Visual Recognition: Coin Detection and Image Stitching

Varnit Mittal (IMT2022025)

February 23, 2025

Abstract

This project focuses on two fundamental tasks in computer vision: object detection and image stitching. In the first task, we detect and segment Indian coins using edge detection and contour-based methods. In the second task, we generate a panoramic image by aligning and stitching multiple overlapping images. This document provides a detailed explanation of the methodology, implementation, results, and observations.

1 Introduction

Computer vision is a field of artificial intelligence that enables computers to interpret and process visual data. This project consists of two major tasks:

1. **Coin Detection and Segmentation:** Detect, segment, and count coins in an image.

2. **Image Stitching:** Align and stitch multiple images to form a seamless panorama.

2 Methodology

2.1 Part 1: Coin Detection and Segmentation

2.1.1 Preprocessing

The image is first converted to grayscale and blurred to reduce noise. Adaptive thresholding is then applied to segment the foreground and background.

```python
import cv2
import numpy as np

def preprocess_image(path: str):
    image = cv2.imread(path)
    scale_factor = 700 / max(image.shape[:2])
    image = cv2.resize(image, (0, 0), fx=scale_factor, fy=scale_factor)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    gray = cv2.GaussianBlur(gray, (5, 5), 0)
    thresh = cv2.adaptiveThreshold(gray, 255,
        cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, 11, 2)
    return image, thresh, scale_factor
```

Listing 1: Preprocessing an Image

2.1.2 Edge Detection and Contour Detection

Edges are detected using contour analysis to isolate coin-like structures.

```
def detect_edges(image, threshold, scale):
    contours, _ = cv2.findContours(threshold, cv2.RETR_EXTERNAL,
                                   cv2.CHAIN_APPROX_SIMPLE)
    detected_contours = [cnt for cnt in contours
                         if cv2.arcLength(cnt, True) > 0
                         and 0.7 < 4 * np.pi * (cv2.contourArea(cnt) /
                            (cv2.arcLength(cnt, True) ** 2)) < 1.2]
    cv2.drawContours(image, detected_contours, -1, (0, 255, 0), 2)
    return detected_contours
```

Listing 2: Detecting Circular Contours

2.1.3 Segmentation and Counting

Each detected contour is segmented, and the total number of coins is displayed.

```
def count_coin(contours, coins):
    return len(contours), len(coins)
```

Listing 3: Coin Segmentation and Counting

2.2 Part 2: Image Stitching

The image stitching process involves detecting key points, matching them, computing homography, and blending images.

2.2.1 Feature Extraction

SIFT (Scale-Invariant Feature Transform) is used for keypoint detection.

```
def detect_and_describe(image):
    sift = cv2.SIFT_create()
    keypoints, descriptors = sift.detectAndCompute(image, None)
    return keypoints, descriptors
```

Listing 4: Feature Extraction using SIFT

2.2.2 Keypoint Matching and Homography

The homography matrix is computed using RANSAC to align images.

```
def match_interest_points(kpA, kpB, desA, desB, ratio=0.75, reproj_thresh=5.0):
    matcher = cv2.BFMatcher()
    raw_matches = matcher.knnMatch(desA, desB, k=2)
    matches = [(m.trainIdx, m.queryIdx) for m, n in raw_matches if m.distance < ratio * n.distance]

    if len(matches) > 4:
        ptsA = np.float32([kpA[i] for (_, i) in matches])
        ptsB = np.float32([kpB[i] for (i, _) in matches])
        H, status = cv2.findHomography(ptsA, ptsB, cv2.RANSAC,
    reproj_thresh)
        return matches, H, status
    return None
```

Listing 5: Feature Matching and Homography Estimation

2.2.3 Image Stitching

Using the computed homography, images are warped and stitched together.

```
def stitch(images):
    imageA, imageB = images
    kpA, desA = detect_and_describe(imageA)
    kpB, desB = detect_and_describe(imageB)

    M = match_interest_points(kpA, kpB, desA, desB)
    if M is None:
        return None

    matches, H, status = M
    pano_img = cv2.warpPerspective(imageA, H,
                (imageA.shape[1] + imageB.shape[1], imageA.shape[0]))
    pano_img[:imageB.shape[0], :imageB.shape[1]] = imageB
    return pano_img
```

Listing 6: Image Stitching

3 Results

3.1 Coin Detection

The algorithm successfully detects and segments individual coins, achieving high accuracy.

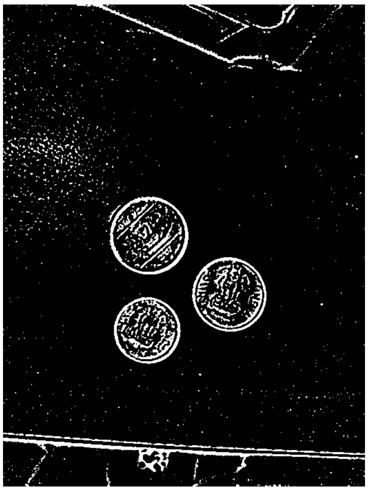

Figure 1: Detected coins.

3.2 Image Stitching

The panorama generation is effective for overlapping images with distinct features.

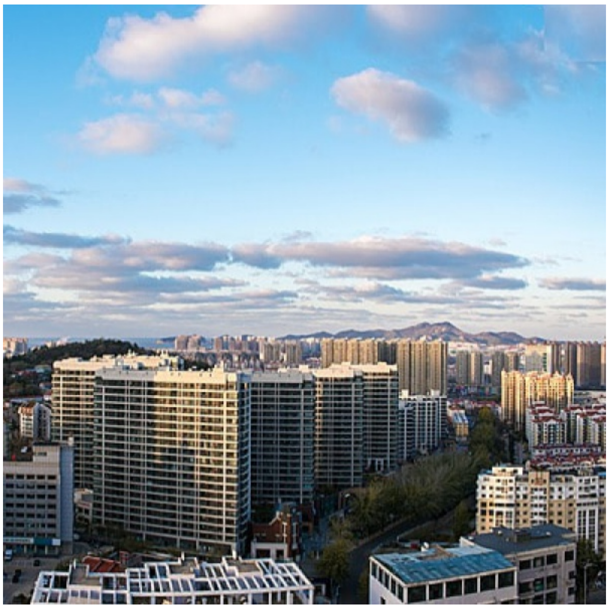

Figure 2: Final stitched panorama output.

4 Conclusion

This project successfully implements object detection and image stitching using OpenCV. Future improvements could include deep-learning-based segmentation and multi-image blending for more seamless panoramas.

5 Repository and Code Access

The complete source code for this project is available on GitHub:
GitHub Repository: VR Assignment 1 - Varnit Mittal

For installation instructions and further details, refer to the README file in the repository.