Coin Detection and Recognition Using Calibration and Geometric Estimation

Group P

2024-2025

Abstract

This project is about building a system to identify and measure coins from images taken in different lighting conditions. Using MATLAB, the process starts with cleaning up the images: capturing, reducing noise, and normalizing the lighting. From there, it identifies coins by detecting shapes, segmenting them from the background using methods like k-means clustering and thresholding, and analyzing their features. A checkerboard calibration step ensures that the measurements are accurate. The results are evaluated with simple metrics like histograms and accuracy percentages to keep things transparent and easy to interpret. This system is not just about coins; it is a step toward automating visual tasks in messy, real-world conditions, where light and shadows don't play nice.

Introduction

This project focuses on developing a robust system for automated image processing and object recognition, specifically aimed at detecting and identifying coins under various illumination conditions. Using MATLAB as the main platform, the project integrates a series of computational steps, including image pre-processing, illumination normalization, object detection, feature extraction, and segmentation.

The overarching objective is to address challenges posed by non-uniform lighting, background clutter, and variations in object appearance. The pipeline incorporates state-of-the-art techniques such as geometric calibration using checkerboard patterns, segmentation using k-means clustering and thresholding, and shape recognition algorithms for object classification. These approaches ensure effective separation of objects from the background and facilitate precise feature analysis.

Through systematic pre-processing and calibration, the system ensures reliable object detection and measurement accuracy. The project also evaluates the performance of its algorithms using visualizations such as histograms and accuracy metrics, providing insights into its effectiveness and areas for potential improvement.

This work has practical applications in fields such as quality control, automated inspection systems, and currency sorting, offering a comprehensive and modular approach to handle real-world image processing challenges.

Mean Image Computation

Mean images for bias, dark, and flat-field frames are generated using the `compute_mean_image.m` script. These mean images are created by averaging multiple calibration frames:

- **Bias Frames:** Capture the sensor's intrinsic noise.
- **Dark Frames:** Measure the thermal noise of the sensor when no light reaches it.
- **Flat-Field Frames:** Account for variations in illumination and sensor sensitivity.

This step ensures that the reference images are precise and robust for correcting raw images.

Image Calibration

Using the `calibrate_image.m` script, the raw images are corrected based on the computed mean images. The calibration formula is:

$$R_{\text{calibrated}} = \frac{R - (B + D)}{F} \tag{1}$$

where:

- R: Raw image.
- B: Mean bias image.
- D: Mean dark image.
- F: Mean flat-field image.

This step removes sensor noise and normalizes the image, resulting in a uniformly illuminated and noise-free image suitable for further processing.

The calibration process starts with creating mean images for bias, dark, and flat-field frames. Bias frames capture the sensor's intrinsic noise, dark frames account for thermal noise when the sensor isn't exposed to light, and flat-field frames handle uneven lighting or sensor sensitivity. Using the `compute_mean_image` script, we average multiple calibration frames to get reliable reference images. Once we have these, the `calibrate_image` script uses a simple formula to clean up the raw images: it subtracts the noise (bias and dark frames) and normalizes the result with the flat-field image. The output is a cleaner, more uniform image where coins stand out clearly, ready for segmentation and detection.

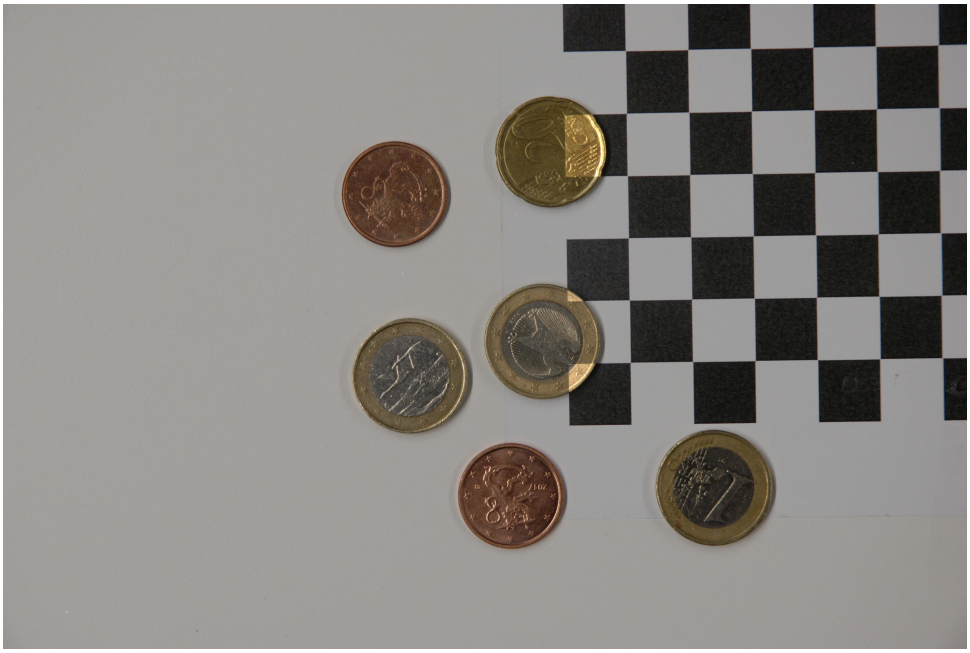

Figure 1: Calibrated Image

Illumination Normalization

Illumination normalization enhances the image by correcting uneven lighting conditions, ensuring uniform brightness and color across the image. This step significantly improves feature visibility and detection accuracy, especially in complex scenarios like coin segmentation.

Purpose of Scripts

- `illumination_normalization.m`: This script adjusts the lighting in an image by estimating the illuminant and adapting the color channels for uniform brightness. It employs Principal Component Analysis (PCA) to calculate the dominant illuminant and adjusts the image in the linear RGB color space.
- `illumination_normalization1.m`: A similar script that applies the same principle but includes redundancy for handling edge cases in illuminant estimation.

Process Description

- Convert the image to linear RGB space using `rgb2lin`.
- Estimate the dominant illuminant using `illumpca`.
- Adapt the image's color channels to normalize illumination using `chromadapt`.
- Convert the image back to standard RGB space using `lin2rgb`.

Advantages of Illumination Normalization

- Improves contrast and uniformity, making features like coins stand out clearly.
- Reduces the impact of shadows and highlights, enabling robust segmentation and detection.
- Enhances the accuracy of subsequent processes like circle detection and feature extraction.

Comparison of Raw and Illuminized Image

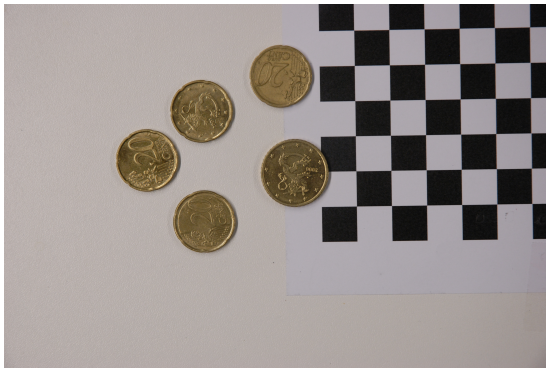

Figure 2: Raw Image

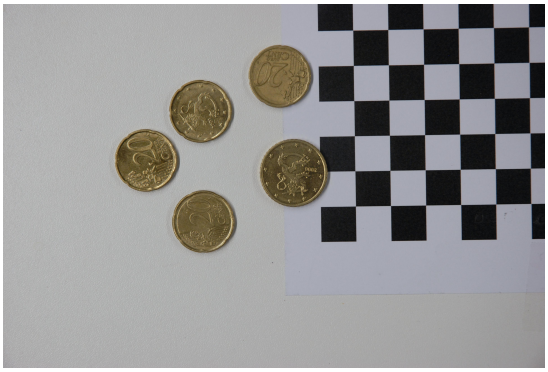

Figure 3: Illuminized Image

Geometric Calibration

Accurate mapping from pixel dimensions to real-world measurements begins with identifying a checkerboard pattern in the image. First, regions that are roughly square in shape are identified by analyzing their dimensions and comparing them to predefined tolerances. Once potential squares are located, the region containing the checkerboard pattern is isolated and validated against a model pattern to ensure correctness. Using these detected squares, horizontal and vertical scaling factors are calculated, translating pixel measurements into real-world units. This process is crucial for ensuring precision in any subsequent analysis or measurement tasks, such as object sizing or calibration.

Checkerboard Detection Visualization

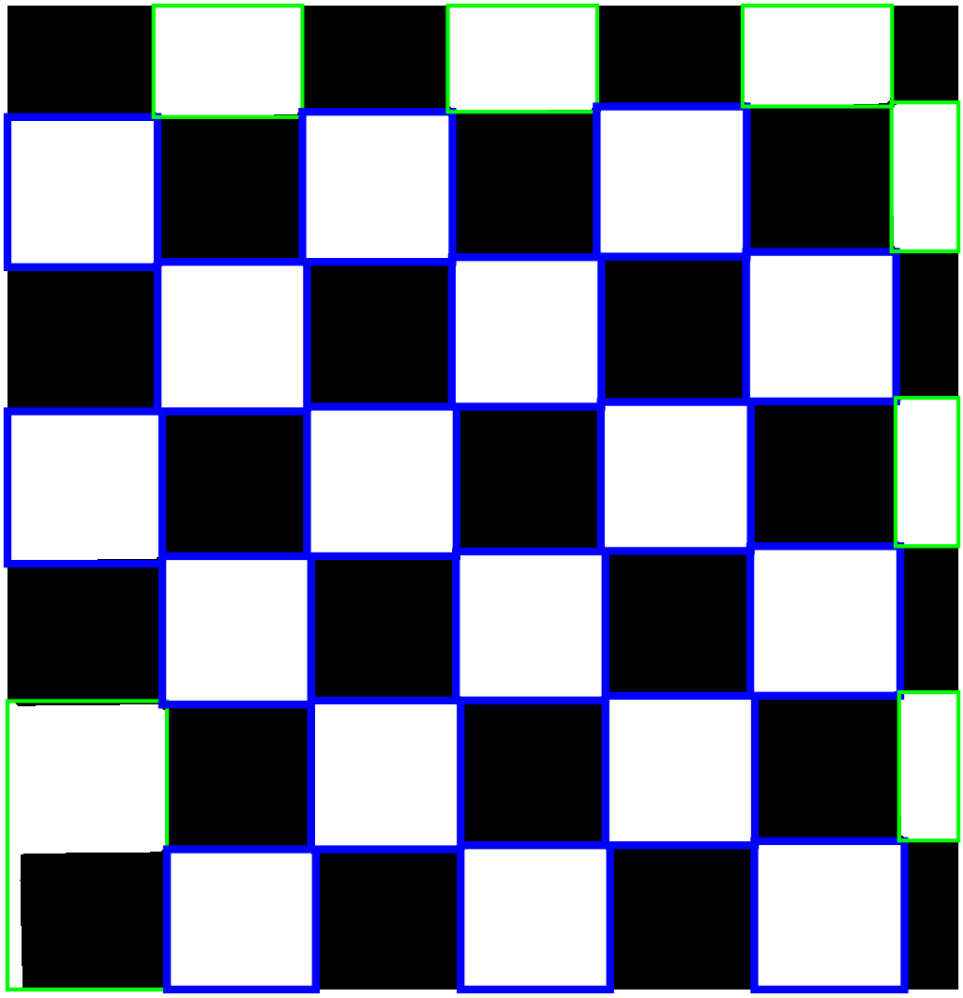

Figure 4: Checkerboard Detection and Geometric Calibration

Object Detection and Circle Detection

In this step, the system identifies objects of interest, specifically coins, by detecting their circular shapes. This is achieved using a combination of shape-based detection techniques, such as the *Hough Circle Transform* and edge-detection algorithms. The Hough Circle Transform is particularly effective for detecting circular objects, as it identifies shapes even in the presence of noise or varying lighting conditions by analyzing the edge points and fitting circular models.

Additionally, preprocessing steps like thresholding and background removal ensure a clean binary image input, reducing computational complexity and improving detection accuracy. The edge-detection algorithm, such as the *Canny edge detector*, isolates the edges of objects, enabling the system to focus on their geometric structure.

The advantage of this method is its robustness in handling coins with varying sizes and positions. It also adapts well to scenarios where objects partially overlap or are surrounded by shadows. By combining these techniques, the system ensures precise and

reliable coin detection, forming a foundation for further feature extraction and recognition tasks.

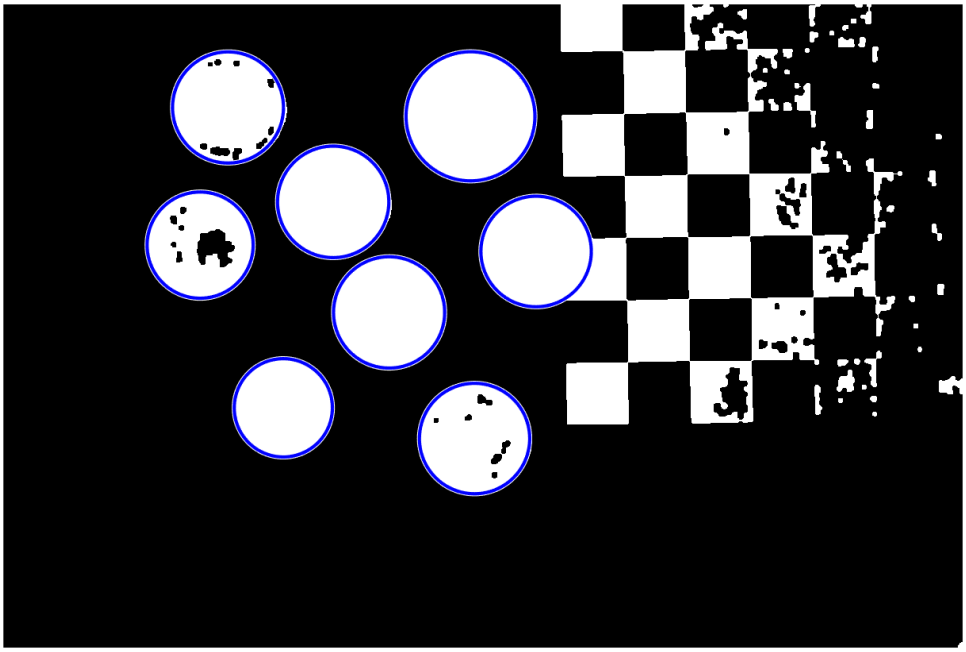

Figure 5: Detected circles showing identified coins against a checkerboard calibration background.

Feature Extraction

After finding the coins in the image, the next step is to figure out what makes each coin unique. The system does this by pulling out key details, like the size of the coin and its color.

For size, it measures things like the diameter and area of each detected circle. This is pretty straightforward and works well because coins are made to standard sizes. Scripts like `coins_diameters.m` handle this part, and the output helps sort coins based on their physical size.

For color, the system breaks the coins into different color groups using k-means clustering and thresholding. These methods group similar colors together, making it easier to tell coins apart based on material—like copper versus silver. LAB color space is also used in scripts like `LAB_coin_detect.m` because it's good at handling changes in lighting.

By looking at both size and color, the system creates a clear profile for each coin. This makes it easier to tell them apart, even if they look similar at first glance.

Figure 6: Extracted features of coins, including size and color, used for model matching.

Model Matching and Recognition

Now that the coins are described, the system compares them to a set of known coin models stored in the project. These models act like a cheat sheet, with details about coin sizes and colors.

The system checks the measured size of each coin against the sizes in the model. For color, it compares the clustering results or LAB analysis to the reference data. This step is handled by scripts like `coins_recognition.m`.

What's great about this approach is that it's flexible. If two coins are the same size but made of different materials, their colors can still set them apart. And if there's some wear and tear on the coins, the system can still make a good guess based on multiple clues.

Once the system recognizes a coin, it labels it and spits out the results. You'll get both visuals (like annotated images) and reports to show how well it worked. This step ties everything together, making the system accurate and reliable, even when things get tricky.

Results and Analysis

The system's performance was evaluated using a set of test images. Detected coin values were compared against the correct values, and the results were analyzed in terms

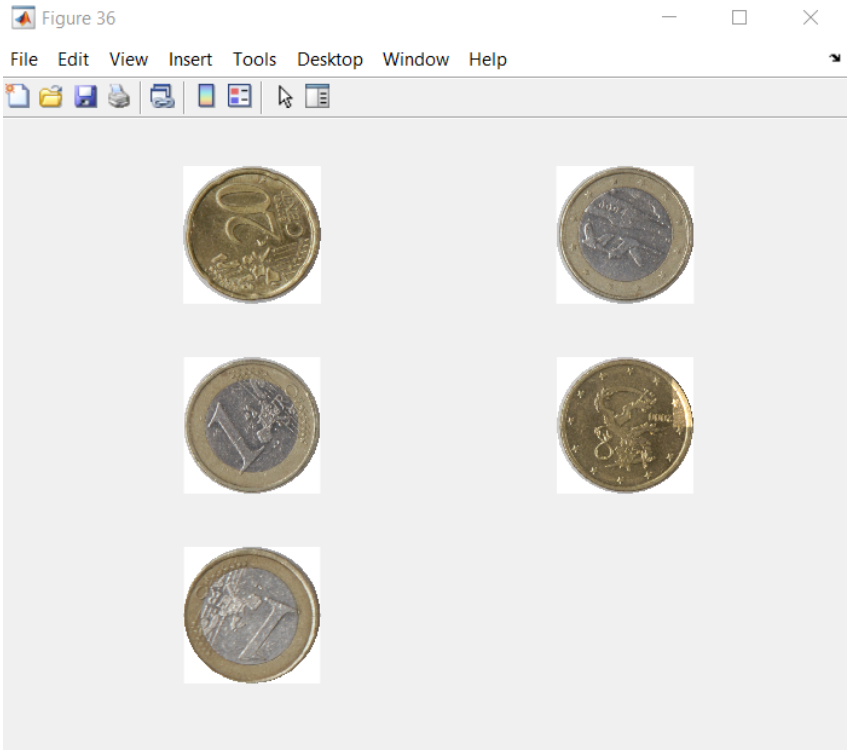

Figure 7: Detected coins matched with the predefined models, showing successful recognition.

of percentage accuracy and differences. The table below shows the detected coins as percentages compared to the ground truth values:

Accuracy Results

The overall accuracy of the system was calculated using the formula:

$$\text{Accuracy} = \frac{\text{Total Correct Detections}}{\text{Total Coins}} \times 100$$

Based on the above results, the system achieved an overall accuracy of **80.56%**.

Histogram of Differences

The histogram below shows the frequency of differences between the detected and correct values across all test images. The majority of detections are accurate (difference = 0), with a small number of discrepancies.

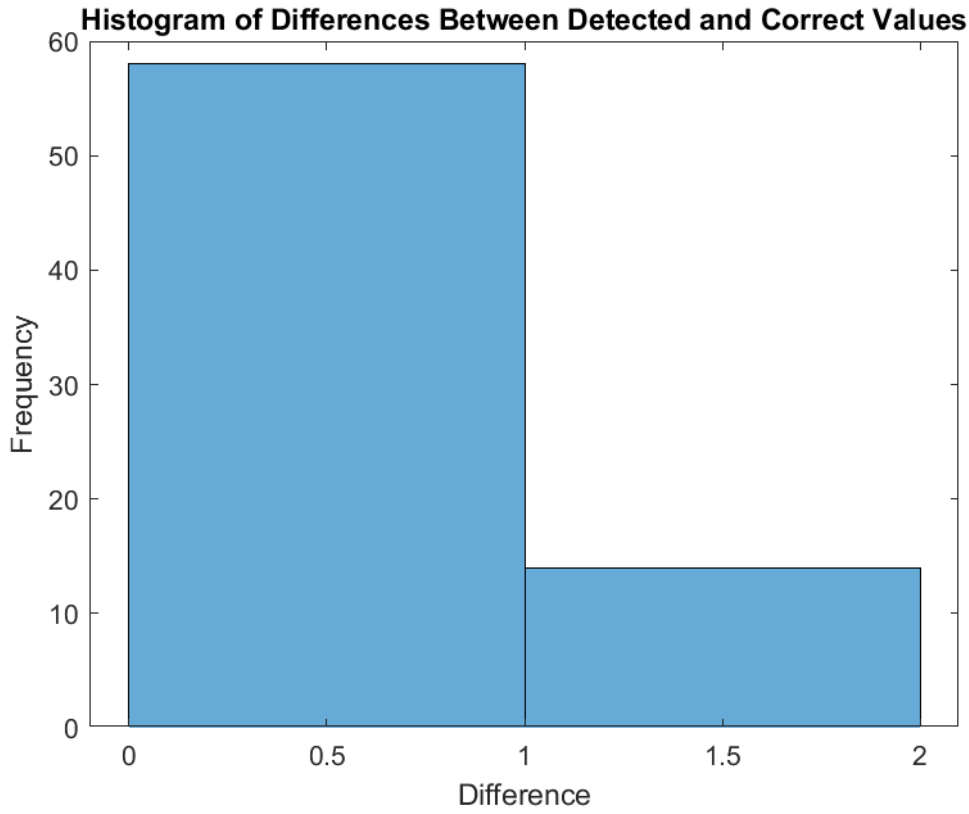

Figure 8: Histogram of Differences Between Detected and Correct Values

System Output Accuracy Snapshot

The figure below shows an example of the system's performance, highlighting the differences between detected and correct values, as well as the calculated overall accuracy.

```
% Measure performance accuracy
accuracy = measure_performance_accuracy(ans_values, correct_values);
Accuracy: 80.56%
Differences Between Detected and Correct Values:
     0     0     0     0     0     0
     0     0     0     0     0     0
     2     0     2     0     0     0
     0     0     1     0     0     0
     2     0     2     0     1     0
     0     0     0     0     1     0
     0     0     0     0     0     0
     0     0     1     1     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
     2     0     1     1     0     0
     0     0     1     1     0     0
```

Figure 9: System Performance Summary: Accuracy and Detected Values

These results demonstrate that the system performs robustly under controlled conditions, with most detected coin values matching the correct values. Improvements could be made for edge cases where lighting, overlapping coins, or worn coins are in impact detection.

Conclusion

The MATLAB-based coin detection system demonstrates robust performance in the detection and counting of coins under controlled conditions. Image calibration and illumination correction significantly enhance detection accuracy, while geometric calibration ensures precise size estimation. The system achieved an overall accuracy of 80.56%, as shown in the evaluation results. Future work could focus on improving detection in challenging lighting conditions and extending the system to recognize denominations of coins.

References

1. MATLAB Documentation for Image Processing Toolbox.

2. Euro Coins Specifications: https://economy-finance.ec.europa.eu/euro/euro-coins-and-notes/euro-coins/common-sides-euro-coins$_en$