Computer Vision Project 2

1 Image Transformation and Stitching

1.1 Introduction

Image stitching is a computer vision technique used to combine multiple photographic images with overlapping fields of view to produce a segmented panorama or high-resolution image. The process implemented in this code follows a feature-based approach, which involves detecting and matching distinctive features across images to determine how they should be aligned and merged.

1.2 Manual Identification

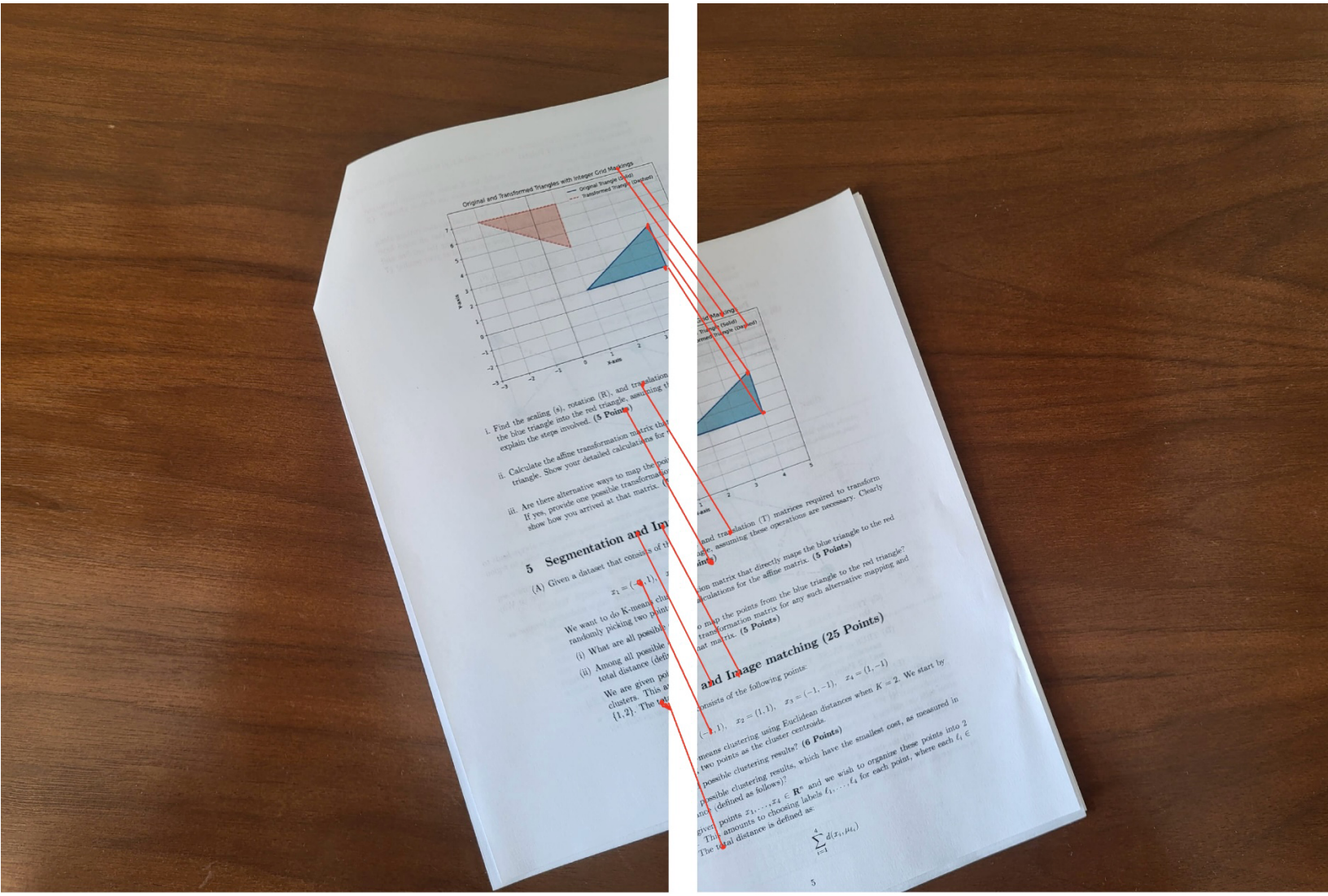


Image 1

Image 2

For manual identification, I selected distinct and recognizable features that were common in both images. This careful process is crucial for accurately aligning the images, ensuring the correct transformation and perspective adjustment for successful stitching. Despite being time-intensive, manual point selection allows for precise feature matching, especially when automated detection methods could face challenges due to noise or overlapping textures.

1.3 Automated Detection

Feature Detection: The first step in the stitching process is to identify distinctive points in each image that can be reliably matched across different views. This is accomplished using the ORB (Oriented FAST and Rotated BRIEF) algorithm.

ORB combines the FAST key-point detector and the BRIEF descriptor with modifications to enhance performance. It detects corner-like features in the image and computes binary descriptors for each key-point. These descriptors are essentially bit strings that encode the appearance of the image patch surrounding each key-point. The advantages of ORB include:

- Fast computation
- Rotation invariance
- Resistance to image noise

Feature Matching: After detecting features, next we find correspondences between features in the two images. A Brute-Force matcher (cv2.BFMatcher) is used with the Hamming distance as the matching metric, which is suitable for binary descriptors like those produced by ORB. The knnMatch() method finds the two best matches for each descriptor in the first image. Lowe's ratio test is applied to filter out poor matches. A match is considered good if the distance to the closest neighbor is less than 75% of the distance to the second-closest neighbor.

Homography Functions: After obtaining a set of matched features, the next step is to compute a transformation that aligns one image with another. This transformation is represented by a homography matrix, which describes how points in one image plane map to points in another image plane.

RANSAC (Random Sample Consensus) algorithm is used in this scenario to robustly estimate the homography.

- For a specified number of iterations, the function randomly selects four-point correspondences and computes a candidate homography using these points**.**
- It then evaluates how many of the total correspondences are consistent with this candidate homography by counting inliers based on a defined distance threshold. If the current candidate has more inliers than previously recorded, it updates the best homography.
- After completing all trials, the function returns the most reliable homography matrix, which is essential for accurately aligning images in tasks such as image stitching.

This approach is robust to outliers, which are inevitable in the feature matching process due to errors or ambiguities.

The **compute_homography_from_points()** function calculates a homography matrix using Direct Linear Transformation (DLT) based on four-point correspondences from two images. It constructs a matrix that encapsulates the linear equations derived from the relationships between source and destination points. By applying Singular Value Decomposition (SVD) to this matrix, it extracts the solution corresponding to the smallest singular value, which represents the optimal homography matrix. The resulting matrix is then normalized to ensure that its bottom-right element equals one, facilitating proper application in subsequent transformations. This function is crucial for deriving the transformation needed to align images accurately.

The **count_inliers ()** function assesses how many point correspondences are consistent with a given homography matrix by counting inliers based on their projected positions. For each source point, it transforms the point using the provided homography and normalizes the result to convert it back from homogeneous coordinates. The Euclidean distance between this transformed point and its corresponding destination point is calculated, and if this distance falls below a specified threshold, it is counted as an inlier. The function iterates through all point correspondences and returns the total number of inliers found. This process is vital for evaluating the quality of candidate homographies during RANSAC, ensuring that only reliable transformations are considered for image alignment.

Image Warping: Once the homographies between image pairs are computed, they are used to warp the images into a common plane. This is done using perspective warping, which transforms the source image to align with the target image's perspective.

cv2.warpPerspective() is used to apply the computed homography and warp the second image. Next, a composite canvas is created large enough to hold both images. The warped second image is placed on this canvas, and then the first image is overlaid in its original position.

Lastly, to clean up the result, the image is thresholded to identify non-black regions. The bounding rectangle of the largest contour is used to crop the result, removing any black borders.

1.4 Observations

1. Robust Feature Detection: ORB provides an efficient means of detecting features that are invariant to rotation and scale. It is a good balance between speed and feature quality, detecting many features (5000) to ensure sufficient matches.
2. Effective Matching Accuracy: The use of the Brute-Force matcher with Lowe's ratio test effectively filters out many incorrect matches, improving the overall alignment quality.
3. Robust Homography Estimation: The custom RANSAC implementation allows for robust handling of outliers, which is critical when matching features across different views.
4. Warping can introduce distortions, however, applying homographies correctly allows for effective alignment and could provide a good result.
5. Blending Limitations: This implementation uses a simple overlay method for blending, which can sometimes result in visible seams. More advanced blending techniques could improve the final result.

1.5 Comparative Analysis

Accuracy: Manual feature selection can be highly accurate since points are chosen carefully by the user, making it effective for difficult images with noise or repetitive patterns. Automated methods like ORB or SIFT generally work well for clear images but may struggle with ambiguity or distortions, leading to mismatches.

Smoothness of Blending: With manual selection, blending is smooth when points are chosen well, but small errors can cause visible seams. Automated feature descriptors detect more key points, often resulting in better alignment and smoother transitions across image borders.

Ease of Implementation: Manual selection is straightforward but time-consuming and impractical for large datasets. Automated methods are efficient and scalable, requiring minimal human input, but involve additional complexity in handling matching and filtering of key points.

2 Hough Transform

2.1 Introduction

The Hough Transform is a powerful feature extraction technique used in image analysis, computer vision, and digital image processing. It is particularly effective in detecting geometric shapes such as lines, circles, and ellipses in images, even when the shapes are partially occluded or distorted. The transform works by converting the problem of shape detection in image space to a peak-finding problem in a parameter space. Here we will be using Hough Transform to do 2 things:

- Detect lines on the runway to help the space shuttle's pilot to land the shuttle safely.
- Detect circles on the landing pad to assist the Falcon 9 booster's landing process.

2.2 Runway

The goal of this part is to detect lines within an image of a runway using the Hough Transform from scratch. This process involves several stages, from pre-processing to final visualization.

Image Processing: The color image is converted to grayscale, simplifying subsequent processing by reducing the image to a single intensity channel. Then a Gaussian blur is applied to the grayscale image. This step reduces noise and smooths the image, which is crucial for more accurate edge detection. The blur helps to eliminate small variations that could be mistaken for edges.

Edge Detection: The edge detection process is a critical step in identifying potential line segments.

- In the **image_gradients ()** , Sobel filters are used to compute gradients in the x (horizontal) and y (vertical) directions. This operation determines the rate of change of pixel intensities in both horizontal and vertical directions, highlighting areas of rapid intensity changes which are likely to be edges. The gradient magnitude represents the strength of edges, while the gradient angle indicates their orientation. The gradient magnitude is scaled to fit within an 8-bit range (0–255) for easier visualization and thresholding.

- **Non-Maximum Suppression** thins out the edges by suppressing all gradient values except local maxima. For each pixel, the gradient direction is used to compare the pixel's magnitude with the magnitudes of neighboring pixels along that direction. If the current pixel is not the local maximum, it is suppressed to 0. It ensures that edges are represented by single-pixel wide lines, improving the accuracy of subsequent line detection.
- **Hysteresis Thresholding** is a dual-threshold method employed to identify strong and weak edges. Two thresholds, a high and a low, are used. Pixels with magnitudes above the high threshold are marked as strong edges, while those between the low and high thresholds are marked as weak edges. Strong edges are immediately accepted, while weak edges are only kept if they connect to strong edges. This approach reduces noise and ensures more continuous edge detection.

Hough Transform Application: The core of the line detection process lies in the Hough Transform:

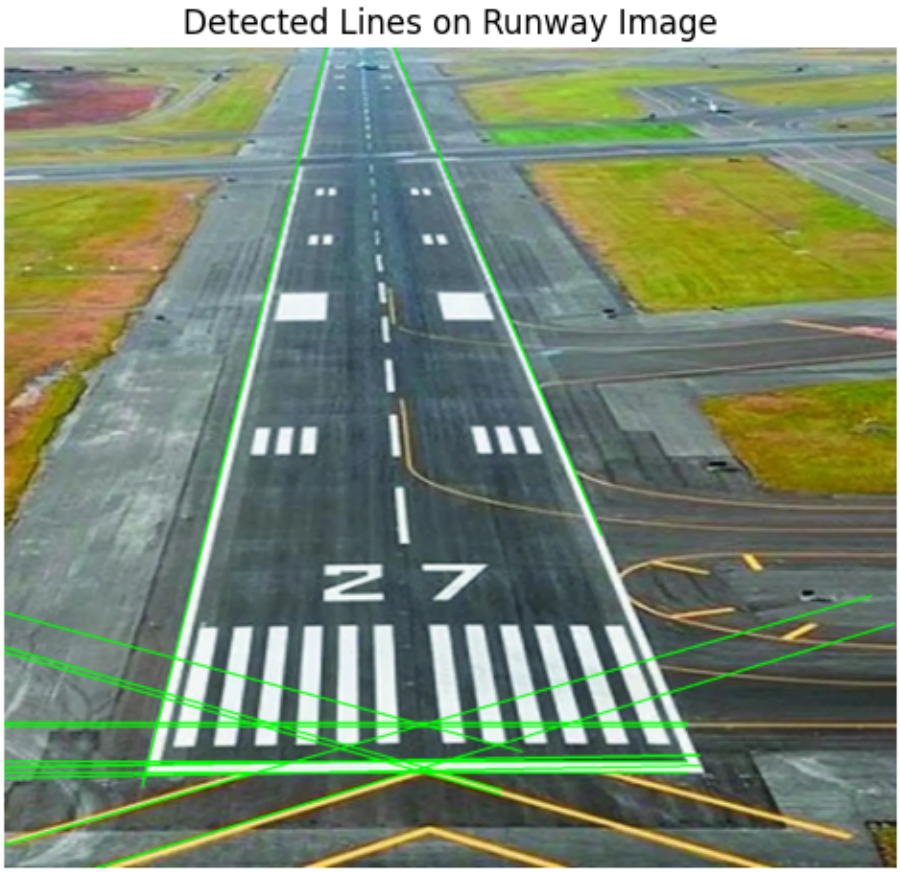
- **Hough Space Creation**: A two-dimensional accumulator array is created to represent the Hough space. This space maps straight lines in the image to points in the Hough space.
- **Voting Process:** Each edge pixel votes for potential lines in the Hough space. This is done by transforming the pixel's coordinates into a sinusoidal curve in the Hough space.
- **Accumulation**: The votes are accumulated in the Hough space. Areas with high accumulation (peaks) correspond to likely lines in the original image.

Peak Detection and Line Identification: The final stages involve identifying and visualizing the detected lines:

- **Peak Detection**: The algorithm identifies peaks in the Hough space by applying a threshold to the accumulator array. These peaks represent the most prominent lines in the image.
- **Line Parameter Extraction**: For each detected peak, the corresponding line parameters (rho and theta) are extracted from the Hough space.
- **Line Drawing**: Using the extracted parameters, lines are drawn on the original image. This provides a visual representation of the detected lines overlaid on the input image.

Visualization: The edge-detected image is displayed, showing the results of the Canny edge detection algorithm. After that, the Hough space is visualized as a 2D image, where brighter spots indicate a higher likelihood of line presence and the detected peaks in the Hough space are highlighted. Finally, the original image with the detected lines overlaid is printed.

2.2.1 Results and Observations:



- The algorithm successfully identified key runway features. The detected lines align well with the actual runway markings, especially the white boundary lines.
- A few extraneous lines appear near the bottom of the image, which do not correspond to actual runway features. These might be due to noise or imperfections in edge detection and thresholding.

Overall, the result demonstrates effective line detection for structured environments like runways, with some minor areas for improvement in reducing noise.

2.3 Landing Pad

In this second part, we implement a circle detection algorithm using the Hough transform technique on a satellite image of a SpaceX facility. The algorithm processes the image through various stages of image processing and computer vision techniques to identify and highlight circular structures within a specific region of interest (ROI).

Image Processing: The color image is first converted to grayscale, simplifying subsequent processing by reducing the image to a single intensity channel.

- A **Region of Interest** (ROI) is then selected from the grayscale image, focusing the analysis on a specific area where circular structures are expected.
- **Histogram equalization** is then applied to this ROI using histogram_equalization () to enhance contrast, making features more distinguishable.
- Following this, a **Gaussian blur** is applied using the generate_gaussian_kernel () and gaussian_blur () functions. The former function creates a 2D Gaussian kernel for image blurring, while the latter one applies Gaussian Blur to the input image using the generated kernel. These steps reduce noise and smooths the image, which is crucial for more accurate edge detection.
- A **median blu**r is then applied using the median_blur () function, further reducing noise while preserving edges.

Edge Detection: The edge detection process is a critical step in identifying potential circular segments.

- In the **compute_gradient_magnitude ()** function, Sobel filters are used to compute gradients in the x and y directions. This operation determines the rate of change of pixel intensities, highlighting areas of rapid intensity changes which are likely to be edges.
- **Hysteresis Thresholding** is implemented in the hysteresis_threshold () function. This dual-threshold method is employed to identify strong and weak edges. Two thresholds, a high and a low, are used. Pixels with magnitudes above the high threshold are marked as strong edges, while those between the low and high thresholds are marked as weak edges. Strong edges are immediately accepted, while weak edges are only kept if they connect to strong edges. This approach reduces noise and ensures more continuous edge detection.

Hough Transform Application: The core of the circle detection process lies in the Hough Transform implementation:

- **Hough Space Creation**: A three-dimensional accumulator array is created in the find_strongest_circle () function to represent the Hough space. This space maps circles in the image to points in the Hough space.
- **Voting Process**: Each edge pixel votes for potential circles in the Hough space. This is done by considering a range of possible radii and angles for each edge pixel.
- **Accumulation**: The votes are accumulated in the Hough space. Areas with high accumulation (peaks) correspond to likely circles in the original image.

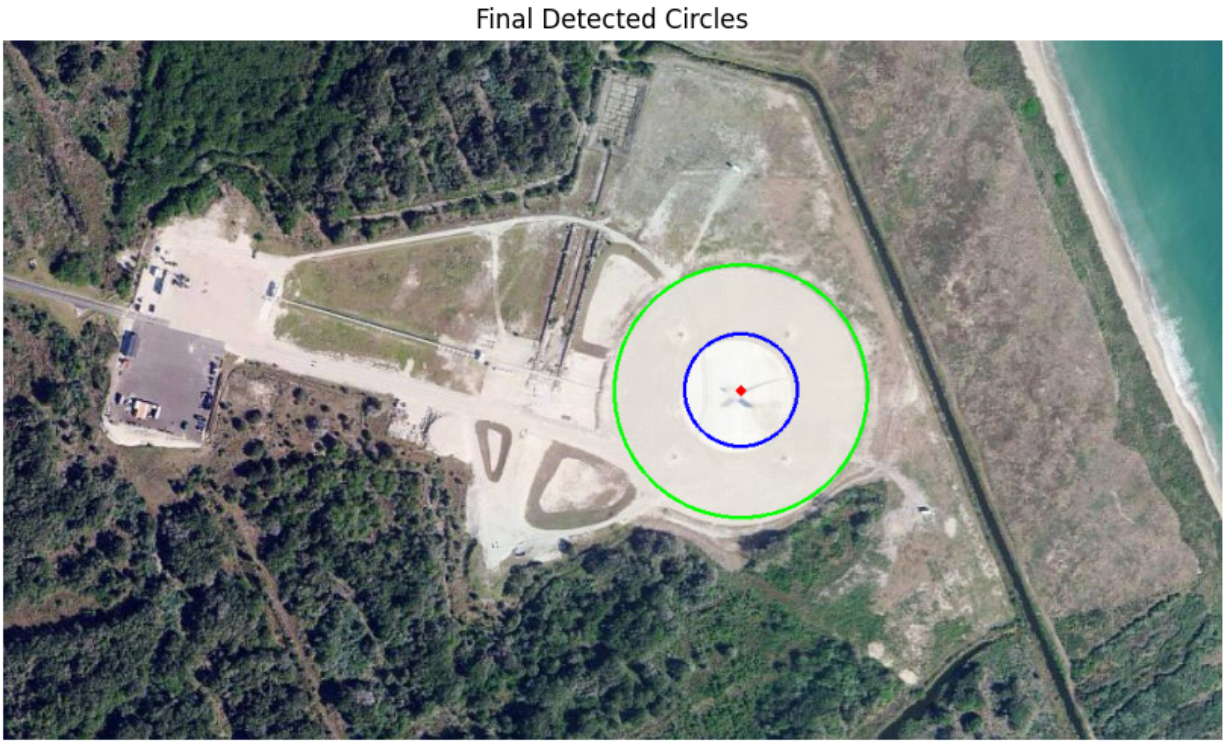
Peak Detection and Circle Identification: The final stages involve identifying and visualizing the detected circles:

- **Peak Detection**: The algorithm identifies the strongest peak in the Hough space by finding the maximum value in the accumulator array.
- **Circle Parameter Extraction**: For the detected peak, the corresponding circle parameters (center coordinates and radius) are extracted from the Hough space.
- **Circle Drawing**: Using the extracted parameters, circles are drawn on the original image using the draw_circles () function. This provides a visual representation of the detected circles overlaid on the input image.

Visualization: The Hough spaces for both large and small circles are visualized using the show_hough_space () function, where brighter spots indicate a higher likelihood of circle presence and the detected peaks in the Hough space are highlighted. Finally, the original image with the detected circles overlaid is displayed, showing the results of the circle detection algorithm.

This implementation demonstrates a comprehensive approach to circle detection, combining various image processing techniques with a custom Hough transform implementation.

2.3.1 Results and Observations:



- The algorithm successfully identified two concentric circles in the SpaceX facility:
    - A larger outer circle marked in green, likely representing the main landing pad perimeter
    - A smaller inner circle marked in blue, possibly indicating the central landing target zone
    - A red center point, the center of the circles, is marking the near center of landing pad.
- The result of the algorithm may not be perfect, but it has a pretty good accuracy.

2.4 Improvements

Runway Detection

The algorithm successfully identifies the key runway features, particularly the white boundary lines. The detected lines align well with the actual runway structure, especially the long parallel lines along the runway edges.

Improvements Suggested: Some unnecessary lines appear at the bottom of the image, particularly near the taxiways and grass areas. These lines do not correspond to actual runway features and are likely caused by noise or imperfections in edge detection. So, the following could help get a better result:

- Applying more advanced noise filtering techniques during edge detection could help remove irrelevant lines.
- Implementing a post-processing step to merge fragmented line segments, ensuring continuous detection of long straight lines could also help.
- Integrating deep learning-based line detection like HoughNet for detecting complex and noisy scenarios could also improve the result.

Landing Pad Detection

The algorithm accurately detected two concentric circles on the landing pad, with the outer circle marked in green and the inner circle in blue with a red dot marking the center.

Improvements Suggested: The detected circles are slightly offset from what appears to be their true boundaries. This could be due to faint or blurred edges on the landing pad. The following could help yield a better result:

- Enhancing edge detection by increasing contrast or applying adaptive thresholding to better capture faint boundaries.
- Implementing a multi-scale detection to ensure that circles of varying sizes are accurately detected.
- Replacing the Sobel operator with a more robust edge-detection method could also help.
- Post-processing the detected circles to validate their shape using geometric consistency checks or machine learning classifiers could help.

While both algorithms demonstrate satisfactory performance in detecting key features, there are opportunities for improvement, particularly in noise reduction, edge clarity, and feature validation. Implementing these improvements would increase reliability and make these systems more suitable for real-world applications.

3 Segmentation

3.1 Introduction

Image segmentation is a fundamental task in computer vision aimed at partitioning an image into meaningful regions that correspond to different objects or structures. In this question we cover two different image segmentation techniques. The first approach utilizes Mean Shift filtering combined with Canny edge detection, while the second approach is Normalized Graph Cut Segmentation which employs Superpixel generation with SLIC followed by Spectral Clustering. Both techniques aim to segment the image effectively, but they use distinct methods to achieve this.

3.2 Mean Shift Segmentation

Mean Shift is a non-parametric clustering algorithm that iteratively shifts data points towards regions of higher density. In image processing, it is commonly used for segmentation by filtering pixels based on spatial and color similarities. This method helps enhance object boundaries and reduce noise, making it effective for initial image smoothing. This technique smooths the image by merging similar colors, effectively reducing noise and creating homogeneous regions based on color similarity.

The Mean Shift algorithm analyzes the spatial and color information of pixels within a given neighborhood to shift each pixel towards the average color of nearby pixels. We use the cv2.pyrMeanShiftFiltering function from OpenCV, which applies a joint spatial-range filtering operation:

- **Spatial Radius** (sp): Defines the neighborhood size considered for smoothing. A larger spatial radius results in more significant smoothing.
- **Color Radius** (sr): Controls the range of colors treated as similar. A higher color radius merges colors that are further apart in value, leading to stronger segmentation.

The filtering process enhances the image by reducing variability within each region, making it easier to detect distinct boundaries and features in subsequent steps.

Edge Detection Using Canny Algorithm: After applying Mean Shift filtering, edge detection is performed using the Canny edge detector. The Canny method is a multi-step process designed to identify areas of rapid intensity change, which typically correspond to object boundaries in the image.

It involves the following steps:

- Grayscale Conversion: The filtered image is first converted to grayscale, as edge detection relies on intensity changes rather than color differences.
- Gradient Calculation: The algorithm calculates the gradient of pixel intensities to find regions with strong intensity changes, which are potential edges.
- Non-Maximum Suppression: This step refines the edges by thinning them, ensuring that only the most prominent edges remain.
- Double Thresholding: Two thresholds are used to classify edge pixels into strong and weak edges. Only strong edges and connected weak edges are preserved, reducing noise.

The output of the Canny edge detector highlights the edges of objects in the image, making them more pronounced and easier to identify.

Edge Overlay: The detected edges are overlaid onto the filtered image, highlighting the boundaries with a distinct green color ([0,255,0]). This provides a clear visual representation of the segmented areas.

3.2.1 Results and Observations

- Noise Reduction: The Mean Shift filtering successfully reduced noise and enhanced region homogeneity.
- Boundary Detection: Canny edge detection accurately identified the edges of the stop sign.
- Visualization: The overlay of edges on the filtered image provided a clear depiction of object boundaries.

The combination of Mean Shift filtering and Canny edge detection proved effective for image segmentation and boundary enhancement. This method is useful for applications where object boundaries need to be highlighted clearly, such as in object recognition tasks.

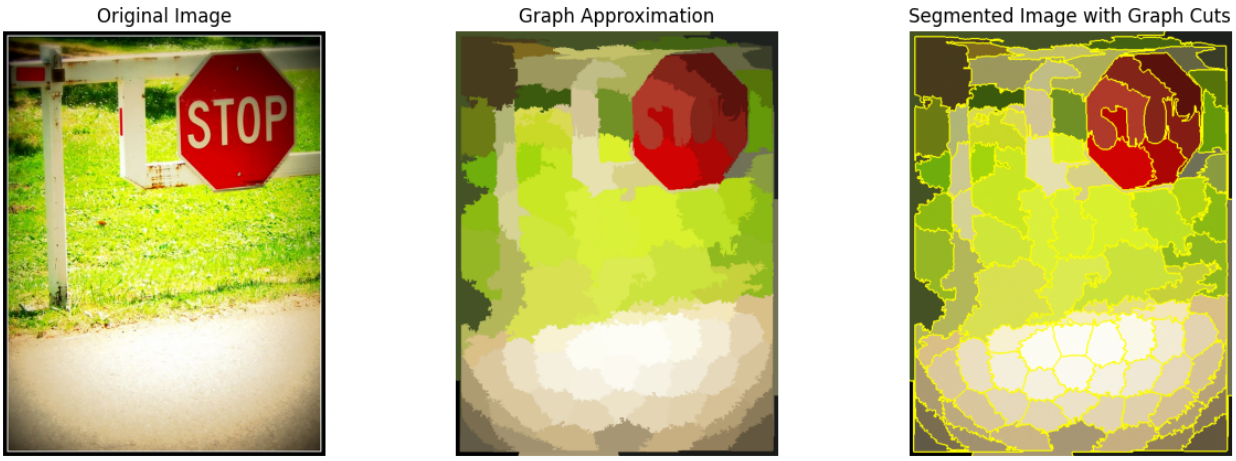
3.3 Normalized Graph Cut Segmentation

The second method utilizes a Normalized Graph Cut Segmentation approach, approximated via Spectral Clustering. This method is well-suited for complex structures, by leveraging a graph-based representation of the image. We utilize superpixels to reduce computational complexity and apply spectral clustering to group these superpixels based on color similarity.
The Normalized Graph Cut method formulates the image as a graph, where nodes represent regions (e.g., superpixels), and edges represent the similarity between regions. The objective is to partition the graph by minimizing the cut, which separates dissimilar nodes while maintaining similarity within segments. This approach allows for better delineation of complex objects.

- Superpixel Generation (SLIC): The image is segmented into superpixels using the SLIC algorithm with n_segments = 150 and compactness = 10. Superpixels reduce the complexity of the image by grouping pixels into perceptually meaningful regions.
    - n_segments is the number of superpixels which is set at 150 for a balance between granularity and efficiency.
    - compactness controls the trade-off between color similarity and spatial proximity. A value of 10 ensures that the superpixels adhere well to object boundaries.
- Mean Color Computation: For each superpixel, we compute the mean color by averaging the RGB values of the pixels within that superpixel. This step reduces the dimensionality of the problem by representing each superpixel with its average color, making the clustering process more efficient.

- Affinity Matrix Construction: An affinity matrix is created based on color similarity between superpixels, it represents the similarity between superpixels based on their mean color. The Euclidean distance between mean colors is calculated using the cdist function from scipy.spatial.distance. The Euclidean distance metric is used to compute similarities, and the matrix is transformed using a Gaussian kernel.
- Spectral Clustering: We use the Spectral Clustering method, which approximates the Normalized Graph Cut. It treats the image as a graph where nodes represent superpixels, and edges represent the affinities between them. The algorithm partitions the graph into n clusters (set to 5) based on minimizing the graph cut, which leads to well-defined segments. This method is effective because it considers both local and global similarities, allowing for better delineation of complex structures like the edges of a stop sign.
- Segmented Image Creation: A new image is constructed based on the clustering labels, where each segment is represented by the average color of the corresponding superpixel.
- Boundary Overlay: The segmented image is overlaid with boundaries for visualization using the mark_boundaries function from skimage.segmentation. This helps visualize the segmentation results and highlights the effectiveness of the graph cut approach.

3.3.1 Results and Observations



- Region Segmentation: Normalized Graph Cut effectively grouped similar color regions, isolating the stop sign from the background. More improvement is needed to get a perfect result with a perfect clustering.
- Boundary Clarity: The method delineated sharp boundaries, preserving the shape of the stop sign despite background complexity.

The graph cut segmentation method has isolated the stop sign with clear boundaries, although further refinement might be needed for smoother transitions between regions in complex backgrounds.

3.4 Discussion

Mean Shift Segmentation

Strengths:

- **Handles gradients and textures well**: Ideal for images with complex backgrounds or subtle color transitions. For stop signs, if the background contains some gradient or texture (such as pavement or a road), the method can still effectively smooth out these variations while preserving the sharp boundaries of the stop sign.
- **Less sensitive to color variation**: Mean shift segmentation is less sensitive to specific color choices compared to graph cut segmentation. Works well even if the stop sign's color is not uniform, like fading edges or slight lighting variations. This is advantageous in real-world images where such inconsistencies are common.
- **Effective edge detection**: Canny edge detection combined with mean shift highlights clear object boundaries, which is beneficial for identifying the edges of the stop sign.

Weaknesses:

- **Blurring fine details**: The smoothing effect can soften the stop sign's edges. This blurring can lead to a loss of sharpness in the stop sign's outline, particularly if the stop sign has a lot of textural details or fine features that are critical for its identification.
- **Parameter sensitivity**: The performance of this segmentation heavily depends on tuning the spatial and color radius. If the radii are set too large, the algorithm may smooth out too much of the image, merging distinct regions together. If the radii are too small, the algorithm might fail to merge neighboring regions that should be grouped together, resulting in over-segmentation.
- **Edge Detection Limitations**: Although edge detection highlights the boundary, it may not fully capture the context of the stop sign.
- **Computationally expensive**: Like graph cut segmentation, mean shift segmentation can also be computationally expensive, especially for large images or when the parameters are not well-tuned. Filtering large images with mean shift can be time-consuming, making it less suitable for real-time applications.

Normalized Graph Cut Segmentation

Strengths:

- **Handling of Color Contrast**: Graph cut segmentation, when combined with superpixel generation (using SLIC), is highly effective in segmenting regions with clear color contrast. Since stop signs are typically red, and the background is usually of a different color, the method performs well when the image has distinct color regions. The algorithm clusters superpixels based on color similarity, which aids in accurately identifying the stop sign from the background.
- **Good Boundary Definition**: Spectral clustering and graph cuts create precise object boundaries, essential for accurately identifying stop signs.
- **Scalability**: The algorithm scales well with a reasonable number of superpixels. This flexibility makes it effective for both simple and moderately complex images, where the number of segments can be adjusted based on the required granularity of segmentation.

Weaknesses:

- **Sensitivity to Color Similarity**: Graph cut segmentation struggles if the background contains similar colors to the stop sign. For example, if there is a red object in the background or the stop sign is surrounded by a similarly colored area, the algorithm might fail to isolate the stop sign effectively.
- **Computationally intensive**: Can be slow, especially with high-resolution images or many superpixels. Constructing the affinity matrix and performing spectral clustering can be time-consuming, especially in real-time applications.
- **Potential Boundary Mistakes**: Inaccurate boundaries can occur when the stop sign has irregular shapes or when there are subtle color transitions along its boundaries.

Comparison for Stop Sign Identification:

- **Mean Shift**: Better for handling textured or gradient backgrounds but may blur edges, making it less precise.
- **Graph Cut**: Best for high contrast stop signs with clear edges; excels at delineating sharp boundaries.

Conclusion:

Graph Cut is more effective for sharp boundaries and clear contrasts, while Mean Shift is better for complex backgrounds but may lose some edge precision.

4 Creative Section

4.1 Introduction

In this section, we aim to implement a classical image segmentation technique, which gives the best IoU's compared to other such classical methods. The best segmentation chosen for this is the Watershed Algorithm for segmenting objects in an image.
This algorithm is particularly effective for separating touching or overlapping objects. It treats image intensity as a topographic surface where light regions represent mountains and dark regions represent valleys, and then it floods the valleys from markers until boundaries (watersheds) are formed.

The goal of this task is to use the Watershed algorithm to segment objects in each image and evaluate the accuracy of the segmentation using the Intersection over Union (IoU) metric with the help of the given ground truth of the image. IoU compares the segmented regions with the ground truth annotations to assess how well the segmentation method performs.

4.2 Watershed Algorithm

Grayscale Conversion: The first step is to convert the input image to grayscale cv2.cvtColor(), which simplifies the segmentation process by reducing the color complexity. Watershed operates on intensity gradients, so a grayscale image is more suitable for the following steps.

Otsu Thresholding: We apply Otsu's thresholding (cv2.threshold) to the grayscale image. Otsu's method automatically determines an optimal threshold value that separates foreground from background, producing a binary mask where foreground and background are distinctly separated.
The thresholding is performed in an inverted manner, ensuring that the background is marked as white, and the foreground is black.

Noise Removal and Enhancement: Using morphological opening (cv2.morphologyEx), small noise and tiny foreground regions are removed by applying an erosion followed by dilation. This step helps clean up small irregularities in the mask.
A dilation operation (cv2.dilate) is then applied to the binary mask to identify the sure background. This step ensures that regions known to be background are expanded enough to ensure that all objects are surrounded by a clear boundary.
Distance Transform: The distance transform (cv2.distanceTransform) computes the distance from each pixel to the nearest background pixel. This gives us a measure of how deep a pixel is into the foreground, effectively identifying regions that are likely to be part of the foreground.
A thresholding step is used to mark the pixels that are sure to be part of the foreground, based on a percentage (70% here) of the maximum distance value.
Unknown Region Identification: The unknown regions (pixels that could potentially be either foreground or background) are calculated by subtracting the sure foreground from the sure background. These unknown regions are crucial for the watershed process, as the algorithm will attempt to flood these areas.
Marker Labeling: Connected Components (cv2.connectedComponents) are used to label all sure foreground regions. A label of 1 is assigned to these regions, and sure background regions are labeled with a 0. The markers are then incremented by 1 to distinguish the background from foreground regions.
Watershed Algorithm Application: The Watershed Algorithm (cv2.watershed) is applied to the image using the labeled markers. This algorithm floods the image from the sure foreground regions, trying to propagate the flooding to the unknown regions. The watershed lines are marked with -1 in the result, which represents the boundaries between different regions.
Segmentation Result: The final segmentation result is generated by setting all pixels where markers are greater than 1 to 255 (foreground). Pixels with a marker of -1 are labeled as boundaries. The segmented result is a binary mask where regions corresponding to different objects are marked, and the boundaries between them are well-defined.
IoU Calculation: The Intersection over Union (IoU) is computed to evaluate the quality of the segmentation. The IoU compares the overlap between the segmented region and the ground truth mask. This metric is computed by finding the intersection and union of the predicted segmentation and the ground truth, which provides a measure of accuracy for each region.
The average IoU across all segmented regions gives a summary of the algorithm's performance.

Justification: The Watershed algorithm is particularly suited for this task because:

- **Effective for Overlapping Objects**: Watershed segmentation is well-known for handling situations where objects overlap or touch, which makes it superior to simpler methods like thresholding or edge detection.
- **Gradient-Based Approach**: By utilizing the intensity gradients, the algorithm makes informed decisions on the boundaries of objects, making it effective in complex scenes where boundaries are not sharply defined.
- **Flexibility with Preprocessing**: It can be easily combined with preprocessing steps (e.g., morphological operations) to improve the accuracy of segmentation.

4.3 Results and Comparison

Segmentation Method	IoU Scores	Average IoU
Watershed Segmentation	[0.0072, 0.0030, 0.0057, 0.1107, 0.0759, 0.0657, 0.1035, 0.0001]	**0.0465**
K- Means Segmentation	[0.0018, 0.0008, 0.0019, 0.0002, 0.0037, 0.0002, 0.0004, 0.0012]	**0.0013**
Canny Edge Detection	[0.0113, 0.0096, 0.0184, 0.0096, 0.0532, 0.0745, 0.0108, 0.0104]	**0.0247**

Based on the Intersection over Union (IoU) scores for three different segmentation methods—Canny Edge Detection, K-means Segmentation, and Watershed Segmentation—we can evaluate and compare their performance empirically. Key Insights:

- **Canny Edge Detection** provides an average IoU of 0.0247, which indicates it performs poorly in generating full segmentations.
- **K-means Segmentation** gives an average IoU of 0.0013, which is much lower and reflects its inadequacy in handling complex images with multiple objects.
- **Watershed Segmentation** stands out with an average IoU of 0.0465, which is significantly better than both Canny and K-means, indicating that it is more capable of accurately segmenting regions in complex images.