# Part 1

# Coin Detection

The code in this part processes images to detect "coins" using OpenCV. It applies image preprocessing, contour detection, and circularity filtering to identify coins. One can check for multiple images of coins within the same input folder.

## Features

- **Image Preprocessing**: Converts images to grayscale, applies Gaussian blur, and performs adaptive thresholding.
- **Contour Detection**: Finds contours and filters them based on circularity and area constraints.
- **Image Segmentation**: Draws detected contours and segments objects using masking.
- **Batch Processing**: Processes all images in a specified folder automatically.

## Code Overview

### Functions:

- `load_and_prepare(img_path)`: Loads an image, resizes it, converts to grayscale, applies blur and thresholding.
- `detect_coins(binary_img, scale)`: Detects circular objects based on contour analysis and circularity measures.
- `store_images(output_img, shapes, save_path, draw_type)`: Saves images with detected contours or segmentation masks.
- `batch_process(input_dir, output_dir)`: Processes multiple images in a directory.

## Example Output

After processing, the output folder will contain:

- **Original with contours**: `<filename>_outline.jpg`
- **Segmented version**: `<filename>_segmented.jpg`
- The number of coins corresponding to the particular image will get printed on the terminal.

# Part 2

# Image Stitching

The code in this part stitches multiple overlapping images to form one new panaromic image with the help of SIFT, Homography and other image stitching methods. One can run and store the image stitching results for multiple scenes.

# Features

- **SIFT Feature Detection**: Extracts keypoints and descriptors from images.
- **Feature Matching**: Uses a Brute Force Matcher (BFMatcher) with a ratio test for optimal matches.
- **Homography Estimation**: Computed the homography using RANSAC (Random Sample Consensus) to warp one image onto another.
- **Image Warping & Blending**: Uses perspective transformation to stitch images together.
- **Automatic Cropping & Match Visualization**: Removes black borders from the final panorama and draws lines connecting keypoints between images for debugging.

# Code Overview

## Functions:

- `extract_features(gray_img)`: Extracts keypoints and descriptors using SIFT.
- `get_matches(kp1, kp2, desc1, desc2, ratio_thresh, reproj_thresh)`: Finds the best matching keypoints between two images using the ratio test and with RANSAC.
- `visualize_matches(img1, img2, kp1, kp2, matches, match_status)`: Draws lines between matched keypoints for debugging purposes.
- `refine_image(img)`: Crops the stitched image to remove black borders.
- `merge_images(pair, ratio=0.75, reproj=4.0, debug=False)`: Stitches two images together using homography transformation.
- `process_directory(src_folder, dest_folder)`: Processes all images in a folder and stitches them sequentially into a panorama.

# Example Output

After processing, the output folder will contain:

- **The final panorama of the images**: `<filename>_panaroma.jpg`
- **Matched images**: `<filename>_match.jpg`

# Requirements

Ensure you have the following dependencies installed:

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

# Usage

1. Place input images in the `input/` folder.
2. Run the code<code_name> using:

   ```
   python <code_name>.py
   ```

3. Processed images will be saved in the `output/` folder with contours and segmented versions for the first part and the stitched final image along with matched interest point between images for the second part of the individual images present in the input folder.

# File Structure

```
├── input/                  # Input images
├── output/                # Processed images with detected coins
├── <code_name>.py              # Main Python script/code
├── README.md              # Project documentation
```

Link to the Github Repository: https://github.com/MVedant21/VR_Assignment1_Vedant_IMT2022519/ (https://github.com/MVedant21/VR_Assignment1_Vedant_IMT2022519/)