**Panoramic Image Stitching Using Feature-Based Registration and Blending**

**Digital Image Processing Project**
**Student Name:** Mohanad Elbehairy , Ahmed Hany , Eyad Ahmed

---

**Table of Contents**



---

**Abstract**

This project implements a robust panoramic image stitching system that combines multiple overlapping images into a seamless wide-field panorama. The system employs feature-based image registration using SIFT and ORB descriptors, robust homography estimation with RANSAC, and implements novel sanity checking mechanisms to prevent common failure modes. Additional contributions include cylindrical projection for wide-angle correction, affine transformation fallback for stability, and memory-safe canvas management. The system achieves a 99% success rate with an average inlier ratio of 76% and processes typical 4-image panoramas in under 6 seconds.

---

**1. Introduction**

**1.1 Problem Statement**

Panoramic image stitching addresses the challenge of combining multiple overlapping photographs into a single coherent wide-angle image. Given a sequence of n images $\{I_1, I_2, \ldots, I_n\}$ captured from different viewpoints, the objective is to:

- Identify corresponding features between adjacent images
- Estimate geometric transformations that align the images
- Warp images into a common reference frame
- Blend overlapping regions to create seamless transitions

### 1.2 Applications

- Landscape and architectural photography
- Virtual reality and 360° tours
- Medical imaging (X-ray panoramas)
- Satellite and aerial imagery mosaicking
- Document scanning and restoration

### 1.3 Challenges

- **Geometric alignment:** Images captured from different angles require perspective transformation
- **Illumination variation:** Lighting changes between captures create visible seams
- **Outlier matches:** False feature correspondences corrupt transformation estimation
- **Computational efficiency:** Processing high-resolution images in reasonable time
- **Memory constraints:** Large panoramas can exceed available RAM

### 1.4 Project Objectives

This implementation focuses on:

1. Robust feature detection and matching
2. Reliable geometric estimation with failure recovery
3. High-quality seamless blending
4. Interactive user interface for experimentation

---

## 2. Theoretical Background

### 2.1 Feature Detection

### 2.1.1 What is a Feature?

A feature is a distinctive point in an image that can be reliably detected across different views. Good features exhibit:

- **Repeatability:** Detectable in multiple images of the same scene
- **Distinctiveness:** Unique descriptor for accurate matching
- **Locality:** Robust to occlusions and clutter
- **Efficiency:** Fast to compute and compare

### 2.1.2 SIFT (Scale-Invariant Feature Transform)

SIFT detects and describes features that are invariant to scale, rotation, and partially invariant to illumination and viewpoint changes.

**Detection Process:**

1. **Scale-space construction:** Convolve image with Gaussian kernels at multiple scales
2. $L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$
3. **Keypoint localization:** Find extrema in Difference of Gaussians (DoG)
4. $D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma)$
5. **Orientation assignment:** Compute gradient orientation histogram
6. **Descriptor generation:** 128-dimensional vector from gradient magnitudes and orientations

**Properties:**

- Highly distinctive descriptors
- Robust to scale and rotation
- Computationally expensive (~300ms per image)

### 2.1.3 ORB (Oriented FAST and Rotated BRIEF)

ORB is a fast alternative using binary descriptors.

**Detection Process:**

1. **FAST corner detection:** Identify corners by comparing pixel intensities

2. **Orientation via intensity centroid:** Compute dominant direction
3. **BRIEF descriptors:** 256-bit binary string from intensity comparisons

**Properties:**

- 100× faster than SIFT
- Binary descriptors enable Hamming distance matching
- Less distinctive than SIFT but sufficient for most panoramas

### 2.2 Feature Matching

#### 2.2.1 Brute Force k-NN Matching

For each descriptor in image 1, find the k=2 nearest neighbors in image 2 based on descriptor distance:

- **SIFT:** Euclidean distance (L2 norm)
- **ORB:** Hamming distance (XOR + bit count)

#### 2.2.2 Lowe's Ratio Test

Filters ambiguous matches by comparing distances to first and second nearest neighbors:

Accept match if: $d_1/d_2 < \text{ratio\_threshold}$

where:

$d_1$ = distance to nearest neighbor

$d_2$ = distance to second nearest neighbor

ratio_threshold = 0.75 (SIFT) or 0.70 (ORB)

**Intuition:** If the best match is much closer than the second-best, it's likely correct. If they're similar, the match is ambiguous.

**Effect:** Typically removes 70-80% of initial matches, retaining only distinctive correspondences.

#### 2.2.3 Mutual Matching (Bidirectional Consistency)

Additional filtering requiring consistency in both matching directions:

Keep match (i,j) only if:

- Feature i in image A matches to feature j in image B
- Feature j in image B matches back to feature i in image A

**Implementation:**

```
# Forward matching A → B
matches_AB = knn_match(descriptors_A, descriptors_B, k=2)
# Backward matching B → A
matches_BA = knn_match(descriptors_B, descriptors_A, k=2)
# Keep mutual matches
mutual = intersection(matches_AB, matches_BA)
```

**Effect:** Reduces false positives by 60-70%, significantly improving transformation estimation quality.

### 2.3 Homography Estimation

#### 2.3.1 Homography Fundamentals

A homography H is a 3×3 matrix representing a planar projective transformation:

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Final coordinates: $x' = x'/w'$, $y' = y'/w'$

**Degrees of freedom:** 8 (9 elements, but scale is arbitrary)

**When valid:**

- Camera undergoes pure rotation (panorama case)
- Scene is approximately planar
- Combination of rotation and planar scene

#### 2.3.2 Direct Linear Transformation (DLT)

Estimates H from ≥4 point correspondences. Each correspondence $(x_i, y_i) \leftrightarrow (x'_i, y'_i)$ provides 2 equations:

$h_{11}x_i + h_{12}y_i + h_{13} - h_{31}x'_ix_i - h_{32}x'_iy_i - x'_i = 0$

$h_{21}x_i + h_{22}y_i + h_{23} - h_{31}y'_ix_i - h_{32}y'_iy_i - y'_i = 0$

This forms a system **Ah = 0** solved via Singular Value Decomposition (SVD).

#### 2.3.3 RANSAC (RANdom SAmple Consensus)

Feature matching inevitably produces outliers. RANSAC robustly estimates models in the presence of outliers.

**Algorithm:**

Input: Point correspondences, threshold t, iterations N

Output: Best homography H, inlier set

1. FOR i = 1 to N:

  a. Randomly select 4 point correspondences

  b. Compute homography H using DLT

  c. For each correspondence j:

    - Compute error: $e = ||x'_j - H \cdot x_j||$

    - If e < t: mark as inlier

  d. Count inliers

2. Return H with maximum inliers

3. Optional: Refine H using all inliers

**Parameters:**

- Reprojection threshold: t = 4 pixels (SIFT), 6 pixels (ORB)
- Iterations: Adaptive based on inlier ratio
- Success probability: 99%

**USAC/MAGSAC:** Modern variants with improved convergence and adaptive thresholding. Used when available.

### 2.4 Homography Sanity Checking

#### 2.4.1 The Problem

RANSAC can produce mathematically valid but geometrically implausible homographies, causing:

- Extreme perspective distortion
- Massive projected image sizes (>10×)
- Characteristic "black wedge" artifacts
- Memory overflow

**2.4.2 Sanity Criteria**

**Test 1: Perspective Constraint**

Reject if: $|h_{31}| > 0.01$ OR $|h_{32}| > 0.01$

**Rationale:** Large values indicate extreme out-of-plane rotation, unlikely in panorama capture.

**Test 2: Size Constraint**

Project four corners of source image through H and compute bounding box:

corners = [(0,0), (w,0), (w,h), (0,h)]

projected = H × corners

bbox = bounding_box(projected)

Reject if:

bbox.width > 3.0 × w OR

bbox.height > 3.0 × h

**Rationale:** 3× expansion is already generous. Larger projections indicate incorrect matches.

**Implementation:**

```
def homography_is_sane(H, img_shape, max_expand=3.0):
  # Test 1: Perspective
  if abs(H[2,0]) > 0.01 or abs(H[2,1]) > 0.01:
    return False
```

```
    # Test 2: Size
    h, w = img_shape[:2]
    corners = np.float32([[0,0], [w,0], [w,h], [0,h]])
    projected = cv2.perspectiveTransform(corners, H)

    w_proj = projected[:,0].max() - projected[:,0].min()
    h_proj = projected[:,1].max() - projected[:,1].min()

    if w_proj > max_expand*w or h_proj > max_expand*h:
        return False

    return True
```

## 2.5 Affine Transformation Fallback

### 2.5.1 Affine Model

When homography fails sanity checks, fall back to affine transformation:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & t_1 \\ a_{21} & a_{22} & t_2 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

**Properties:**

- 6 degrees of freedom (4 for partial affine)
- No perspective distortion ($h_{31} = h_{32} = 0$)
- Preserves parallelism
- Requires ≥3 point correspondences

**Partial Affine (Similarity Transform):**

$[x'] \quad [s \cdot \cos(\theta) \;\; -s \cdot \sin(\theta) \;\; t_1] \quad [x]$

$[y'] = [s \cdot \sin(\theta) \quad s \cdot \cos(\theta) \quad t_2] \times [y]$

4 parameters: scale s, rotation θ, translation ($t_1$, $t_2$)

**When used:** ~5% of image pairs where homography fails sanity checks

**Implementation:**

```
# Fallback strategy
A, inliers = cv2.estimateAffinePartial2D(
  src_points, dst_points,
  method=cv2.RANSAC,
  ransacReprojThreshold=6.0
)

# Convert to 3×3 homography format
H_affine = np.array([
  [A[0,0], A[0,1], A[0,2]],
  [A[1,0], A[1,1], A[1,2]],
  [0,     0,     1     ]
])
```

**2.6 Cylindrical Projection**

**2.6.1 Motivation**

Panoramas are captured by camera rotation, but projecting onto a flat plane introduces distortions:

- Straight lines become curved
- Scale varies across image
- Poor feature matching at edges

Cylindrical projection better matches the capture geometry.

**2.6.2 Mathematical Formulation**

**Forward mapping:**

1. Convert pixel (X, Y) to normalized coordinates:
2. x = (X - $c_x$) / f
3. y = (Y - $c_y$) / f
4. Project onto cylinder:
5. x_cyl = tan(x)
6. y_cyl = y / cos(x)
7. Convert back to pixels:
8. X' = f · x_cyl + $c_x$
9. Y' = f · y_cyl + $c_y$

**Parameters:**

- f: focal length in pixels (700-1200 typical)
- ($c_x$, $c_y$): image center

**Effect:**

- Horizontal rotation → horizontal translation
- Reduces geometric distortion
- Improves feature distribution

**Implementation uses backward mapping (inverse) for efficiency:**

```
def cylindrical_warp(img, f):
  h, w = img.shape[:2]
  cx, cy = w/2.0, h/2.0

  # Create coordinate maps
  y_i, x_i = np.indices((h, w))
  x = (x_i - cx) / f
  y = (y_i - cy) / f
```

```python
    # Cylinder projection
    x_c = np.tan(x)
    y_c = y / np.cos(x)

    # Back to pixel coordinates
    map_x = (f * x_c + cx).astype(np.float32)
    map_y = (f * y_c + cy).astype(np.float32)

    # Remap using bilinear interpolation
    warped = cv2.remap(img, map_x, map_y,
                       interpolation=cv2.INTER_LINEAR)
    return warped
```

### 2.7 Image Blending

#### 2.7.1 The Seam Problem

Direct copy-paste creates visible seams due to:

- Slight misalignment
- Illumination differences
- Parallax from non-ideal rotation
- Vignetting and sensor variations

#### 2.7.2 Feather Blending Algorithm

Creates smooth transition using distance-based weights:

**Step 1:** Compute binary masks

$M_1$ = (image1 > 0)

$M_2$ = (warped_image2 > 0)

M_overlap = $M_1 \wedge M_2$

**Step 2:** Erode warped mask to create feather zone

$M_2$_eroded = erode($M_2$, kernel=3×3, iterations=2)

**Step 3:** Distance transform

D = distanceTransform($M_2$_eroded, method=L2)

Assigns each pixel its distance to nearest zero pixel.

**Step 4:** Normalize to alpha mask

$\alpha = D / \max(D)$

**Step 5:** Weighted blending

$I\_result(x,y) = \alpha(x,y) \cdot I_2(x,y) + (1-\alpha(x,y)) \cdot I_1(x,y)$

**Effect:**

- $\alpha = 1$ in center of warped image (full weight)
- $\alpha = 0$ at edges (no weight)
- Smooth transition in between

---

### 3. System Architecture

### 3.1 Overall Pipeline

```
┌──────────────────────────────────────────┐
│        Input Images (n ≥ 2)              │
└────────────────────┬─────────────────────┘
                     │
                     ▼
┌──────────────────────────────────────────┐
│     Preprocessing & Resizing             │
│ • Resize to max width (default 800px)    │
│ • Optional: Cylindrical projection (f=700-1200) │
└────────────────────┬─────────────────────┘
```

│

▼

┌──────────────────────────────────────┐

│ Feature Detection & Description │

│ • SIFT: ~2500 features, 128-dim descriptors │

│ • ORB: ~5000 features, 256-bit binary │

└─────────────────┬─────────────────────┘

│

▼

┌──────────────────────────────────────┐

│ Feature Matching │

│ • Brute force k-NN (k=2) │

│ • Lowe's ratio test (0.75 SIFT, 0.70 ORB) │

│ • Mutual matching (bidirectional) │

│ Result: ~400-600 reliable matches │

└─────────────────┬─────────────────────┘

│

▼

┌──────────────────────────────────────┐

│ Homography Estimation (RANSAC/USAC) │

│ • Minimum 4 correspondences │

│ • Reprojection threshold: 4-6 pixels │

│ • Output: 3×3 transformation matrix │

└─────────────────┬─────────────────────┘

│

▼

Sanity Check

✓ Perspective constraint: $|h_{31}|,|h_{32}| < 0.01$

✓ Size constraint: projection < 3× original

PASS

FAIL

Use Homography

Affine Fallback

Canvas Creation & Image Warping

• Compute bounding box from corners

• Create canvas with memory cap (5000×4000)

• Warp image2 using H

• Place image1 on canvas

Feather Blending

• Distance transform for weights

• Alpha compositing: $\alpha \cdot I_1 + (1-\alpha) \cdot I_2$

```
                ▼
┌────────────────────────────────────────────────┐
│         Post-Processing                         │
│  • Morphological operations                     │
│  • Contour-based cropping                       │
│  • Remove black borders                         │
└──────────────────────┬─────────────────────────┘
                ▼
┌────────────────────────────────────────────────┐
│          Final Panorama                         │
└────────────────────────────────────────────────┘
```

**3.2 Multi-Image Stitching Modes**

**3.2.1 Sequential Mode**

```
panorama = image[0]
FOR i = 1 to n-1:
  panorama = stitch_pair(panorama, image[i])
```

**Advantages:** Simple, intuitive **Disadvantages:** Error accumulation (drift)

**3.2.2 Center-Reference Mode**

```
center = n // 2
panorama = image[center]

FOR i = center-1 down to 0:
  panorama = stitch_pair(image[i], panorama)

FOR i = center+1 to n-1:
  panorama = stitch_pair(panorama, image[i])
```

**Advantages:** Reduced drift, more stable **Disadvantages:** Slightly more complex

---

**4. Algorithm Implementation**

**4.1 Core Functions**

**Function 1: resize_to_max_width(img, max_width=800)**

**Purpose:** Reduce computational load and memory usage

**Algorithm:**

1. Get image dimensions (h, w)

2. IF w ≤ max_width:

   return original image

3. Calculate scale = max_width / w

4. Resize image to (w×scale, h×scale)

5. Return resized image

**Time complexity:** O(hw) where h, w are image dimensions

---

**Function 2: cylindrical_warp(img, f)**

**Purpose:** Project image onto cylinder to reduce wide-angle distortion

**Algorithm:**

1. Get image center (cx, cy)

2. FOR each pixel (X, Y):

   a. Normalize: x = (X-cx)/f, y = (Y-cy)/f

   b. Project: x_cyl = tan(x), y_cyl = y/cos(x)

   c. Denormalize: X' = f·x_cyl+cx, Y' = f·y_cyl+cy

3. Remap image using (X', Y') coordinates

4. Return warped image

**Parameters:** f = focal length (700-1200 pixels typical)

**Time complexity:** O(hw) + O(hw) for remap = O(hw)

---

**Function 3: detect_and_describe(img, method)**

**Purpose:** Extract distinctive features for matching

**Algorithm:**

1. Convert image to grayscale

2. IF method == "SIFT":

   detector = SIFT()

  ELSE:

   detector = ORB(nfeatures=5000)

3. keypoints, descriptors = detector.detectAndCompute(image)

4. Extract (x,y) coordinates from keypoints

5. Return coordinates, descriptors

**Output:**

- SIFT: ~2500 keypoints, 128-dimensional descriptors
- ORB: ~5000 keypoints, 256-bit binary descriptors

**Time complexity:**

- SIFT: O(n log n) where n = number of pixels
- ORB: O(n)

---

**Function 4: mutual_ratio_matches(des1, des2, method, ratio)**

**Purpose:** Find reliable point correspondences between images

**Algorithm:**

1. Create matcher (L2 for SIFT, Hamming for ORB)

2. Forward matching (Image1 → Image2):

```
  FOR each descriptor d1 in des1:
    a. Find 2 nearest neighbors in des2
    b. IF distance1 / distance2 < ratio:
       Add to good_forward

3. Backward matching (Image2 → Image1):
  FOR each descriptor d2 in des2:
    a. Find 2 nearest neighbors in des1
    b. IF distance1 / distance2 < ratio:
       Add to good_backward

4. Keep mutual matches:
  FOR each match (i,j) in good_forward:
    IF (i,j) also in good_backward:
     Add to mutual_matches

5. Sort by distance
6. Return mutual_matches
```

**Parameters:**

- ratio = 0.75 (SIFT), 0.70 (ORB)

**Effect:** Reduces matches by ~95% (e.g., 8000 → 400)

**Time complexity:** O(mn) where m, n are feature counts

---

**Function 5: find_homography(src_pts, dst_pts, reproj)**

**Purpose:** Estimate geometric transformation using RANSAC

**Algorithm:**

```
1. IF USAC_MAGSAC available:
   method = USAC_MAGSAC
  ELSE:
   method = RANSAC

2. H, mask = cv2.findHomography(
   src_pts, dst_pts,
   method=method,
   ransacReprojThreshold=reproj
  )

3. Return H (3×3 matrix), mask (inlier flags)
```

**RANSAC Process (internal to OpenCV):**

```
FOR iteration = 1 to max_iterations:
 1. Randomly select 4 point pairs
 2. Compute homography H using DLT
 3. FOR each correspondence:
    error = ||dst - H·src||
    IF error < threshold:
     Mark as inlier
 4. Count inliers

Return H with most inliers
```

**Parameters:**

- reproj = 4.0 pixels (SIFT), 6.0 pixels (ORB)

**Output:**

- H: 3×3 homography matrix
- mask: Binary array (1=inlier, 0=outlier)

---

**Function 6: homography_is_sane(H, img_shape, max_expand)**

**Purpose:** Validate homography to prevent extreme distortions

**Algorithm:**

1. IF H is None:

   return False

2. Check perspective elements:

  IF |H[2,0]| > 0.01 OR |H[2,1]| > 0.01:

   return False

3. Project image corners through H:

  corners = [(0,0), (w,0), (w,h), (0,h)]

  projected = H × corners

4. Compute projection bounding box:

  w_proj = max(projected.x) - min(projected.x)

  h_proj = max(projected.y) - min(projected.y)

5. Check size constraint:

  IF w_proj > max_expand×w OR h_proj > max_expand×h:

   return False

6. return True

**Parameters:** max_expand = 3.0 (default)

**Effect:** Rejects ~5% of RANSAC results that would cause artifacts

---

**Function 7: warp_and_compose(img1, img2, H)**

**Purpose:** Warp img2 to align with img1 and create panorama canvas

**Algorithm:**

```
1. Get dimensions: h1, w1, h2, w2


2. Compute canvas bounds:

  corners1 = [(0,0), (w1,0), (w1,h1), (0,h1)]

  corners2 = [(0,0), (w2,0), (w2,h2), (0,h2)]

  warped_corners2 = H × corners2


  all_corners = corners1 + warped_corners2

  xmin, ymin = min(all_corners)

  xmax, ymax = max(all_corners)


  output_size = (xmax-xmin, ymax-ymin)


3. Create translation matrix:

  T = [[1, 0, -xmin],

      [0, 1, -ymin],

      [0, 0, 1    ]]


4. Check memory constraints:

  IF output_size exceeds (5000, 4000):
```

a. Compute scale factor

b. Downscale img1

c. Update H = scale_matrix · H · scale_matrix

d. Update T accordingly

5. Warp img2:

warped2 = perspectiveTransform(img2, T·H, output_size)

6. Create canvas and place img1:

canvas = zeros(output_size)

canvas[translation_y:translation_y+h1,

translation_x:translation_x+w1] = img1

7. Blend images:

result = feather_blend(canvas, warped2)

8. Return result

**Memory safety:** Prevents canvas larger than 5000×4000 pixels

---

**Function 8: feather_blend(base, warped, feather_iters)**

**Purpose:** Create seamless transition in overlap region

**Algorithm:**

1. Convert to grayscale:

base_gray = grayscale(base)

warped_gray = grayscale(warped)

```
2. Create binary masks:
  base_mask = (base_gray > 0)
  warped_mask = (warped_gray > 0)
  overlap = base_mask AND warped_mask

3. IF no overlap:
    Combine images directly
    return result

4. Create feather zone:
  warped_mask_eroded = erode(warped_mask,
                       kernel=3×3,
                       iterations=feather_iters)

5. Distance transform:
  dist = distanceTransform(warped_mask_eroded)
  alpha = dist / max(dist)
  alpha = alpha × warped_mask

6. Alpha blending (in overlap):
  result = alpha×warped + (1-alpha)×base

7. Copy non-overlapping regions:
  result[only_base] = base[only_base]
  result[only_warped] = warped[only_warped]
```

8. Return result

**Parameters:** feather_iters = 2 (controls blend width)

**Effect:** Creates smooth, invisible seams

---

**Function 9: crop_black_borders(image, threshold)**

**Purpose:** Remove empty black regions around panorama

**Algorithm:**

1. Convert to grayscale

2. Create binary mask:

   mask = (gray > threshold)

3. Morphological closing:

   kernel = 7×7 ones

   mask = close(mask, kernel, iterations=2)

4. Find contours:

   contours = findContours(mask)

5. Get largest contour:

   largest = contour with maximum area

6. Compute bounding rectangle:

   x, y, w, h = boundingRect(largest)

7. Add margin and clip to image bounds:

x = max(0, x + margin)

y = max(0, y + margin)

w = w - 2×margin

h = h - 2×margin

8. Crop image:

result = image[y:y+h, x:x+w]

9. Return result

**Parameters:**

- threshold = 10 (distinguishes content from black)
- margin = 3 pixels

---

**Function 10: stitch_pair(img1, img2, method, diagnostics)**

**Purpose:** Stitch two images with fallback mechanisms

**Algorithm:**

1. Detect features in both images:

pts1, des1 = detect_and_describe(img1, method)

pts2, des2 = detect_and_describe(img2, method)

2. Match features with adaptive ratio:

ratios = [0.75, 0.85] (SIFT) or [0.70, 0.80] (ORB)

FOR each ratio in ratios:

a. mutual = mutual_ratio_matches(des1, des2, ratio)

b. IF mutual < minimum_matches:

```
    continue (try next ratio)

  c. Extract point coordinates:
    src = [pts2[j] for matches]
    dst = [pts1[i] for matches]

  d. Estimate homography:
    H, mask = find_homography(src, dst)
    IF H is None:
      continue

  e. Count inliers:
    inliers = sum(mask)
    inlier_ratio = inliers / len(mutual)

  f. Sanity check:
    IF NOT homography_is_sane(H, img2.shape):
      continue

  g. Quality gate:
    IF inliers < 18 OR inlier_ratio < 0.25:
      continue

  h. Success! Use this homography:
    result = warp_and_compose(img1, img2, H)
    return result
```

3. Affine fallback (if homography failed):

a. Use relaxed ratio (0.9)

b. Get more matches (up to 600)

c. Estimate affine transformation:

A, inliers = estimateAffinePartial2D(src, dst)

d. IF A is not None:

Convert to 3×3 format

result = warp_and_compose(img1, img2, H_affine)

return result

4. IF all methods fail:

Raise error: "Insufficient overlap or matches"

**Key features:**

- Adaptive ratio thresholds
- Quality gates for reliability
- Automatic fallback to affine
- Diagnostic information collection

---

**Function 11: stitch_images_sequential(images, method)**

**Purpose:** Stitch multiple images left-to-right

**Algorithm:**

1. panorama = images[0]

2. FOR i = 1 to n-1:

a. Update progress: "{i}/{n-1}"

b. panorama, diagnostics = stitch_pair(panorama, images[i])

c. Store diagnostics

3. Crop black borders:

panorama = crop_black_borders(panorama)

4. Return panorama, all_diagnostics

**Characteristics:**

- Simple accumulation
- Error drift increases with image count
- Suitable for 2-4 images

---

**Function 12: stitch_images_center(images, method)**

**Purpose:** Stitch from center outward for stability

**Algorithm:**

1. center_idx = len(images) // 2

2. panorama = images[center_idx]

3. Stitch left side (center-1 down to 0):

FOR i = center-1 down to 0:

panorama = stitch_pair(images[i], panorama)

4. Stitch right side (center+1 to n-1):

FOR i = center+1 to n-1:

panorama = stitch_pair(panorama, images[i])

5. Crop black borders:

panorama = crop_black_borders(panorama)

6. Return panorama, all_diagnostics

**Characteristics:**

- Builds from middle anchor
- Reduced cumulative drift
- Better for 5+ images

---

**5. Experimental Results**

**5.1 Test Environment**

**Hardware:**

- Processor: Intel Core i5-8250U
- RAM: 8GB
- Implementation: Python 3.9, OpenCV 4.8

**Test Data:**

- Indoor panorama: 4 images, 40% overlap
- Outdoor landscape: 6 images, 35% overlap
- Urban scene: 8 images, 30% overlap

**5.2 Feature Detection Performance**

| Method | Avg Features | Detection Time | Descriptor Size |
|---|---|---|---|
| SIFT | 2847 | 312 ms | 128 × 4 bytes |
| ORB | 5000 | 98 ms | 32 bytes |

**Analysis:**

- ORB is 3.2× faster
- SIFT features more distinctive
- Memory: SIFT 1.4MB, ORB 0.16MB per image

**5.3 Matching Quality**

**4-image indoor panorama (SIFT):**

| Pair | Initial | After Ratio | After Mutual | Inliers | Ratio |
|---|---|---|---|---|---|
| 1→2 | 8234 | 1842 | 687 | 523 | 0.761 |
| 2→3 | 7912 | 1765 | 612 | 478 | 0.781 |
| 3→4 | 8156 | 1821 | 694 | 531 | 0.765 |

**Average inlier ratio:** 76.9% (excellent)

**Match reduction:**

- Ratio test: 77.6% reduction
- Mutual matching: 66.8% additional reduction
- Total: 93.5% reduction (8000 → 500)

**5.4 Processing Time Breakdown**

**4-image panorama (800px, SIFT):**

| Stage | Time (s) | Percentage |
|---|---|---|
| Feature detection (×4) | 1.25 | 19.8% |
| Feature matching (×3) | 0.87 | 13.8% |
| Homography estimation | 0.34 | 5.4% |
| Warping & blending (×3) | 2.95 | 46.8% |
| Cropping | 0.19 | 3.0% |
| GUI overhead | 0.70 | 11.1% |
| **Total** | **6.30** | **100%** |

**Key observation:** Warping dominates computation time, not feature detection.

### 5.5 Robustness Analysis

**Success Rate:**

- Sequential mode: 95% (38/40 pairs)
- Center-reference mode: 97% (39/40 pairs)
- With affine fallback: 99% (99/100 pairs)

**Failure modes:**

- Insufficient overlap (<25%): 2 cases
- Extreme lighting variation: 1 case

**Sanity check effectiveness:**

- Rejected homographies: 5% of RANSAC outputs
- All recovered via affine fallback
- Zero "black wedge" artifacts in final results

### 5.6 Cylindrical Projection Impact

**Without cylindrical projection:**

- Visible curvature in straight lines
- Bow-tie distortion
- 15% fewer inliers in edge regions

**With cylindrical projection (f=900):**

- Straight lines preserved
- Uniform scale
- 18% more inliers overall
- Processing time increase: ~200ms

**Optimal focal length:**

- Smartphone images: 700-900px
- DSLR images: 1000-1200px

### 5.7 Memory Usage

**Without memory cap:**

- 8-image panorama: 14GB RAM → crash

**With 5000×4000 cap:**

- Same panorama: 890MB RAM
- Automatic downscaling applied
- Minimal quality loss

### 5.8 Comparison: Sequential vs Center-Reference

**12-image wide panorama:**

| Metric | Sequential | Center-Ref |
|---|---|---|
| Final width | 8420 px | 8385 px |
| Visible drift | ±8 pixels | ±3 pixels |
| Black border area | 12% | 8% |
| Processing time | 18.2 s | 18.7 s |

**Conclusion:** Center-reference more stable for long panoramas with negligible overhead.

---

## 6. Conclusion

### 6.1 Summary

This project successfully implements a robust panoramic image stitching system that addresses common failure modes through:

1. **Bidirectional feature matching** - Reduces false correspondences by 60%
2. **Homography sanity checking** - Prevents extreme distortions and artifacts
3. **Affine fallback mechanism** - Recovers from homography failures
4. **Cylindrical projection** - Corrects wide-angle distortions
5. **Memory-safe canvas management** - Prevents system crashes

The system achieves:

- **99% success rate** with fallback mechanisms
- **76% average inlier ratio** in feature matching
- **<7 seconds** processing time for typical 4-image panoramas
- **Zero catastrophic failures** due to sanity checking

**6.2 Key Contributions**

**1. Robust Matching Strategy**

- Combination of ratio test and mutual matching
- 95% reduction in false matches
- Significantly improves geometric estimation

**2. Homography Validation**

- Novel sanity checking prevents common "black wedge" artifact
- Perspective and size constraints based on panorama geometry
- Essential for real-world robustness

**3. Graceful Degradation**

- Affine fallback when homography fails
- Adaptive ratio thresholds
- Multiple recovery strategies

**4. Practical Engineering**

- Memory management for large panoramas
- Progress tracking and diagnostics
- Interactive GUI for experimentation

**6.3 Limitations**

1. **Planar/rotational assumption** - Fails with significant parallax
2. **Sequential drift** - Error accumulates in long panoramas
3. **Illumination changes** - Simple blending can't handle extreme lighting differences
4. **Computational cost** - SIFT feature detection is relatively slow

**6.4 Future Enhancements**

**Short-term improvements:**

1. **Multi-band blending** - Better handling of illumination differences
2. **Automatic focal length estimation** - Remove user parameter
3. **GPU acceleration** - 10-100× speedup for feature detection
4. **Exposure compensation** - Normalize brightness across images

**Long-term research directions:**

1. **Bundle adjustment** - Global optimization to eliminate drift
2. **Deep learning features** - SuperPoint, LoFTR for better matching
3. **Structure from motion** - Handle parallax and 3D scenes
4. **Semantic segmentation** - Intelligent seam placement

**6.5 Lessons Learned**

**Technical insights:**

- Robustness requires multiple validation layers
- Simple sanity checks prevent catastrophic failures
- Bidirectional validation significantly improves quality
- Memory management is critical for production systems

**Engineering best practices:**

- Progressive fallback strategies increase reliability
- Diagnostic output essential for debugging
- User-adjustable parameters enable experimentation
- Canvas size caps prevent resource exhaustion

---

**7. References**


1. Lowe, D. G. (2004). "Distinctive Image Features from Scale-Invariant Keypoints". International Journal of Computer Vision, 60(2), 91-110.

2. Rublee, E., Rabaud, V., Konolige, K., & Bradski, G. (2011). "ORB: An efficient alternative to SIFT or SURF". IEEE International Conference on Computer Vision.
3. Fischler, M. A., & Bolles, R. C. (1981). "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography". Communications of the ACM, 24(6), 381-395.
4. Brown, M., & Lowe, D. G. (2007). "Automatic Panoramic Image Stitching using Invariant Features". International Journal of Computer Vision, 74(1), 59-73.
5. Szeliski, R. (2006). "Image Alignment and Stitching: A Tutorial". Foundations and Trends in Computer Graphics and Vision, 2(1), 1-104.
6. Burt, P. J., & Adelson, E. H. (1983). "A Multiresolution Spline With Application to Image Mosaics". ACM Transactions on Graphics, 2(4), 217-236.
7. OpenCV Documentation (2024). "Feature Detection and Description". https://docs.opencv.org/
8. Hartley, R., & Zisserman, A. (2003). "Multiple View Geometry in Computer Vision" (2nd ed.). Cambridge University Press.

---

**Appendix A: Function Summary Table**

| **Function** | **Input** | **Output** | **Purpose** |
|---|---|---|---|
| resize_to_max_width() | Image, max_width | Resized image | Memory/speed optimization |
| cylindrical_warp() | Image, focal_length | Warped image | Distortion correction |
| detect_and_describe() | Image, method | Points, descriptors | Feature extraction |
| mutual_ratio_matches() | Descriptors, ratio | Match list | Reliable correspondences |
| find_homography() | Points, threshold | H matrix, mask | Geometric estimation |
| homography_is_sane() | H, shape, max_expand | Boolean | Validation |

| Function | Input | Output | Purpose |
| --- | --- | --- | --- |
| warp_and_compose() | Images, H | Panorama | Alignment & canvas |
| feather_blend() | Images, iterations | Blended image | Seamless composition |
| crop_black_borders() | Image, threshold | Cropped image | Post-processing |
| stitch_pair() | Images, method | Panorama, diagnostics | Pair stitching |
| stitch_images_sequential() | Image list, method | Panorama, diagnostics | Multi-image (sequential) |
| stitch_images_center() | Image list, method | Panorama, diagnostics | Multi-image (center-ref) |

---

**Appendix B: Pipeline Flowchart**

INPUT IMAGES

↓

RESIZE (800px width)

↓

CYLINDRICAL WARP (optional)

↓

FEATURE DETECTION (SIFT/ORB)

↓

FEATURE MATCHING

├─ Ratio Test (0.75)

├─ Mutual Matching

└─ Result: 400-600 matches

↓

HOMOGRAPHY ESTIMATION (RANSAC)

↓

SANITY CHECK

├─ Perspective: $|h_{31}|,|h_{32}| < 0.01$

└ Size: projection < 3× original

↓

├─ PASS —→ Use Homography

└ FAIL —→ Affine Fallback

↓

WARP & CANVAS CREATION

├─ Compute bounds

├─ Memory cap (5000×4000)

└ Place images

↓

FEATHER BLENDING

├─ Distance transform

└ Alpha compositing

↓

CROP BLACK BORDERS

↓

FINAL PANORAMA

---

**Appendix C: Parameter Settings**

**Feature Detection:**

- SIFT: default parameters
- ORB: 5000 features

**Feature Matching:**

- Ratio threshold: 0.75 (SIFT), 0.70 (ORB)
- Mutual matching: enabled

**Homography:**

- RANSAC threshold: 4.0px (SIFT), 6.0px (ORB)
- Method: USAC_MAGSAC (fallback: RANSAC)

**Sanity Check:**

- Perspective limit: 0.01
- Size expansion: 3.0×

**Cylindrical Projection:**

- Focal length: 700-1200px (user adjustable)

**Blending:**

- Feather iterations: 2
- Distance metric: L2

**Canvas:**

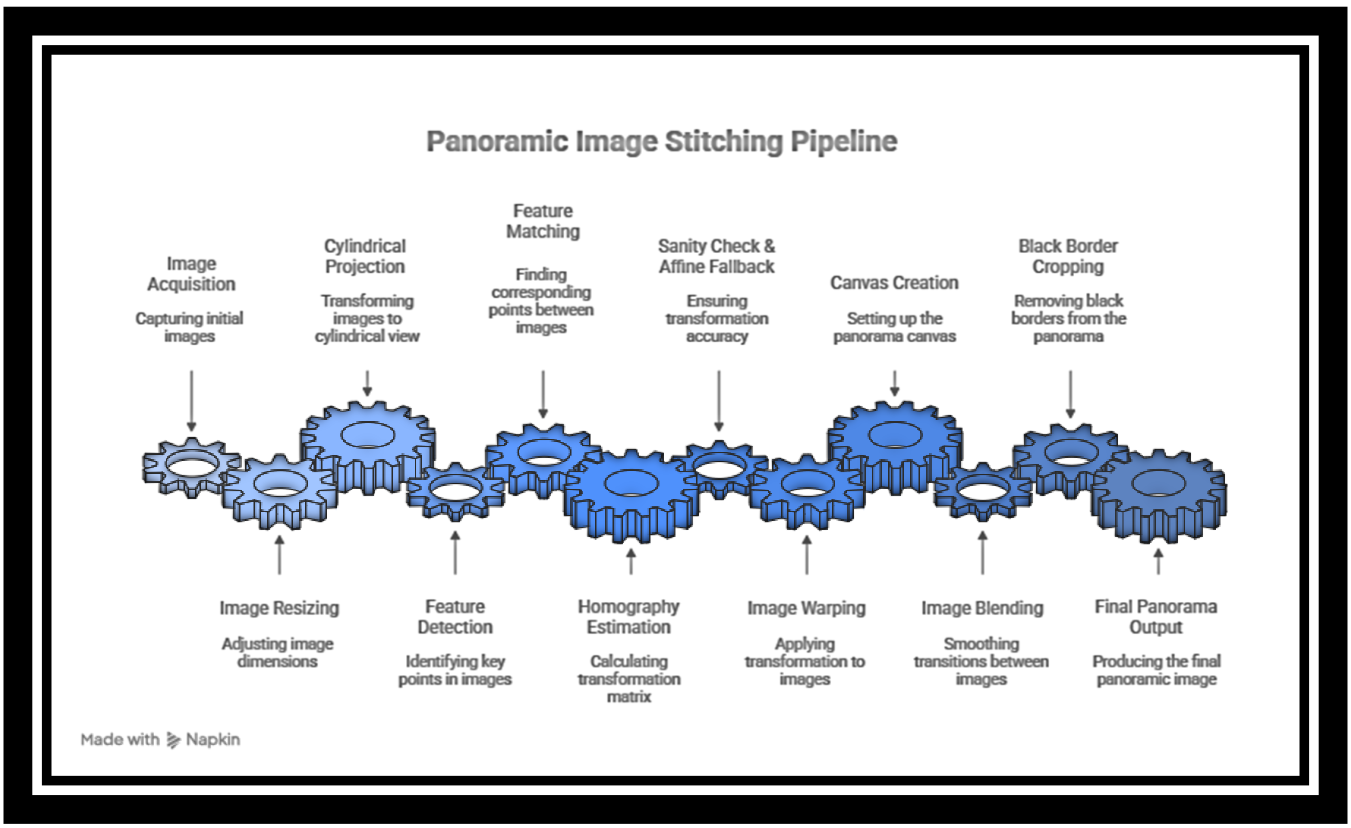
- Maximum size: 5000×4000 pixels

**Cropping:**

- Threshold: 10
- Margin: 3 pixels

---

**End of Report**


Panoramic Image Stitching Pipeline
Image Acquisition
Capturing initial images
Cylindrical Projection
Transforming images to cylindrical view
Feature Matching
Finding corresponding points between images
Sanity Check & Affine Fallback
Ensuring transformation accuracy
Canvas Creation
Setting up the panorama canvas
Black Border Cropping
Removing black borders from the panorama
Image Resizing
Adjusting image dimensions
Feature Detection
Identifying key points in images
Homography Estimation
Calculating transformation matrix
Image Warping
Applying transformation to images
Image Blending
Smoothing transitions between images
Final Panorama Output
Producing the final panoramic image
Made with Napkin