Assignment Part 1: Coin Segmentation

Overview

This Python script processes an image of coins to detect and segment individual coins using OpenCV. It applies several image preprocessing techniques, including:

- **Grayscale conversion**
- **Blurring**
- **Edge detection**
- **Morphological transformations**
- **Contour detection**

The goal is to accurately **segment** the coins and **count** the total number present in the image.

Cloning the Repository

To get started, first clone this repository:

```
git clone https://github.com/sohith18/VR-Assignment-1.git
cd VR-Assignment-1
```

Dependencies

Ensure you have the following Python libraries installed:

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

Alternatively, you can install dependencies using a **virtual environment** with Conda:

```
conda create --name vrenv python=3.12
conda activate vrenv
pip install -r requirements.txt
```

This ensures all dependencies are installed in an isolated environment.

Running the Script

1. Place the `Coins.jpg` image in the same directory as the script.
2. Ensure the `seg_coins/` folder exists in the same directory.
3. Run the script using:

```
python Assignment_part1.py
```

4. Segmented coins will be saved in the `seg_coins/` folder.

Displayed Output

When the script runs, **three windows** will pop up:

- **Canny Edges:** Displays detected edges in the image.
- **Thresholded Image:** Shows the binary threshold applied to separate the coins from the background.
- **Segmented Mask:** Displays the mask used to extract individual coins.

The script will also print the total number of detected coins.

Methods Used & Observations

Image Resizing

- The image is resized to **25% of its original size** to speed up processing and improve visualization.

Grayscale Conversion

- Converts the image to **grayscale** to simplify processing by reducing color channels.

Gaussian Blurring

- A **Gaussian blur** is applied to reduce noise and smooth the image.
- A **(3,3) kernel size** was found to be the most effective, as it removed fine details on the coins while preserving their shape.

Canny Edge Detection

- The **Canny algorithm** is used to detect edges in the image.
- This helps outline the coins but is not used directly for contour detection due to potential gaps in edges.

Thresholding

- **Otsu's thresholding** is applied to create a **binary image**, enhancing coin-background separation.

Morphological Transformations

- **Morphological closing** (dilation followed by erosion) is applied to remove small gaps and noise.
- A **(5,5) kernel size** was chosen as it effectively cleaned up noise while maintaining coin shapes.

Contour Detection & Filtering

- **Contours** are detected from the processed binary image using `cv2.findContours()`.
- Only **large contours (area > 15,000 pixels)** are considered as valid coins.
- Each detected contour is drawn on a mask, and individual coin images are saved.

Counting Coins

- The total number of detected coins is determined by counting the **filtered contours**.

Results

- The script **successfully segments** the coins and **counts** them.
- Each coin is saved separately in the `seg_coins/` folder with a unique filename.
- The selected parameters were found to be **optimal** for this specific image.

Assignment Part 2: Creating Panorama

Overview

This Python script performs image stitching to create a panorama from multiple images using OpenCV. The script applies feature detection, keypoint matching, homography estimation, and image blending techniques to seamlessly stitch images together.

Cloning the Repository

To get started, first clone this repository:

```
 git clone https://github.com/sohith18/VR-Assignment-1.git
cd VR-Assignment-1
```

Dependencies

Ensure you have the following Python libraries installed:

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

Alternatively, you can install dependencies using a **virtual environment** with Conda:

```
 conda create --name vrenv python=3.12
conda activate vrenv
pip install -r requirements.txt
```

This ensures all dependencies are installed in an isolated environment.

Running the Script

1. Place the two input images in the `images/` directory and name them `img1.jpg` and `img2.jpg`.
2. Run the script:

```
python Assignment_part2.py
```

3. The resulting stitched panorama will be displayed and saved as `Panaroma.jpg` in the same directory.

Displayed Output

When the script runs, **four windows** will pop up:

- **Feature Matching:** Displays the matched keypoints between the two images.
- **Distance Transform 1 & 2:** Shows the distance transform of the images used for blending.
- **Panorama:** Displays the final post-processed stitched image.

Methods Used with Observations

Image Preprocessing

- **Grayscale Conversion:** Converts the input images to grayscale for feature detection.
- **Rescaling:** Images are resized to 15% of their original size to improve computational efficiency.

Feature Detection and Matching

- **SIFT Keypoint Detection:** The SIFT (Scale-Invariant Feature Transform) algorithm detects keypoints and computes descriptors.

- **Brute-Force Matching:** The BFMatcher with L2 norm and cross-checking finds the best matches between features in the two images.
- **Filtering Matches:** Matches are sorted based on distance, and the top 75 matches are used for accuracy.

Homography Estimation

- **RANSAC Algorithm:** Homography matrix estimation removes outliers and finds a perspective transformation using at least four corresponding points.
- **Warp Perspective:** The second image is warped using the homography matrix to align it with the first image.

Image Blending

- **Distance Transform:** Distance maps are computed for smooth blending.
- **Seam Smoothing:** The overlapping regions of the images are blended based on their distance transform values to reduce visible seams.

Results

- Successfully stitches the input images into a panorama.
- Uses feature matching and homography estimation for accurate alignment.
- Displays the final stitched image on the screen.
- Saves the stitched panorama as `Panaroma.jpg` in the same directory.
- Seam blending technique enhances the visual appearance of the final image.