Computer Vision Project Report: Coin Detection and Panoramic Image Stitching

Niranjan Gopal
IMT2022543

February 23, 2025

Abstract

This report presents the implementation and analysis of two computer vision projects: (1) a coin detection and counting system utilizing image segmentation techniques, and (2) an automated image stitching pipeline for creating panoramic images. The first part focuses on detecting and counting coins in images using contour detection and area-based segmentation. The second part demonstrates the creation of panoramic images through feature matching, homography computation, RANSAC algorithm and image warping. Both implementations showcase practical applications of computer vision algorithms while addressing real-world challenges in image processing. All the codes have been uploaded to github :-Repository Link

1 Coin Detection and Analysis

1.1 Detection Methodology

The coin detection system employs OpenCV's contour detection algorithm to identify circular objects within the image. The implementation follows these key steps:

1. Convert the input image to grayscale
2. Apply threshold to create a binary image

3. Detect contours using `cv2.findContours`
4. Filter contours based on area to eliminate noise

```
# Find contours
thresh_copy = thresh.copy()
contours, _ = cv2.findContours(
            thresh,
            cv2.RETR_TREE,
            cv2.CHAIN_APPROX_NONE
            )
area = {}
for i in range(len(contours)):
    cnt = contours[i]
    ar = cv2.contourArea(cnt)
    area[i] = ar
srt = sorted(area.items(), key=lambda x: x[1], reverse=True)
results = np.array(srt).astype("int")
num = np.argwhere(results[:, 1] > 500).shape[0]
print("Number␣of␣coins␣", num-1)
```

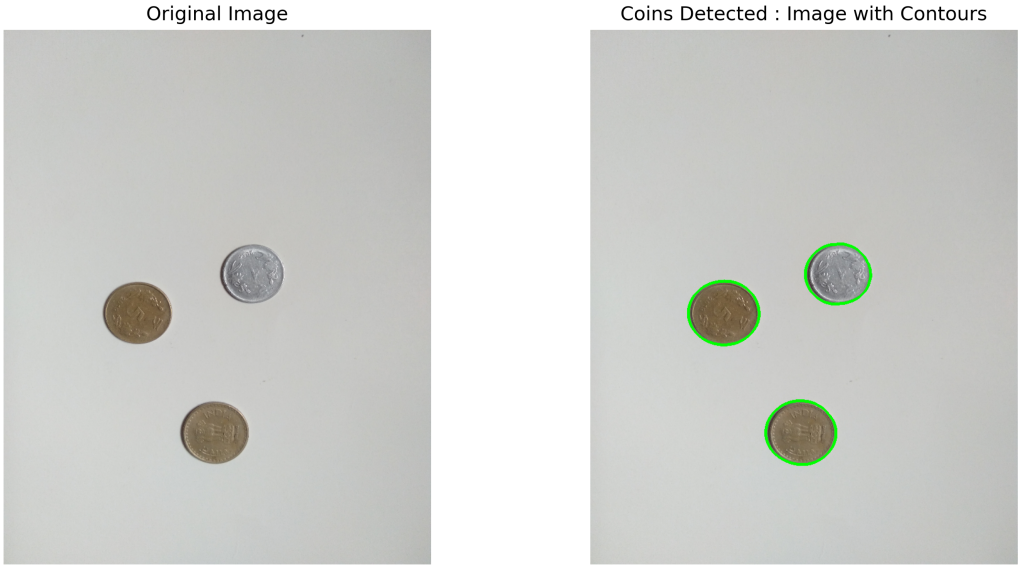

Figure 1: Original and Contour-detected Images

1.2 Segmentation Process

The segmentation phase involves:

1. Computing contour areas
2. Sorting contours by area in descending order
3. Applying area threshold (≥ 500 pixels)
4. Creating unique color masks for each identified coin

```
# Sort contours by area in descending order
srt = sorted(area.items(),
         key=lambda x: x[1],
         reverse=True
     )
results = np.array(srt).astype("int")
num = np.argwhere(results[:, 1] > 500).shape[0]
```

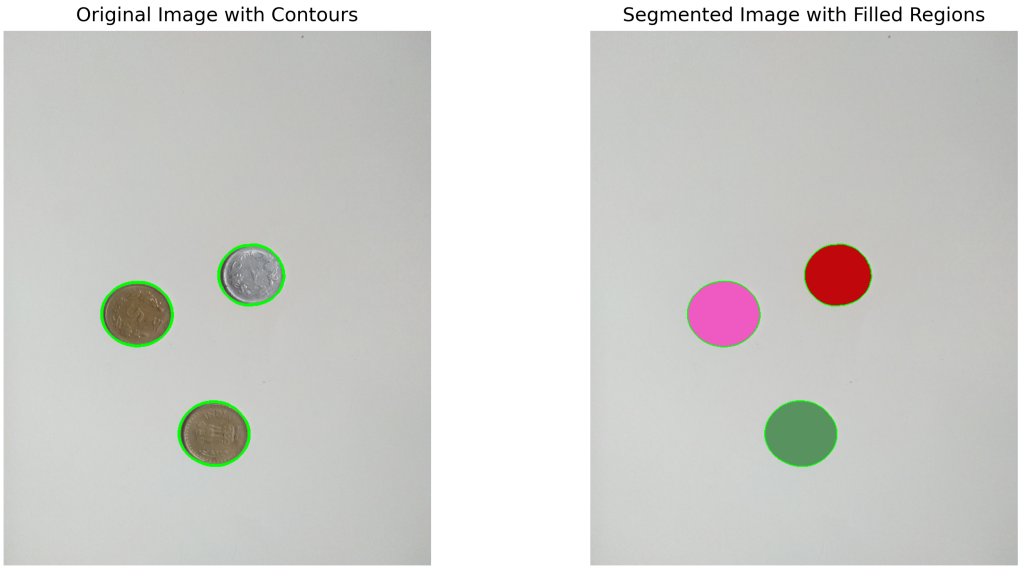

Figure 2: Segmentation results with colored regions

1.3 Coin Counting

The counting mechanism utilizes the filtered contours to determine the number of coins present in the image. The implementation subtracts one from the total number of contours to account for the outer boundary, providing an accurate count of coins in the image.

2 Panoramic Image Stitching

2.1 Feature Detection and Matching

The panorama creation process begins with robust feature detection:

```
def get_keypoint(left_img, right_img):
    l_img = cv2.cvtColor(left_img, cv2.COLOR_BGR2GRAY)
    r_img = cv2.cvtColor(right_img, cv2.COLOR_BGR2GRAY)
    key_points1 = sift.detect(l_img, None)
    key_points1, descriptor1 = surf.compute(l_img, key_points1)
    l = [ key_points1, descriptor1, key_points2, descriptor2  ]
    return l
```

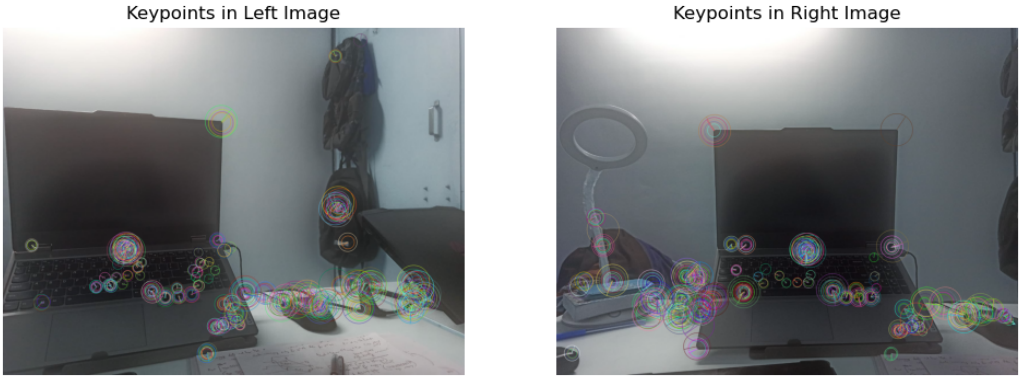

Figure 3: Feature points computed using SIFT detector / ORB detector

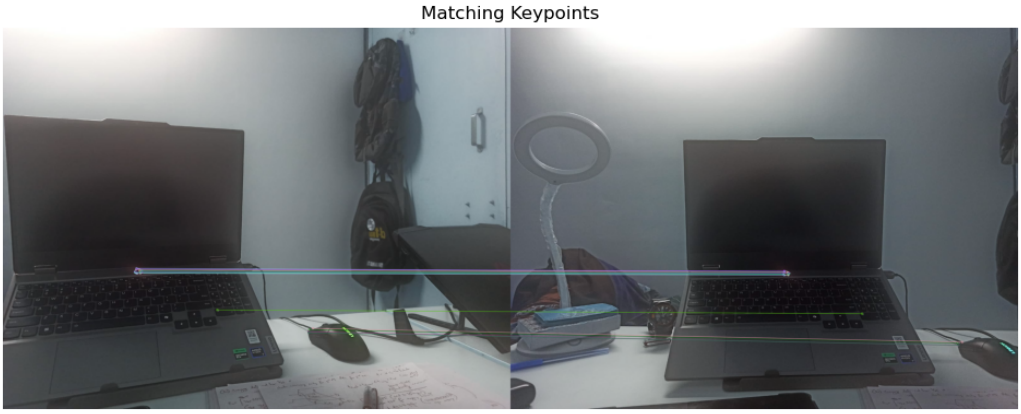

Figure 4: Matching Features

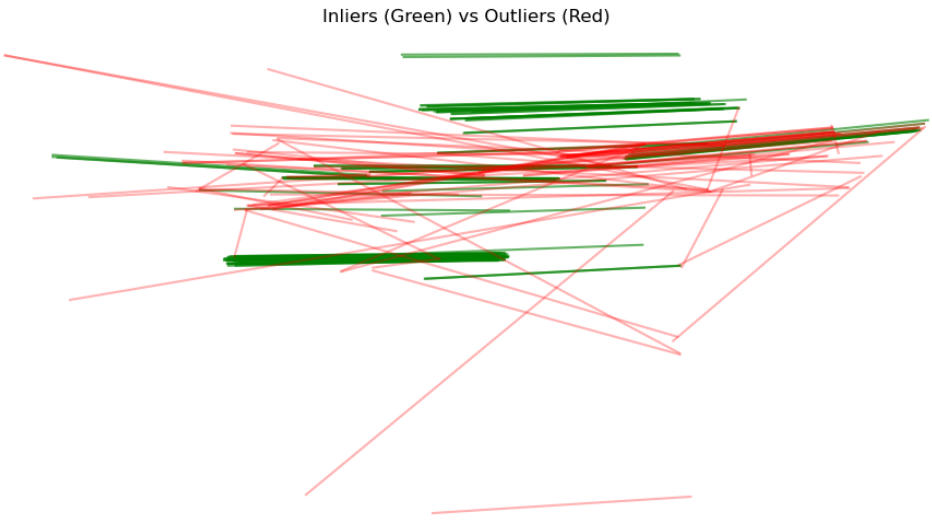

Figure 5: Using RANSAC to filter Inliers and Outliers

2.2 Homography Computation

The homography matrix is computed and then piped to the RANSAC algorithm to ensure a robust perspective transformation.

```
def homography(points):
    A = []
    for pt in points:
        x, y = pt[0], pt[1]
        X, Y = pt[2], pt[3]
        A.append([x, y, 1, 0, 0, 0, -X * x, -X * y, -X])
        A.append([0, 0, 0, x, y, 1, -Y * x, -Y * y, -Y])

    A = np.array(A)
    u, s, vh = np.linalg.svd(A)
    H = vh[-1, :].reshape(3, 3)
    H /= H[2, 2]
    return H

def ransac(good_pts):
    best_inliers = []
    final_H = []
    t = 5
    for i in range(5000):
        random_pts = random.choices(good_pts, k=4)
        H = homography(random_pts)
        # ... RANSAC implementation
    return final_H
```

2.3 Image Warping and Blending

The final stage involves:

1. Applying perspective transformation
2. Computing output image dimensions
3. Warping the source image
4. Blending overlapping regions

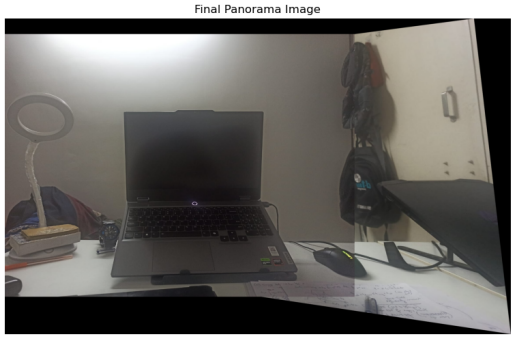

Figure 6: Final Panorama of Photos taken using Phone

Figure 7: Panorma of Photos that were taken under similar Illumination

3 Challenges and Solutions

3.1 Illumination Variations

One significant challenge encountered was dealing with varying illumination conditions:

- **Problem:** Coins with high reflectivity or poor lighting conditions blend with the background
- **Impact:** Contour areas fall below the 500-pixel threshold
- **Solution:** Implemented adaptive thresholding and explored histogram equalization techniques

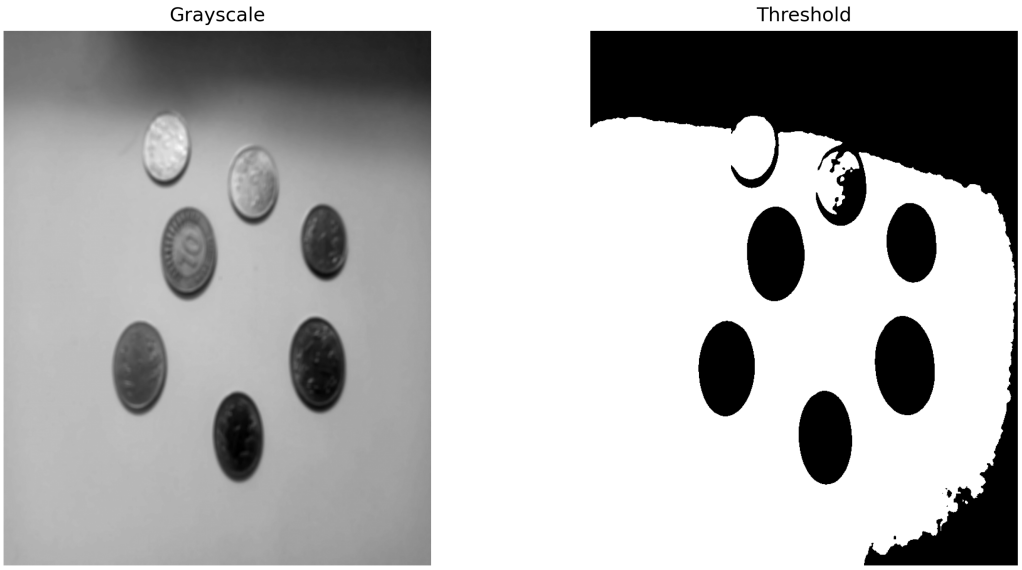

Figure 8: Thresholding of Edge Case Images

Figure 9: Detector Not Working because of unsuitable contours

3.2 SIFT/SURF Patent Restrictions

A notable technical challenge arose from OpenCV's implementation of feature detection algorithms:

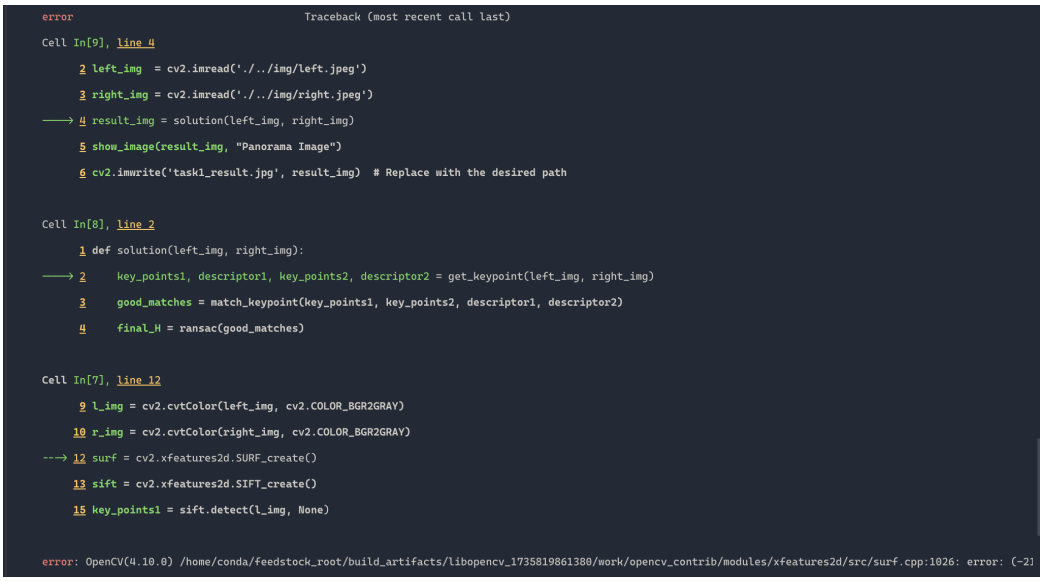

Figure 10: Error that suggested that the algorithm is patented

- **Original Implementation:** Initially used SIFT/SURF for feature detection
- **Error Encountered:** OpenCV raised a patent restriction error as SIFT and SURF are patented algorithms
- **Solution:** Replaced with ORB (Oriented FAST and Rotated BRIEF) algorithm

The switch from SIFT/SURF to ORB represents a transition from patented to open-source computer vision algorithms. While SIFT and SURF are known for their robust feature detection capabilities, ORB provides comparable performance while being free to use. ORB combines the FAST keypoint detector and BRIEF descriptor with modifications to enhance performance, making it an efficient alternative for feature detection and description in our panorama pipeline.

```
# Original code (patent-restricted)
key_points1 = sift.detect(l_img, None)
key_points1, descriptor1 = surf.compute(l_img, key_points1)

# Replacement using ORB (open-source)
key_points1, descriptor1 = orb.detectAndCompute(l_img, None)
key_points2, descriptor2 = orb.detectAndCompute(r_img, None)
```

4 Conclusion

This project successfully implemented two fundamental computer vision applications. The coin detection system demonstrates robust performance in controlled environments, while the panorama stitching pipeline creates seamless wide-angle images. Future work could focus on improving illumination handling in coin detection and optimizing the stitching algorithm for multiple image inputs.

5 References

1. OpenCV Documentation
2. Computer Vision: Algorithms and Applications (Szeliski)
3. Digital Image Processing (Gonzalez & Woods)