1. Coin Detection

Objective

The goal of this script is to detect and count the number of coins in an image using OpenCV's image processing techniques.

What Was Tried

1. **Grayscale Conversion**:
    - Converted the original image to grayscale to simplify processing.
    - Worked well as it helped in edge detection and contour identification.
2. **Gaussian Blurring**:
    - Applied a Gaussian blur to reduce noise and smoothen the image.
    - Helped in eliminating minor details while retaining prominent coin edges.
3. **Edge Detection Using Canny**:
    - Used the Canny edge detector to identify edges in the image.
    - This step was crucial as it highlighted the outlines of the coins.
4. **Dilation**:
    - Applied dilation to make detected edges more pronounced.
    - Helped in joining broken edges, improving contour detection.
5. **Contour Detection**:
    - Used `cv2.findContours()` to identify the individual coins.
    - The count of coins was obtained based on the number of detected contours.

What Worked

The overall pipeline successfully detected and counted coins in the image.
Contour detection provided an accurate count when coins were well separated.
Visualization using `matplotlib` clearly displayed the different stages of processing.

What Didn't Work

If coins were overlapping or touching, they were sometimes detected as a single object.
Small variations in lighting conditions affected edge detection accuracy.
Some noise was misidentified as coins due to improper thresholding.

Final Approach

- The final approach retained grayscale conversion, Gaussian blurring, and Canny edge detection.
- Dilation was used to strengthen weak edges.

- The `cv2.findContours()` function successfully extracted the number of coins.
- Future improvements could involve applying adaptive thresholding or using Hough Circles for better detection.

2. Panorama Stitching

Objective

The goal of this script is to stitch multiple images together to create a panorama using OpenCV's `Stitcher_create()` function.

What Was Tried

1. **Reading Input Images**:
    - Initially, image paths were loaded incorrectly as lists of file paths instead of actual images.
    - Fixed by using `cv2.imread()` to read the images before passing them to the stitcher.
2. **Feature Detection Using SIFT**:
    - Used the Scale-Invariant Feature Transform (SIFT) algorithm to detect keypoints.
    - This step helped to identify overlapping regions between images.
3. **Image Stitching Using OpenCV's Stitcher**:
    - Applied OpenCV's `cv2.Stitcher_create()` to merge images.
    - The function automatically aligns, warps, and blends images.
4. **Visualization of Keypoints**:
    - Once stitched, SIFT was used to detect keypoints in the final panorama.
    - Keypoints were drawn on the final stitched image.

What Worked

Image stitching successfully worked when images had a significant overlapping region.
The use of SIFT ensured that keypoints were detected accurately.
The final output was saved and displayed properly.

What Didn't Work

If the images had poor overlap, stitching failed or produced distortions.
If there was a significant difference in lighting, the transition between images was noticeable.
The Stitcher sometimes failed without a clear reason (returning an error code).

Final Approach

- Ensured images were correctly loaded before stitching.
- Used SIFT to detect features and enhance alignment.
- Applied `cv2.Stitcher_create()` for automatic stitching.
- The final image was saved and displayed with key points highlighted.
- Future improvements could involve manually refining feature matching or using RANSAC filtering.