4) are more sensitive to parameter variations compared to simpler scenes.

4.3 Discussion of Results

The experiments highlight the effectiveness of the implemented pipeline for fundamental matrix estimation. The normalized 8-point algorithm provided a reliable initial estimate, but its performance depended heavily on scene complexity.

Simpler scenes, such as Image Pair 1 and 2, exhibited lower residual errors (e.g., 2.403 and 1.9937), whereas more complex scenes like Image Pair 3 and 4 resulted in higher errors (e.g., 15.675 and 21.8654), underscoring its sensitivity to noise and outliers.

RANSAC proved effective in refining results by removing outliers, achieving significant error reductions across all image pairs. For example, in Image Pair 4, the residual error dropped from 21.8654 to 0.41651, demonstrating RANSAC's robustness in noisy settings. However, its reliance on initial feature quality highlights the importance of robust feature matching.

Non-linear optimization further refined the fundamental matrix, achieving the lowest errors in all cases. This step minimized epipolar distance, reducing errors even in challenging scenes (e.g., Image Pair 4 improved to 0.22299). These results emphasize the importance of optimization for fine-tuning.

Parameter experiments revealed the trade-offs between matching quantity and quality. Lower `MaxRatio` values generally produced more robust results, as seen in Image Pair 4, where reducing `MaxRatio` from 1.0 to 0.6 decreased the error from 77.1161 to 22.6602. However, higher `MaxRatio` increased the number of matches but introduced more outliers, necessitating additional refinement.

Finally, additional matches along epipolar lines validated the computed matrices, further demonstrating geometric consistency and pipeline robustness. Overall, each stage contributes to handling scenes of varying complexity.

5 Conclusion

This project successfully implemented a robust pipeline for fundamental matrix computation and epipolar geometry validation using a combination of traditional and robust estimation techniques. The pipeline encompassed feature detection and matching, outlier removal with RANSAC, non-linear optimization, and additional matching along epipolar lines, which collectively enhanced the accuracy and reliability of the results.

Experimental evaluations demonstrated that:

- The normalized 8-point algorithm provided reliable initial estimates but required robust refinement techniques to handle noise and outliers effectively.
- RANSAC played a critical role in identifying inliers and improving residual errors, making the pipeline robust against challenging scenes with repetitive patterns and occlusions.
- Non-linear optimization further minimized symmetrical epipolar distances, achieving the most accurate results across all experiments, particularly in complex real-world scenarios.
- Parameter tuning was crucial for balancing the trade-off between the number of matches and their quality, with lower thresholds yielding fewer but more robust matches.

Moreover, the implementation of an interactive epipolar line viewer provided an intuitive way to validate results and understand the geometric relationships between image pairs. This tool showcased the practical applicability of the fundamental matrix and epipolar geometry in computer vision tasks.

The results highlight the versatility and robustness of the implemented pipeline across diverse image pairs, from structured indoor environments to complex outdoor scenes. Future work can extend this study by integrating advanced feature detection methods, such as SIFT or ORB, and exploring adaptive parameter selection strategies to further enhance performance in real-world applications.

6 Appendix: Source Code

6.1 main.m

```
clc;
clear;

% Load images
img1 = imread('./source/image_4_a.jpg');
img2 = imread('./source/image_4_b.jpg');

% Feature detection and matching
[matchedPoints1, matchedPoints2] = feature_matching(img1, img2);

% Check the number of matched points
if size(matchedPoints1, 1) < 7
    error('Not enough matched points. At least 7 matches are required.');
end

% Fundamental matrix computation
if size(matchedPoints1, 1) == 7
    % 7-point algorithm
    F_initial = seven_point_algorithm(matchedPoints1.Location, matchedPoints2.
        Location);
    visualize_epipolar_lines(F_initial, matchedPoints1, matchedPoints2, img1, img2, '
        7-point');
    disp('7-point algorithm result:');
elseif size(matchedPoints1, 1) >= 8
    % Normalized 8-point algorithm
    F_initial = normalized_eight_point(matchedPoints1.Location, matchedPoints2.
        Location);
    visualize_epipolar_lines(F_initial, matchedPoints1, matchedPoints2, img1, img2, '
        normalized_8-point');
    disp('Normalized 8-point algorithm result:');
else
    error('Not enough matched points. At least 7 matches are required.');
end

% Compute the initial residual error
residual_initial = compute_residual_error(F_initial, matchedPoints1, matchedPoints2);
disp(['Initial Residual Error: ', num2str(residual_initial), newline]);

% RANSAC
```

```
[F_ransac, inlierPoints1, inlierPoints2] = ransac_fundamental_matrix(...
    matchedPoints1.Location, matchedPoints2.Location, 2000, 1);
visualize_epipolar_lines(F_initial, inlierPoints1, inlierPoints2, img1, img2, 'RANSAC');
disp('RANSAC result:');
residual_ransac = compute_residual_error(F_ransac, inlierPoints1, inlierPoints2);
disp(['Residual Error (RANSAC): ', num2str(residual_ransac), newline]);

% Nonlinear optimization
F_optimized = nonlinear_optimization(F_ransac, inlierPoints1, inlierPoints2);
visualize_epipolar_lines(F_initial, inlierPoints1, inlierPoints2, img1, img2, 'optimized');
disp('Optimized Fundamental Matrix:');
residual_optimized = compute_residual_error(F_optimized, inlierPoints1, inlierPoints2);
disp(['Residual Error (Optimized): ', num2str(residual_optimized), newline]);

% Results
disp('Optimized Fundamental Matrix:');
disp(F_optimized);

% Find more matches along epipolar lines
find_matches_along_epilines(F_optimized, img1, img2, inlierPoints1, inlierPoints2);

% Parameter experimentation
experiment_parameters(img1, img2);

% Epipolar line visualization
interactive_epipolar_lines(F_optimized, img1, img2);
```

6.2 feature_matching.m

```
function [matchedPoints1, matchedPoints2] = feature_matching(img1, img2, minQuality, maxRatio)
% If minQuality or maxRatio are not provided, use default values
if nargin < 3
    minQuality = 0.01; % Default corner quality threshold
end
if nargin < 4
    maxRatio = 0.8; % Default matching ratio threshold
end

```

```
% Convert to grayscale images
if size(img1, 3) == 3
    img1 = rgb2gray(img1);
end
if size(img2, 3) == 3
    img2 = rgb2gray(img2);
end

% Detect feature points
points1 = detectHarrisFeatures(img1, 'MinQuality', minQuality, 'FilterSize', 5);
points2 = detectHarrisFeatures(img2, 'MinQuality', minQuality, 'FilterSize', 5);

% Extract feature descriptors
[features1, validPoints1] = extractFeatures(img1, points1);
[features2, validPoints2] = extractFeatures(img2, points2);

% Match feature points
indexPairs = matchFeatures(features1, features2, 'MaxRatio', maxRatio);
matchedPoints1 = validPoints1(indexPairs(:, 1));
matchedPoints2 = validPoints2(indexPairs(:, 2));

% Visualize the initial matched points
figureHandle = figure; % Save the figure handle
showMatchedFeatures(img1, img2, matchedPoints1, matchedPoints2, 'montage');
title(['Feature␣Matching␣(minQuality=', num2str(minQuality), ',␣maxRatio=', num2str(
    maxRatio), ')']);

% Dynamically generate the file name
fileName = sprintf('feature_matching_min%.1e_max%.1e', minQuality, maxRatio);

% Save as a PNG and PDF without white borders
outputDir = './temp_results/'; % Directory to save results
if ~exist(outputDir, 'dir')
    mkdir(outputDir); % Create the directory if it does not exist
end

% Save as a PNG file
exportgraphics(figureHandle, fullfile(outputDir, [fileName, '.png']), 'Resolution',
    600);

% Close the figure to avoid memory usage
close(figureHandle);
```

```
end
```

6.3 visualize_epipolar_lines.m

```
function visualize_epipolar_lines(F, matchedPoints1, matchedPoints2, img1, img2,
    methodName)
% Ensure the fundamental matrix and point sets are of double type
F = double(F);
if isa(matchedPoints1, 'cornerPoints')
    points1_h = [matchedPoints1.Location, ones(size(matchedPoints1.Location, 1), 1)];
    points2_h = [matchedPoints2.Location, ones(size(matchedPoints2.Location, 1), 1)];
elseif ismatrix(matchedPoints1)
    points1_h = [matchedPoints1, ones(size(matchedPoints1, 1), 1)];
    points2_h = [matchedPoints2, ones(size(matchedPoints2, 1), 1)];
else
    error('Unsupported data type for matched points.');
end

% Compute the epipolar lines
epiLines1 = (F' * points2_h')'; % Epipolar lines for points in image 2 in image 1
epiLines2 = (F * points1_h')'; % Epipolar lines for points in image 1 in image 2

% Create a figure and visualize the matched points and epipolar lines
figureHandle = figure;
subplot(1, 2, 1);
imshow(img1); hold on; title('Epipolar Lines in First Image');
plot(points1_h(:, 1), points1_h(:, 2), 'go'); % Plot the matched points
for i = 1:size(epiLines1, 1)
    lineEq = epiLines1(i, :);
    % Check the validity of the epipolar line equation
    if any(isnan(lineEq)) || all(lineEq(1:2) == 0)
        disp(['Invalid epipolar line for point ', num2str(i)]);
        continue;
    end
    linePoints = lineToBorderPoints(lineEq, size(img1));
    % Ensure linePoints is a 1x4 matrix
    if numel(linePoints) == 4
        % Plot the epipolar line
        line([linePoints(1), linePoints(3)], [linePoints(2), linePoints(4)], 'Color',
            'r', 'LineWidth', 0.5);
    else
        disp(['Invalid line points for epipolar line ', num2str(i)]);
```

```
    end
end
hold off;

subplot(1, 2, 2);
imshow(img2); hold on; title('Epipolar Lines in Second Image');
plot(points2_h(:, 1), points2_h(:, 2), 'go'); % Plot the matched points
for i = 1:size(epiLines2, 1)
    lineEq = epiLines2(i, :);
    % Check the validity of the epipolar line equation
    if any(isnan(lineEq)) || all(lineEq(1:2) == 0)
        disp(['Invalid epipolar line for point ', num2str(i)]);
        continue;
    end
    linePoints = lineToBorderPoints(lineEq, size(img2));
    % Ensure linePoints is a 1x4 matrix
    if numel(linePoints) == 4
        % Plot the epipolar line
        line([linePoints(1), linePoints(3)], [linePoints(2), linePoints(4)], 'Color',
            'r', 'LineWidth', 0.5);
    else
        disp(['Invalid line points for epipolar line ', num2str(i)]);
    end
end
hold off;

% Dynamically generate the file name
fileName = sprintf('epilines_%s', methodName);

% Save the file
outputDir = './temp_results/'; % Directory to save results
if ~exist(outputDir, 'dir')
    mkdir(outputDir); % Create the directory if it does not exist
end

% Save as a PNG file
savePath = fullfile(outputDir, [fileName, '.png']);
exportgraphics(figureHandle, savePath, 'Resolution', 600);

% Close the figure to avoid memory usage
close(figureHandle);

```

```
78 end
```

6.4 seven_point_algorithm.m

```
1  function F = seven_point_algorithm(points1, points2)
2  points1_h = [points1, ones(size(points1, 1), 1)];
3  points2_h = [points2, ones(size(points2, 1), 1)];
4  A = [points2_h(:,1).*points1_h(:,1), points2_h(:,1).*points1_h(:,2), points2_h(:,1),
       ...
5      points2_h(:,2).*points1_h(:,1), points2_h(:,2).*points1_h(:,2), points2_h(:,2),
           ...
6      points1_h(:,1), points1_h(:,2), ones(size(points1_h, 1), 1)];
7  [~, ~, V] = svd(A);
8  F1 = reshape(V(:,end-1), [3, 3]);
9  F2 = reshape(V(:,end), [3, 3]);
10 syms lambda;
11 F = lambda * F1 + (1 - lambda) * F2;
12 eq = det(F);
13 lambdas = double(vpasolve(eq, lambda));
14 F = lambdas(1) * F1 + (1 - lambdas(1)) * F2;
15 F = F / norm(F);
16 end
```

6.5 normalized_eight_point.m

```
1  function F = normalized_eight_point(points1, points2)
2  [points1_norm, T1] = normalize_points(points1);
3  [points2_norm, T2] = normalize_points(points2);
4  A = [points2_norm(:,1).*points1_norm(:,1), points2_norm(:,1).*points1_norm(:,2),
       points2_norm(:,1), ...
5      points2_norm(:,2).*points1_norm(:,1), points2_norm(:,2).*points1_norm(:,2),
           points2_norm(:,2), ...
6      points1_norm(:,1), points1_norm(:,2), ones(size(points1_norm, 1), 1)];
7  [~, ~, V] = svd(A);
8  F_norm = reshape(V(:,end), [3, 3]);
9  [U, S, V] = svd(F_norm);
10 S(3,3) = 0;
11 F_norm = U * S * V';
12 F = T2' * F_norm * T1;
13 end
14
```

```
function [points_norm, T] = normalize_points(points)
centroid = mean(points);
scale = sqrt(2) / mean(sqrt(sum((points - centroid).^2, 2)));
T = [scale, 0, -scale * centroid(1); 0, scale, -scale * centroid(2); 0, 0, 1];
points_h = [points, ones(size(points, 1), 1)];
points_norm_h = (T * points_h')';
points_norm = points_norm_h(:, 1:2);
end
```

6.6 ransac_fundamental_matrix.m

```
function [F_ransac, inlierPoints1, inlierPoints2] = ransac_fundamental_matrix(points1
    , points2, numIterations, distanceThreshold)
numPoints = size(points1, 1);
maxInliers = 0;
F_ransac = [];
points1_h = [points1, ones(numPoints, 1)];
points2_h = [points2, ones(numPoints, 1)];
for i = 1:numIterations
    randomIndices = randperm(numPoints, 8);
    samplePoints1 = points1(randomIndices, :);
    samplePoints2 = points2(randomIndices, :);
    F_candidate = normalized_eight_point(samplePoints1, samplePoints2);
    distances = compute_epipolar_distances(F_candidate, points1_h, points2_h);
    inliers = find(distances < distanceThreshold);
    numInliers = length(inliers);
    if numInliers > maxInliers
        maxInliers = numInliers;
        F_ransac = F_candidate;
        bestInliers = inliers;
    end
end
inlierPoints1 = points1(bestInliers, :);
inlierPoints2 = points2(bestInliers, :);
end

function distances = compute_epipolar_distances(F, points1, points2)
epiLines1 = (F' * points2')';
epiLines2 = (F * points1')';
dist1 = abs(sum(points1 .* epiLines1, 2)) ./ sqrt(epiLines1(:, 1).^2 + epiLines1(:,
    2).^2);
```

```
dist2 = abs(sum(points2 .* epiLines2, 2)) ./ sqrt(epiLines2(:, 1).^2 + epiLines2(:, 2).^2);
distances = dist1 + dist2;
end
```

6.7 nonlinear_optimization.m

```
function F_optimized = nonlinear_optimization(F, points1, points2)
% Ensure the fundamental matrix F is of double precision type
F = double(F);

% Ensure the point sets are of double precision type
points1 = double(points1);
points2 = double(points2);

% Convert the point sets to homogeneous coordinates
points1_h = [points1, ones(size(points1, 1), 1)];
points2_h = [points2, ones(size(points2, 1), 1)];

% Define the optimization objective function
fun = @(F_vec) compute_residual_error(reshape(F_vec, [3, 3]), points1_h, points2_h);

% Set optimization options
options = optimoptions('lsqnonlin', 'Display', 'off');

% Call lsqnonlin for optimization
F_vec = lsqnonlin(fun, F(:), [], [], options);

% The optimized fundamental matrix
F_optimized = reshape(F_vec, [3, 3]);
end

function residuals = compute_residual_error(F, points1, points2)
% Compute the symmetrical epipolar distance for points to epipolar lines
epiLines1 = (F' * points2')'; % Epipolar lines for points in image 2 in image 1
epiLines2 = (F * points1')'; % Epipolar lines for points in image 1 in image 2

% Compute the normalized distances
dist1 = abs(sum(points1 .* epiLines1, 2)) ./ sqrt(epiLines1(:, 1).^2 + epiLines1(:, 2).^2);
dist2 = abs(sum(points2 .* epiLines2, 2)) ./ sqrt(epiLines2(:, 1).^2 + epiLines2(:, 2).^2);
```

```

% Return the residuals
residuals = dist1 + dist2;
end
```

6.8 compute_residual_error.m

```
function residualError = compute_residual_error(F, matchedPoints1, matchedPoints2)
% Ensure the fundamental matrix and point sets are of type double
F = double(F);
if isa(matchedPoints1, 'cornerPoints')
    points1_h = [matchedPoints1.Location, ones(size(matchedPoints1.Location, 1), 1)];
    points2_h = [matchedPoints2.Location, ones(size(matchedPoints2.Location, 1), 1)];
elseif ismatrix(matchedPoints1)
    points1_h = [matchedPoints1, ones(size(matchedPoints1, 1), 1)];
    points2_h = [matchedPoints2, ones(size(matchedPoints2, 1), 1)];
else
    error('Unsupported data type for matched points.');
end

% Compute the symmetrical epipolar distance for the points to the epipolar lines
epiLines1 = (F' * points2_h')'; % Epipolar lines for points in image 2 in image 1
epiLines2 = (F * points1_h')'; % Epipolar lines for points in image 1 in image 2

% Distance calculation (normalized)
dist1 = abs(sum(points1_h .* epiLines1, 2)) ./ sqrt(epiLines1(:, 1).^2 + epiLines1(:,
    2).^2);
dist2 = abs(sum(points2_h .* epiLines2, 2)) ./ sqrt(epiLines2(:, 1).^2 + epiLines2(:,
    2).^2);

% Average residual of the symmetrical epipolar distance
residualError = mean(dist1 + dist2);

% Output the residual
disp(['Symmetrical Epipolar Residual Error: ', num2str(residualError)]);
end
```

6.9 find_matches_along_epilines.m

```
function find_matches_along_epilines(F, img1, img2, matchedPoints1, matchedPoints2)
% Check the type of matchedPoints1 and matchedPoints2
```

```
3  if isa(matchedPoints1, 'cornerPoints')
4      points1_h = [matchedPoints1.Location, ones(size(matchedPoints1.Location, 1), 1)];
5  elseif ismatrix(matchedPoints1)
6      points1_h = [matchedPoints1, ones(size(matchedPoints1, 1), 1)];
7  else
8      error('Unsupported data type for matched points.');
9  end
10
11 % Compute the epipolar lines in image 2
12 epiLines2 = points1_h * F';
13
14 % Create a side-by-side image display
15 figureHandle = figure;
16 subplot(1, 2, 1);
17 imshow(img1); title('First Image with matching points');
18 hold on;
19 plot(matchedPoints1(:, 1), matchedPoints1(:, 2), 'go', 'MarkerSize', 5, 'LineWidth',
       0.5); % Plot matching points in image 1
20 hold off;
21
22 subplot(1, 2, 2);
23 imshow(img2); title('Second Image with epipolar lines and matching points');
24 hold on;
25
26 for i = 1:size(epiLines2, 1)
27     % Get the current epipolar line equation
28     lineEq = epiLines2(i, :);
29
30     % Calculate the intersection points of the epipolar line with the image borders
31     linePoints = lineToBorderPoints(lineEq, size(img2));
32     % Plot the epipolar line
33     if numel(linePoints) == 4
34         line([linePoints(1), linePoints(3)], [linePoints(2), linePoints(4)], 'Color',
               'r', 'LineWidth', 0.5);
35     else
36         disp(['Invalid line points for epipolar line ', num2str(i)]);
37     end
38
39     % Find the matching point along the epipolar line
40     bestMatch = find_best_match_along_epiline(lineEq, img1, img2, matchedPoints1(i,
           :));
41     if ~isempty(bestMatch)
```

```
        % Mark the original point in image 1
        subplot(1, 2, 1);
        hold on;
        plot(matchedPoints1(i, 1), matchedPoints1(i, 2), 'bo', 'MarkerSize', 5, '
            LineWidth', 0.5);

        % Plot the matching point in image 2
        subplot(1, 2, 2);
        plot(bestMatch(1), bestMatch(2), 'bo', 'MarkerSize', 5, 'LineWidth', 0.5);
    end
end
hold off;

% Save the image
outputDir = './temp_results/'; % Directory to save results
if ~exist(outputDir, 'dir')
    mkdir(outputDir); % Create the directory if it does not exist
end

% Save as a PNG file
savePath = fullfile(outputDir, 'matches_along_epilines.png');
exportgraphics(figureHandle, savePath, 'Resolution', 600);

% Close the figure to avoid memory usage
close(figureHandle);

end

function bestMatch = find_best_match_along_epiline(lineEq, img1, img2, point1)
% Define the size of the search window
patchSize = 5; % Image patch size

% Get the image patch around point 1
x1 = round(point1(1));
y1 = round(point1(2));
patch1 = img1(max(1, y1-patchSize):min(size(img1, 1), y1+patchSize), ...
    max(1, x1-patchSize):min(size(img1, 2), x1+patchSize));

% Initialize the best match
bestScore = inf; % The smaller the better
bestMatch = [];

```

```
% Traverse points along the epipolar line in image 2 to find the matching point
for x2 = 1:size(img2, 2)
    % Calculate the y2 value
    y2 = -(lineEq(1) * x2 + lineEq(3)) / lineEq(2);
    y2 = round(y2); % Round to the nearest integer
    if y2 < 1 || y2 > size(img2, 1) % Check if the point is within the image
        boundaries
        continue;
    end

    % Get the image patch around point 2
    patch2 = img2(max(1, y2-patchSize):min(size(img2, 1), y2+patchSize), ...
        max(1, x2-patchSize):min(size(img2, 2), x2+patchSize));

    % Calculate the similarity (SSD or NCC)
    if size(patch1) == size(patch2) % Ensure the two image patches are the same size
        score = sum((double(patch1(:)) - double(patch2(:))).^2); % SSD
        if score < bestScore
            bestScore = score;
            bestMatch = [x2, y2];
        end
    end
end
end
```

6.10 experiment_parameters.m

```
function experiment_parameters(img1, img2)
% Define the range of experimental parameters
minQualityValues = [0.01, 0.03, 0.05];
maxRatioValues = [0.6, 0.8, 1.0];

% Iterate through all combinations of parameters
for minQuality = minQualityValues
    for maxRatio = maxRatioValues
        disp(['Experimenting␣with␣MinQuality:␣', num2str(minQuality), ...
            '␣and␣MaxRatio:␣', num2str(maxRatio)]);

        % Call the feature detection and matching function
        [matchedPoints1, matchedPoints2] = feature_matching(img1, img2, minQuality,
            maxRatio);

```

```
        % Check the number of matched points
        numMatches = size(matchedPoints1, 1);
        if numMatches < 7
            % Does not meet the requirement for the 7-point algorithm
            disp(['Not enough matches for fundamental matrix estimation. At least 7 matches are required.', newline]);
            continue;
        elseif numMatches == 7
            % Call the 7-point algorithm
            disp('Using the 7-point algorithm for fundamental matrix estimation.');
            F7 = seven_point_algorithm(matchedPoints1.Location, matchedPoints2.Location);
            residual7 = compute_residual_error(F7, matchedPoints1, matchedPoints2);
            disp(['Residual Error (7-point): ', num2str(residual7), newline]);
        elseif numMatches >= 8
            % Call the normalized 8-point algorithm
            disp('Using the normalized 8-point algorithm for fundamental matrix estimation.');
            F8 = normalized_eight_point(matchedPoints1.Location, matchedPoints2.Location);
            residual8 = compute_residual_error(F8, matchedPoints1, matchedPoints2);
            disp(['Residual Error (8-point): ', num2str(residual8), newline]);
        end
    end
end
end
```

6.11 interactive_epipolar_lines.m

```
function interactive_epipolar_lines(F, img1, img2)
% Create a side-by-side layout with two subplots
figure;
subplot(1, 2, 1); % Subplot 1
imshow(img1); title('Click a point to view its epipolar line');
subplot(1, 2, 2); % Subplot 2
imshow(img2); title('Epipolar Line in Second Image');

% Infinite interactive loop
while true
    % User clicks a point in the first image
    subplot(1, 2, 1);
    [x, y, button] = ginput(1); % Get the clicked point and mouse button
```

```
    % Check if "Enter" or "Esc" key is pressed to exit
    if isempty(x) || button == 27 % Press Enter or Esc to exit
        disp('Exiting␣interactive␣mode.');
        break;
    end

    % Convert to homogeneous coordinates
    clickedPoint = [x, y, 1];

    % Compute the corresponding epipolar line in image 2
    epiLine2 = clickedPoint * F; % Epipolar line for the clicked point in image 2
    borderPts2 = lineToBorderPoints(epiLine2, size(img2)); % Intersection points of
        the epipolar line with the image 2 borders

    % Compute the epipolar line in image 1 (mapping from image 2 to image 1)
    epiLine1 = (F' * clickedPoint')'; % Transpose to ensure it's a 1x3 row vector
    borderPts1 = lineToBorderPoints(epiLine1, size(img1)); % Intersection points of
        the epipolar line with the image 1 borders

    % Plot the clicked point and epipolar line in image 1
    subplot(1, 2, 1);
    hold on;
    plot(x, y, 'go', 'MarkerSize', 10, 'LineWidth', 2); % Mark the clicked point
    line(borderPts1(:, [1, 3])', borderPts1(:, [2, 4])', 'Color', 'b', 'LineWidth',
        1.5); % Draw the epipolar line
    hold off;

    % Plot the corresponding epipolar line in image 2
    subplot(1, 2, 2);
    hold on;
    line(borderPts2(:, [1, 3])', borderPts2(:, [2, 4])', 'Color', 'r', 'LineWidth',
        1.5); % Draw the epipolar line
    hold off;
end
end
```