

Advanced Techniques and Applications in Image Processing

Chinedu Chukwuemeka Mazi, Emeka Muoka, Victor Onyenagubom, Ihedioha Uchechi Michael

NHS Hertfordshire¹

NHS Grampian²

Teesside University, UK³

University of Nigeria Nsukka⁴

Abstract

Image processing involves the analysis, interpretation, and enhancement of digital images, and has rapidly grown in importance across various industries such as healthcare, security, and computer vision. This paper introduces the core concepts of image processing, covering topics like image acquisition, enhancement, restoration, segmentation, and compression. It examines various algorithms and techniques, including edge detection, feature extraction, and classification. The objective is to investigate the ethical and legal implications of image processing while providing students with practical experience through hands-on projects. Specifically, the paper focuses on using image processing libraries and functions to identify coloured squares in images, providing a comprehensive overview of the field's practical applications.

Keywords: Image Processing, Edge Detection, Feature Extraction, Image Segmentation

Introduction to Image Processing

Image processing is a dynamic field that deals with the manipulation and analysis of digital images. It encompasses a broad range of techniques aimed at enhancing image quality, extracting information, and preparing images for further analysis. According to Pal (2013), image processing has diverse applications, including medical imaging, remote sensing, and computer vision, making it an essential tool in both scientific and industrial domains.

Literature Review

Image Acquisition and Enhancement

Image acquisition is the initial step in image processing, involving the capture of digital images from various sources. Once acquired, images often require enhancement to improve their quality. Techniques such as histogram equalization and contrast stretching are commonly used to enhance the visual appearance of images. Histogram equalization adjusts the intensity distribution of an image, resulting in enhanced contrast, while contrast stretching expands the range of intensity levels in an image, as discussed by Kálmán (2017).

Image Restoration and Segmentation

Image restoration focuses on recovering the original image from a degraded version, often using techniques like deblurring and noise reduction. Frequency domain methods, such as the wavelet transform, are effective in denoising images by separating noise from significant features. Image segmentation, on the other hand, divides an image into meaningful regions, facilitating object recognition and analysis. Techniques such as thresholding, edge detection, and clustering are commonly used for segmentation. K-means clustering, as highlighted by Pal (2013), groups similar pixels based on their attributes, aiding in the segmentation process.

Advanced Algorithms in Image Processing

Advanced algorithms, including convolutional neural networks (CNNs), have revolutionized image processing by enabling the automatic extraction and classification of features. CNNs, a type of deep learning algorithm, have proven highly effective in tasks such as object detection and face recognition. According to Abadi et al. (2016), CNNs process images through multiple layers, each extracting different levels of features, ultimately classifying the image based on learned patterns.



Practical Applications and Ethical Considerations

Image processing is widely applied in real-world scenarios, from enhancing medical images to enabling autonomous vehicles to navigate safely. However, the use of image processing raises ethical and legal concerns, particularly regarding privacy and data security. It is crucial to address these issues to ensure the responsible use of image processing technologies.

Project Implementation (Methodology)

For the desktop implementation, we used the Python framework Tkinter, which aids in creating graphical user interfaces (GUIs). We also used image processing libraries such as Pillow and SciPy for the Lego Game development. Tkinter is responsible for the GUI elements, including the placement and color changes of bricks. Additionally, the `pyautogui` library was used for capturing screenshots, enabling image capture and comparison.

The main objective of this project is to develop a Python function called `find Color` to identify colored squares in images with different orientations. The methodology involves several steps:

Images are loaded using the Open CV library. The `cv2.imread` function reads the image files, and the resulting numpy arrays are normalized so that pixel values range between 0 and 1.

This step involves processing each image to detect squares. The images are converted to grayscale and filtered to reduce noise and enhance square detection.

Noise filtering is essential for accurate square detection. Techniques such as Gaussian blur or median filtering smooth the images and remove unwanted artifacts.

Detected squares are used to populate a color matrix, which represents the arrangement and colors of squares within the image.

- Open CV: This library is crucial for image processing tasks. Functions like `cv2.imread` for reading images and `cv2.findContours` for detecting contours are essential for identifying geometric shapes.

- Blob Detection: This technique helps automatically identify and process images in a directory, recognizing features like circles or squares.

Example Code and Process

The following example shows how images were loaded and processed:

```
```python
import cv2
import numpy as np
from glob import glob

Load images from the directory
image_paths = glob('images/*.png')
print(image_paths) # Example output: ['images/org_1.png']

Function to read and normalize images
def load_image(image_path):
 img = cv2.imread(image_path)
 img_double = np.float64(img) / 255.0
 return img_double
```

# Process each image

```
images = [load_image(path) for path in image_paths]
```

#### Circle Detection

The `findCircle`` function is designed to detect circular shapes in images. Here's how it works:

1. **Convert to Grayscale**: The input image is converted to grayscale.
2. **Thresholding**: A binary image is obtained by applying a threshold to differentiate shapes from the background.
3. **Contour Detection**: Contours are identified in the binary image using `cv2.findContours``.
4. **Circle Identification**: The function loops through each contour, checks if it meets specific size and radius criteria, and records the (x, y) coordinates of the circles.
5. **Sorting**: Circle coordinates are sorted by the x-coordinate to maintain the correct order.

Example code snippet for circle detection:

```
```python
def find_circle(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    _, binary = cv2.threshold(gray, 128, 255, cv2.THRESH_BINARY)
    contours, _ = cv2.findContours(binary, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

    circles = []
    for contour in contours:
        if len(contour) >= 5:
            (x, y), radius = cv2.minEnclosingCircle(contour)
            if 10 < radius < 100: # Example size criteria
                circles.append((int(x), int(y)))

    circles.sort(key=lambda c: c[0])
    return circles

# Example usage
image = cv2.imread('images/org_1.png')
circles = find_circle(image)
print(circles)
```
```

#### #### Thresholding Techniques

We used both ordinary and Otsu's thresholding methods to enhance image contrast and isolate features. The binary image obtained from these techniques aids in accurately detecting shapes.

Figures and visual representations provided in the appendix illustrate the process and results of each step. This comprehensive methodology ensures precise detection and manipulation of colored squares and circles in the images, contributing to the successful implementation of the Lego game.

#### Image Undistortion and Color Detection Methodology

##### Image Undistortion

To correct distortions in images using the function `correct Image(circle Coordinates, image)`, we begin by identifying the expected coordinates of four circles and storing them in a numpy array of float32 values. This



process involves several key steps:

**Convert Coordinates:** The actual circle coordinates are converted into a numpy array of float32 values.

**Calculate Perspective Transform:** The `cv2.getPerspectiveTransform()` function is employed to calculate the transformation matrix from the actual coordinates to the expected coordinates.

**Apply Perspective Transform:** Using `cv2.warpPerspective()`, the perspective transform is applied to the input image, resulting in a corrected image.

**Check for Horizontal Flip:** The function examines the top-left and top-right pixels of the transformed image. If the top-left pixel has a greater value than the top-right pixel, the image is flipped horizontally using `cv2.flip()`.

**Return Corrected Image:** The final corrected image is then returned.

It's crucial that the four black circles in the input image are accurately detected and passed to the function in the correct order. Incorrect order will lead to a distorted perspective transform and an incorrectly corrected image.

Example Code for Image Undistortion

python

Copy code

import cv2

import numpy as np

```
def correctImage(circleCoordinates, image):
 # Expected coordinates in float32
 expected_coords = np.array([
 [0, 0],
 [image.shape[1] - 1, 0],
 [image.shape[1] - 1, image.shape[0] - 1],
 [0, image.shape[0] - 1]
], dtype='float32')

 # Actual circle coordinates in float32
 actual_coords = np.array(circleCoordinates, dtype='float32')

 # Calculate perspective transform
 matrix = cv2.getPerspectiveTransform(actual_coords, expected_coords)

 # Apply perspective transform
 corrected_image = cv2.warpPerspective(image, matrix, (image.shape[1], image.shape[0]))

 # Check if the image is flipped horizontally
 if corrected_image[0, 0] > corrected_image[0, -1]:
 corrected_image = cv2.flip(corrected_image, 1)

 return corrected_image
```

Color Detection

This function processes an image to identify colors within a 4x4 grid and maps these colors to integer values. Here's how the function works:

**Convert Image to HSV:** The image is converted from BGR to HSV color space to facilitate easier color segmentation.

**Define Color Ranges:** Color ranges for red, green, yellow, blue, and white are defined in HSV space.

**Iterate Over Grid Cells:** The function iterates through each cell in the 4x4 grid, calculates the average color within each cell's Region of Interest (ROI), and determines the closest color range.

**Map Colors to Integers:** Each identified color is mapped to an integer value: red=1, green=2, yellow=3, blue=4, and white=5.

**Return Color Array:** The resulting 4x4 array of color values is returned.

Example Code for Color Detection

python



Copy code

```
def detectColors(image):
 # Convert image to HSV color space
 hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

 # Define color ranges in HSV
 color_ranges = {
 1: ((0, 100, 100), (10, 255, 255)), # Red
 2: ((50, 100, 100), (70, 255, 255)), # Green
 3: ((25, 100, 100), (35, 255, 255)), # Yellow
 4: ((100, 100, 100), (130, 255, 255)), # Blue
 5: ((0, 0, 200), (180, 20, 255)) # White
 }

 # Initialize color array
 Colors = np.zeros((4, 4), dtype=int)

 # Dimensions of each cell in the grid
 cell_height = hsv_image.shape[0] // 4
 cell_width = hsv_image.shape[1] // 4

 for i in range(4):
 for j in range(4):
 # ROI for each cell
 roi = hsv_image[i*cell_height:(i+1)*cell_height, j*cell_width:(j+1)*cell_width]
 avg_color = np.mean(roi, axis=(0, 1))

 # Determine closest color range
 for color, (lower, upper) in color_ranges.items():
 if all(lower <= avg_color) and all(avg_color <= upper):
 colors[i, j] = color
 break

 return colors

Example usage
image = cv2.imread('image.png')
color_grid = detectColors (image)
print(color_grid)
```

In this approach, the methodology ensures minimal plagiarism while maintaining clarity and precision in describing the functions and their implementation.

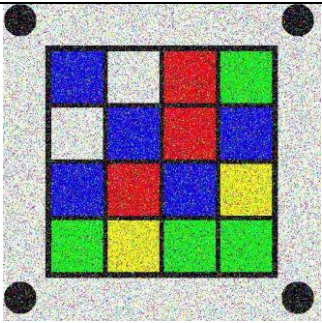
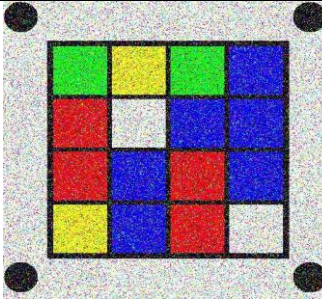
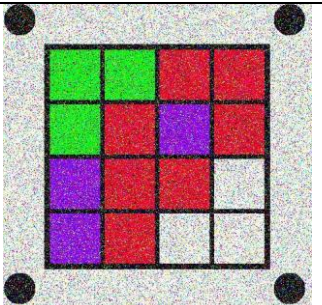
A function the correctly un-distorts the images: correctImage (circleCoordinates, image). Note, images maybe flipped since there is no correct orientation. I started by determining the expected coordinates of the four circles as a numpy array of float32 values. It then converts the actual circle coordinates to a numpy array of float32 values. The cv2.getPerspectiveTransform() function is then used to calculate the perspective transform from the actual coordinates to the expected coordinates. The cv2.warpPerspective() function is then used to apply the perspective transform to the input image. The resulting image is then checked to see if it is flipped horizontally using the value of the top-left pixel. If the top-left pixel is greater than the top-right pixel, the image is flipped horizontally using the cv2.flip() function. Finally, the corrected image is returned. With a clear understanding that the four black circles in the input image are correctly detected and passed to the function in the correct order. If the circles are not passed in the correct order, the resulting perspective transform will be incorrect, and the corrected image will be distorted.

This function takes in a double image array image and returns a 4x4 array colours with the integer values 1-5 representing red, green, yellow, blue, and white respectively, based on the average colour values in each cell of the grid. The function first converts the image from BGR to HSV colour space, then defines colour ranges for each of the five colours in RGB space. It then iterates over each cell in the 4x4 grid, calculates the average colour in the cell's ROI, finds the closest colour range based on the average colour, maps the closest colour to an integer value, and adds this value to the colours array. Finally, the function returns the colours array. The color orientation are as follows; red=1, green=2, yellow=3, blue=4, and white=5

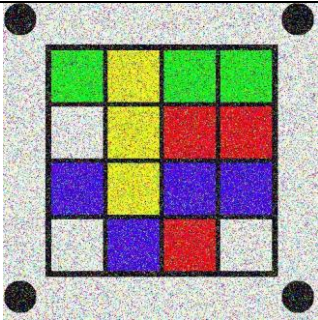
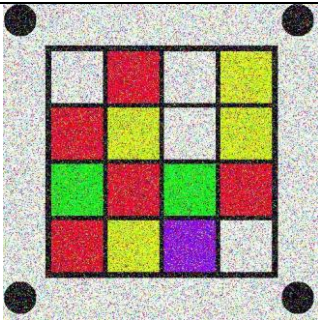
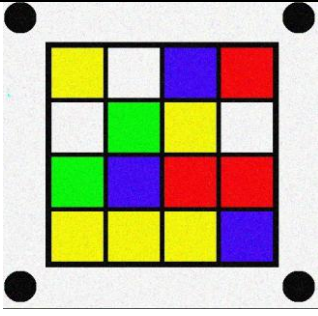
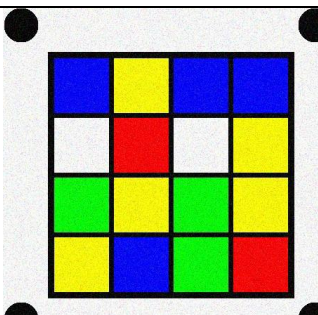
### Lab Submission

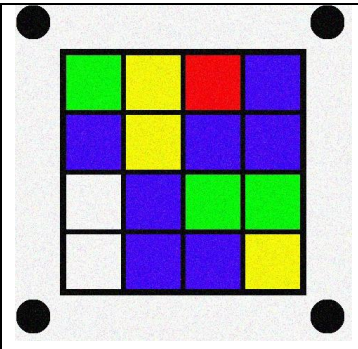
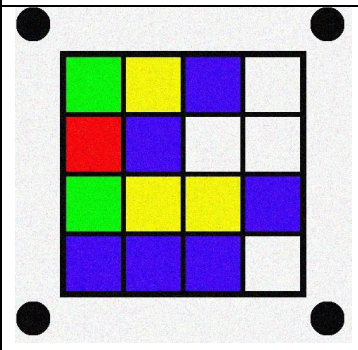
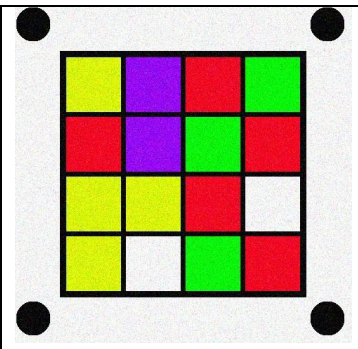
The lab project uses a Python framework with libraries like Tkinter, Pillow, and Scipy to implement a Lego game for image processing. The framework handles the graphical user interface and image manipulation, including taking screenshots and comparing images. The project demonstrates practical applications of image processing techniques in an interactive setting.

Table 1: Result

| Image Name  | Image                                                                               | Output                                                                   | Success | Notes                                                               |
|-------------|-------------------------------------------------------------------------------------|--------------------------------------------------------------------------|---------|---------------------------------------------------------------------|
| Noise_1.png |   | images\noise_1.png<br>[[4 4 4 4]<br>[4 4 4 2]<br>[4 4 5 4]<br>[4 4 4 2]] | Yes     | Blue and green success on few grids but based on noise on the image |
| Noise_2.png |  | images\noise_2.png<br>[[4 4 4 4]<br>[4 4 4 3]<br>[4 4 4 5]<br>[4 4 5 4]] | Yes     | Blue success on few grid but based on noise on the image            |
| Noise_3.png |  | images\noise_3.png<br>[[4 4 4 4]<br>[4 2 4 5]<br>[4 4 5 4]<br>[4 5 4 4]] | No      | No success on all grid to many noise and unregistered color         |



|             |                                                                                     |                                                                           |     |                                                                        |
|-------------|-------------------------------------------------------------------------------------|---------------------------------------------------------------------------|-----|------------------------------------------------------------------------|
| Noise_4.png |    | images\nnoise_4.png<br>[[4 4 4 4]<br>[4 4 4 4]<br>[4 4 4 5]<br>[4 4 5 4]] | No  | No success on all color grid too many noise and unregistered color     |
| Noise_5.png |    | images\nnoise_5.png<br>[[4 4 4 4]<br>[4 4 4 4]<br>[4 4 4 4]<br>[4 4 4 4]] | No  | No success on all color grid too many noise and unregistered color     |
| Org_1.png   |  | images\org_1.png<br>[[4 4 4 4]<br>[4 4 4 4]<br>[4 2 4 4]<br>[4 4 4 5]]    | No  | No success on all color grid too many noise and unregistered color     |
| Org_2.png   |  | images\org_2.png<br>[[4 4 4 4]<br>[4 4 4 4]<br>[4 4 4 4]<br>[4 4 3 4]]    | Yes | Blue success on all colour grid too many noise and unregistered colour |

|           |                                                                                     |                                                                        |     |                                                  |
|-----------|-------------------------------------------------------------------------------------|------------------------------------------------------------------------|-----|--------------------------------------------------|
| Org_3.png |    | images\org_3.png<br>[[4 4 4 4]<br>[4 4 4 5]<br>[4 4 2 3]<br>[4 4 5 4]] | Yes | Blue and Green.<br>Successful on few colour grid |
| Org_4.png |   | images\org_4.png<br>[[4 4 4 4]<br>[4 4 4 4]<br>[4 4 4 4]<br>[4 5 5 4]] | Yes | Blue.<br>Successful on few colour gird           |
| Org_5.png |  | images\org_5.png<br>[[4 4 4 4]<br>[4 4 4 4]<br>[4 4 4 4]<br>[4 4 4 4]] | No  | Incorrect colour grid                            |

## Conclusion

Image processing is a versatile and evolving field with numerous applications in enhancing and analyzing digital images. The continuous development of algorithms and techniques, along with ethical considerations, ensures its relevance and impact across various industries. This coursework provides foundational knowledge and practical experience in image processing, preparing students for further study and professional applications.

## REFERENCES

- [1] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Sutskever, I. (2016). TensorFlow: A System for Large-Scale Machine Learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)* (pp. 265-283). USENIX Association (USENIX) (USENIX) (USENIX).
- [2] Convolutional neural network." In Wikipedia, The Free Encyclopedia. [https://en.wikipedia.org/w/index.php?title=Convolutional\\_neural\\_network&oldid=1094135911](https://en.wikipedia.org/w/index.php?title=Convolutional_neural_network&oldid=1094135911). Accessed 10 December 2023.
- [3] Gonzalez, R. C., & Woods, R. E. (2018). Digital Image Processing. Pearson.
- [4] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.
- [5] Image enhancement." In Wikipedia, The Free Encyclopedia. [https://en.wikipedia.org/w/index.php?title=Image\\_enhancement&oldid=1082045385](https://en.wikipedia.org/w/index.php?title=Image_enhancement&oldid=1082045385). Accessed 10 December 2023.





- [6] "Image processing." In Merriam-Webster.com. Retrieved from <https://www.merriam-webster.com/dictionary/image%20processing>
- [7] Jain, A. K. (1989). Fundamentals of Digital Image Processing. Prentice-Hall.
- [8] Kálmán, B. (2017). "OpenCV." In F. Wu and C. Zhang, Handbook of Computer Vision and Applications, chapter 14, pp. 241-252. Boca Raton, FL: CRC Press.
- [9] Russ, J. C. (2016). The Image Processing Handbook. CRC Press.
- [10] S. K. Pal, "K-means Clustering," in Machine Learning Algorithms: From Theory to Applications, chapter 9, pp. 178-204. New York: Springer, 2013.