

# Visual Recognition - Assignment 1

Siddeshwar Kagatikar, IMT2022026

February 28, 2025

Project Repository Link

## 1 Objectives

- **Part 1: Coin Detection and Segmentation** - Detect, segment, and count Indian coins from an image using computer vision techniques.
- **Part 2: Image Stitching** - Generate a stitched panorama from multiple overlapping images.

## 2 Project Structure

```
VR_Assignment1_SiddeshwarKagatikar_IMT2022026/
| Part1/                      # Coin Detection & Segmentation
|   input_coins/               # Input images (coins)
|   contours/                  # Contour-detected images
|   Edges/                     # Edge-detected images
|   segments/                  # Segmented coins
|     | coin1/
|     | coin2/
|     | coin3/
|     | coin4/
|   VR1.ipynb                 # Jupyter Notebook for Part1
|
| Part2/                      # Image Stitching (Panorama Creation)
|   images/                     # Input images (overlapping scenes)
|   output/                    # Stitched outputs
|     | stitchedOutput.png
|     | stitchedOutputProcessed.png
|   VR2.ipynb                 # Jupyter Notebook for Part2
|
| requirements.txt            # Dependencies
| link.txt                    # GitHub Link
| README.md                   # README file
```

### 3 Task 1: Detect, Segment, and Count Coins

#### 3.1 Methods Tried

- Initially, contour detection was capturing not only the outer boundary of the coins but also internal details, leading to fragmented contours instead of complete shapes.
- Different edge detection thresholds were tested in `cv2.Canny()` to balance between detecting the coin outline and avoiding excessive internal edges.
- Morphological operations such as `cv2.morphologyEx()` with closing were applied to smoothen gaps and enhance contour formation, but fine-tuning was needed to prevent merging of closely placed coins.

#### 3.2 Steps and Methods Used

##### Preprocessing:

- Convert image to grayscale using `cv2.cvtColor()`.
- Apply Gaussian blur with `cv2.GaussianBlur()`.
- Use Canny edge detection with `cv2.Canny()`.

##### Contour Detection:

- Detect contours with `cv2.findContours()`.
- Approximate shapes with `cv2.approxPolyDP()`.

##### Segmentation and Counting:

- Extract bounding rectangles using `cv2.boundingRect()`.
- Count detected coins and display the count.

#### 3.3 Results

- Total Coins Detected: Displayed in console.
- Segmented Coins Saved: `Part1/segments/`.

### 4 Observations for Coin Detection and Segmentation

#### 4.1 Preprocessing Outputs

- **Grayscale Image:** Converts the original image to gray-scale for simplified processing.

- **Blurred Image:** Gaussian blur reduces noise and prevents false edge detections.
- **Edge Detection Output:** Canny edge detection extracts strong edges to define coin boundaries.

## 4.2 Contour Detection Outputs

- **Detected Contours Image:** The algorithm outlines the coins detected in green.
- Non-circular objects are filtered out to avoid false detections.



Figure 1: Detected Coins- Image1



Figure 2: Detected Coins - Image2



Figure 3: Detected Coins - Image3



Figure 4: Detected Coins - Image4

### 4.3 Segmentation Outputs

- Individual coin images are saved in Part1/segments/.
- Helps validate if overlapping coins are segmented correctly.

### 4.4 Coin Counting Output

- The total number of coins detected is displayed on the console.
- Any discrepancy in count suggests possible segmentation issues (e.g., overlapping coins being merged).

### 4.5 Results

- **Total Coins Detected:** (Displayed in console)
- **Edges of Coins Saved:** Part1/Edges.
- **Segmented Coins Saved:** Part1/segments.
- **Visual Output:** /Part1/contour.

## 5 Task 2: Create a Stitched Panorama

### 5.1 Methods Tried

- **Manual Homography Estimation:** Attempted to estimate homography using manually selected keypoints, but it was inaccurate and required excessive fine-tuning.
- **Pairwise Blending Instead of Stitching:** Tried blending adjacent images without feature matching, but alignment was poor, causing ghosting effects.

### 5.2 Steps and Methods Used

1. **Feature Detection & Matching:** ORB feature detector was used to find and match keypoints between consecutive images.
2. **Pairwise Stitching:** The OpenCV Stitcher API was applied to stitch adjacent images together into partial panoramas.
3. **Final Panorama Generation:** All images were stitched together using the Stitcher API to create the full panorama.
4. **Post-Processing & Cropping:** Borders were removed using contour detection, and the final stitched image was cropped for a clean output.

### 5.3 Loading Images

**Objective:** Read input images from the directory and prepare them for processing.

**Methods:**

- Fetch image paths from the `./images/` folder using `glob.glob()`.
- Sort filenames to maintain stitching order.
- Read images using `cv2.imread()`.

### 5.4 Displaying Input Images

**Objective:** Show input images before processing.

**Methods:**

- Convert images to RGB using `cv2.cvtColor()`.
- Display images using `matplotlib.pyplot`.

### 5.5 Feature Extraction and Matching

**Objective:** Identify and visualize corresponding keypoints between images.

**Methods:**

- Detect keypoints using the ORB algorithm (`cv2.ORB_create()`).
- Extract descriptors for each keypoint.
- Match keypoints using `cv2.BFMatcher()`.
- Filter good matches by sorting based on distance.
- Draw thicker matching lines with unique colors using `cv2.line()`.

### 5.6 Visualizing Keypoints

**Objective:** Display and save keypoint matches.

**Methods:**

- Draw matched keypoints with thicker lines and random colors.
- Store output images for verification.

### 5.7 Image Stitching (Pairwise Stitching)

**Objective:** Merge images step by step to form an intermediate panorama.

**Methods:**

- Use `cv2.Stitcher_create()` to stitch two images.
- Save intermediate stitched images if stitching is successful.

## 5.8 Final Panorama Stitching

**Objective:** Merge all images into a complete panorama.

**Methods:**

- Use `cv2.Stitcher_create()` to stitch all images.
- Save the final output if stitching is successful.

## 5.9 Post-Processing and Cropping

**Objective:** Remove black borders and refine the final panorama.

**Methods:**

- Convert the stitched image to grayscale (`cv2.cvtColor()`).
- Apply binary thresholding (`cv2.threshold()`).
- Detect contours (`cv2.findContours()`).
- Extract the largest contour and create a bounding box.
- Crop the stitched image based on bounding box coordinates.

# 6 Observations for Image Stitching and Panorama Generation

## 6.1 Feature Detection and Matching Outputs

- Keypoints are detected using ORB.
- Descriptors are matched using the BFMatcher algorithm.
- Matched keypoints between consecutive images are stored in `keypoints_1_2.png` and `keypoints_2_3.png`.

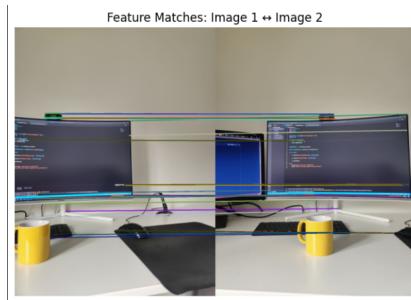


Figure 5: Feature Map of Image 1,2

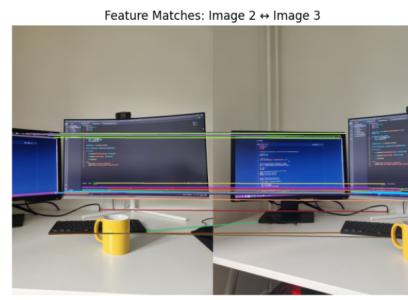


Figure 6: Feature Map of Image 2,3

## 6.2 Intermediate Panoramas

- Images are stitched in pairs to generate intermediate panoramas.
- The intermediate results are saved for verification.

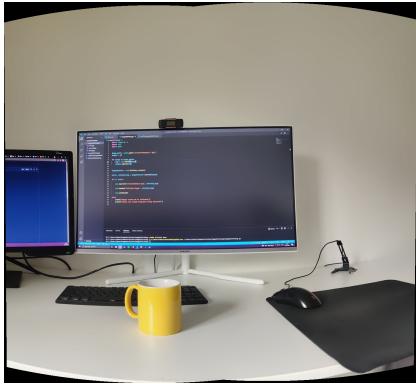


Figure 7: Panorama of Image 1,2

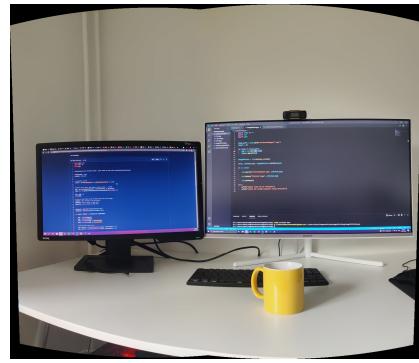


Figure 8: Panorama of Image 2,3

## 6.3 Final Panorama Output

- The final stitched panorama is stored as `stitchedOutput.png`.
- Post-processing removes black borders and enhances the result.
- The refined image is stored as `stitchedOutputProcessed.png`.

## 6.4 Results

- **Keypoints Matched:** Visualized in `keypoints_1_2.png` and `keypoints_2_3.png`.
- **Intermediate Panoramas:** Saved as intermediate stitched images.
- **Final Panorama:** Saved as `stitchedOutput.png` and `stitchedOutputProcessed.png`.

## 7 What I Learned from This Project

This project provided valuable insights into computer vision, image processing, and object detection techniques. Key learnings include:

- **Preprocessing Techniques:** Importance of grayscale conversion, Gaussian blurring, and edge detection.

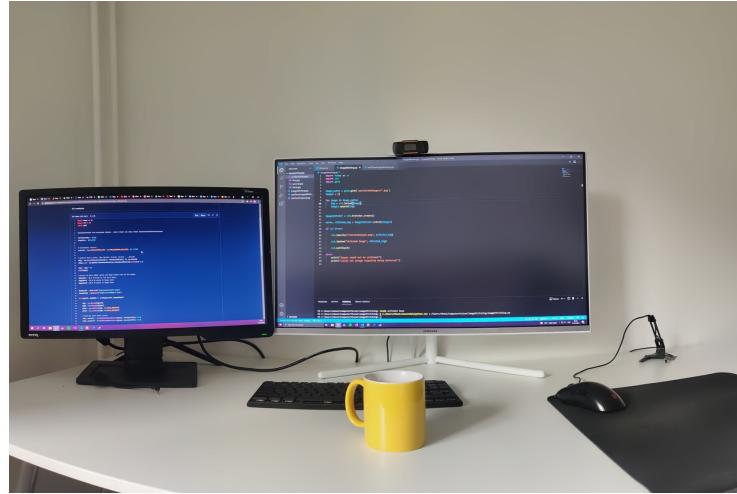


Figure 9: Final stiched panorama

- **Feature Detection and Matching:** Extracting and matching keypoints and descriptors using ORB.
- **Contour Detection and Object Segmentation:** Using contour-based approaches for object detection.
- **Homography and Image Transformation:** Understanding homography matrices for image alignment.
- **Challenges in Computer Vision:** Addressing issues like incorrect keypoint matching, misalignment, and segmentation errors.
- **Post-processing Refinements:** Techniques such as morphological operations, cropping, and bounding box filtering to enhance accuracy.