README

Coin Detection and Panorama Stitching

This project consists of two computer vision tasks using OpenCV:

Task 1: Coin Detection and Segmentation

- Detects and counts coins in an image using edge detection and contour extraction.
- Outputs various intermediate processing steps and the final detection result.

Task 2: Panorama Stitching

- Merges multiple overlapping images into a single wide-angle view using feature-based image stitching.
- Uses keypoint detection, feature matching, and homography transformation.

Both tasks employ OpenCV and NumPy for image processing and transformation.

Repository Structure

The repository is organized into two main folders and two files:

- **Task 1 (Coin Detection and Segmentation)**
- **Task 2 (Panorama Stitching)**
- **VR Assignment 1.pdf**
- **README.pdf**

Each task folder contains three subfolders:

1. **Experiment**: Includes images and Jupyter Notebook files used during experimentation with different methods.
2. **Final Submission**: Contains the main Python script along with input images required for execution.
3. **Output Images**: Stores the output images generated at different processing steps, including the final result.

Task 1: Coin Detection and Segmentation

Approach and Methods Used

1. Image Preprocessing

- **Grayscale Conversion**: Converts the image to a single-channel grayscale format.
- **Gaussian Blurring**: Reduces noise and smooths the image to improve edge detection.

2. Edge Detection

- **Canny Edge Detection**: Identifies the edges of coins, helping in segmentation.

3. Contour Detection and Segmentation

- **Finding Contours**: Uses `cv2.findContours()` to detect the boundaries of objects.
- **Drawing Contours**: Outlines detected coins on the original image for visualization.

How to Run the Code

Prerequisites

- Python 3.x
- OpenCV (`cv2`)
- NumPy
- Matplotlib

Installation

```
pip install opencv-python numpy matplotlib
```

Execution Steps

1. Place the input image (`IndianCoins.jpg`) in the project directory.
2. Run the script:
   ```
   python coin_detection.py
   ```
3. The script will display processed images and save them in the directory.

Output Images

1. **Grayscale.jpg** - Grayscale version of the input image.
2. **Blurred.jpg** - Image after applying Gaussian blur.
3. **Canny_Edges.jpg** - Edge-detected image using the Canny algorithm.
4. **Detected_Coins.jpg** - Final image with detected coins outlined.

Observations and Results

- The script successfully detects and counts coins in the image.
- Gaussian blurring improves segmentation by reducing noise.
- Canny edge detection effectively highlights the boundaries.
- Contour detection accurately outlines coins.

Example Output

Total number of coins detected: 22
Processed images saved successfully.

Task 2: Panorama Stitching

Approach and Methods Used

1. Feature Detection and Description

- **ORB (Oriented FAST and Rotated BRIEF)**: Extracts keypoints and computes feature descriptors.

2. Feature Matching

- **Brute Force Matcher**: Matches keypoints between overlapping images using Hamming distance.

3. Homography Computation

- **RANSAC (Random Sample Consensus)**: Finds a robust transformation matrix to align images.

4. Image Warping and Blending

- **Perspective Warping**: Uses homography to transform images.
- **Blending Techniques**: Smooths transition areas between stitched images.

How to Run the Code

Prerequisites

- Python 3.x
- OpenCV (`cv2`)
- NumPy

Installation

```
pip install opencv-python numpy
```

Execution Steps

1. Place input images (`image_1.jpg, image_2.jpg, image_3.jpg`) in the project directory.
2. Run the script:
   ```
   python panorama_stitching.py
   ```
3. The script will process and merge the images into a single panoramic view.

Output Images saved in the output_images folder

1. **left_grayscale.jpg, center_grayscale.jpg, right_grayscale.jpg** - Grayscale versions of the left, center, and right images used for stitching.
2. **Image1_keypoints.jpg -** Displays key points detected in both the left and center images, which are later stitched.
3. **image2_keypoints.jpg** - Displays key points detected in both the center and right images, which are later stitched.
4. **stitched_panorama.jpg** - The stitched panoramic output.

Observations and Results

- ORB successfully detects and matches key points across images.
- RANSAC improves alignment by filtering out incorrect matches.
- Homography transformation accurately warps images for seamless blending.
- Final panorama output maintains high resolution and minimal distortions.

Conclusion

- **Task 1** demonstrates effective object detection using contour-based segmentation.
- **Task 2** achieves seamless image stitching through feature-based alignment.
- Both tasks leverage OpenCV's powerful image processing capabilities for real-world applications.

Author: Ayush Arya Kashyap
Date: 13 February 2025