CMPS446 – Fall 2024

(Image Processing & Computer Vision)

FawryVision - Team 23

Presented To:

Dr. Youssef Ghattas

Prepared By:

Ahmed Tarek AbdelAal	1200088
Basel Hossam Abdelmoula	1200462
Heba Gamal	1200076

Table of Contents

Table of Contents 2
1. Project Overview 3
2. Implementation details 3
2.1. Receipt Extraction Step 4
2.1.1 Edge based segmentation 4
2.1.2 Color based segmentation 6
2.2 Cropping the 16 digits Number 9
1. Performance & accuracy 10
2. Measuring accuracy 11
3. How to operate 12
First method : 12
Second method : 12
4. References 13

1. Project Overview

FawryVision is an image processing system designed to help users quickly and accurately extract a Fawry 16-digit code from receipts, Balance Recharge Code typically used to recharge phone credit. This system aims to reduce the challenges associated with manually reading and entering these codes, which can be difficult for individuals with vision impairments or prone to human error. By leveraging image processing techniques, FawryVision automatically identifies, isolates, and displays the code for easy input, minimizing the potential for misreading digits due to poor print quality, low lighting, or user oversight.

NOTE: This project emphasizes classical image processing techniques to maximize efficiency and avoid reliance on deep learning.

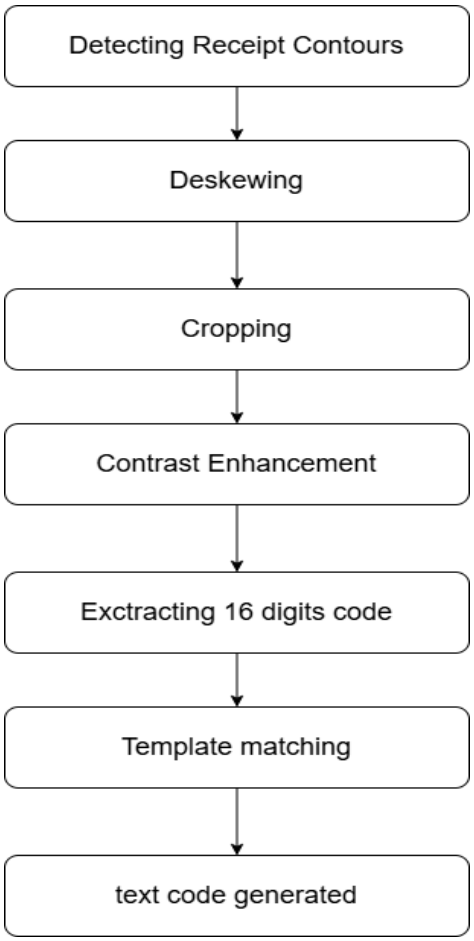
2. Implementation details

We explored two distinct pipelines for segmenting receipts from noisy background. While both pipelines follow

the same overall structure, they differ in the initial segmentation steps.

The first approach employs edge-based segmentation, whereas the second relies on color-based segmentation. Initially, we implemented the edge-based segmentation method; however, it encountered limitations and failed in certain test cases. Consequently, we pursued an alternative approach using color-based segmentation.

In the following sections, each pipeline will be described in detail. Ultimately, we integrated both approaches into a unified pipeline, which demonstrated improved performance by successfully handling a greater number of test cases.



2.1. Receipt Extraction Step

2.1.1 Edge based segmentation

Convert the image to grayscale and apply contrast enhancement using Gamma correction.

a) Apply Canny edge detector with high sigma value to reduce the edges detected from the background noise
b) Dilate the edges to connect the receipt edges if it's disconnected, so that *find_countours* can detect the receipt contour.
c) Apply Thinning to the dilated image before the corner detection step because thick edges resulting from dilation produce false positive corners. The dilation Step's only usage was connecting the receipts edge if disconnected. After it's connected we can return it back thin .
d) Find The largest Contour area
e) Use Approximate Polygon to get exactly 4 corners. If more than 4 corners are detected then try more iterations with tuning parameters to get exactly 4. However, the default parameters are experimented to handle most of the different test cases , Iterations is done for extremely noisy images
f) Using the four corners , apply Perspective Transformation that fixes the perspective view of the receipt as well as fixing the slight skews in the card .

Figure a

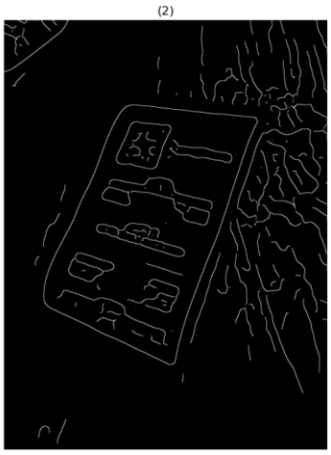


Figure b

Figure c

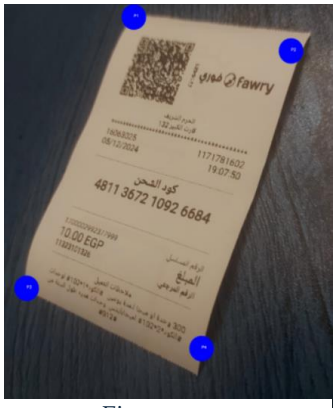


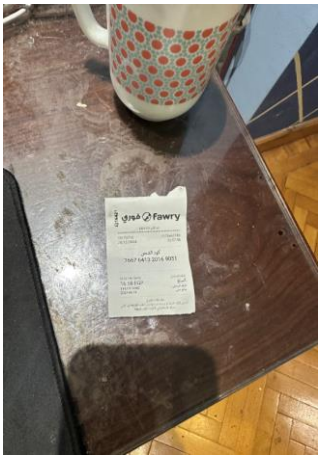
Figure a

Figure f

Problems with this approach:

Depending only on the edge detection to find the receipt contours was experimented to be greatly affected by noise , for example if the background contains other objects or lightning , the sigma in canny's edge detector wouldn't be sufficient to remove the noise .

For the following image , notice the detected edges after the canny's edge detector in image (2) . The receipt's contour is not connected which will never be found by finding contours , and applying dilation here to connect the edges resulted in a more noisy image (3) which we cannot process to the next step to find the card contour.



(1)

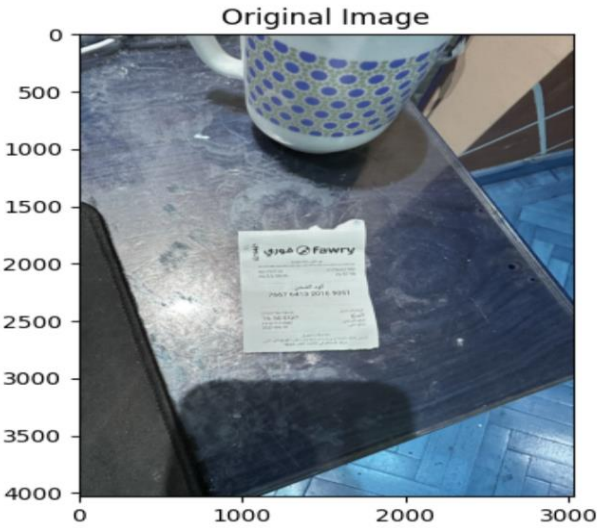
(2)

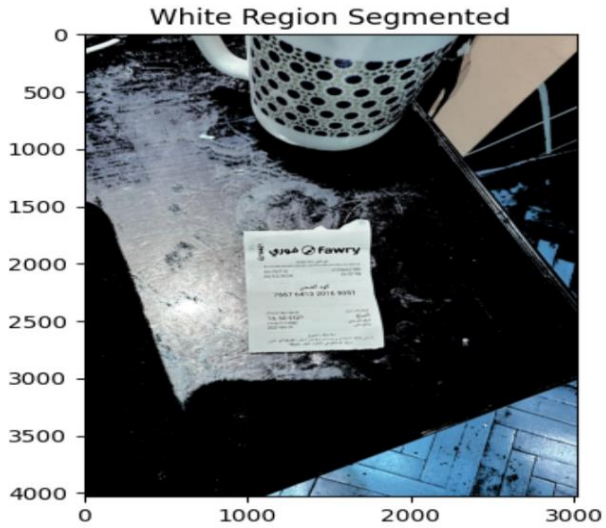
(3)

2.1.2 Color based segmentation

we used the fact that the receipt is a white area and used k-means for segmentation, the one used in the first approach, before which decreased the percentage of error for noisy images and the contours are much better.

a. As a starting point, K means is performed on the whole RGB image, number of clusters used is 2, this ensures that the brighter colors will be grouped together (including the receipt), and the darker parts will be grouped into another cluster, The result is something like this one





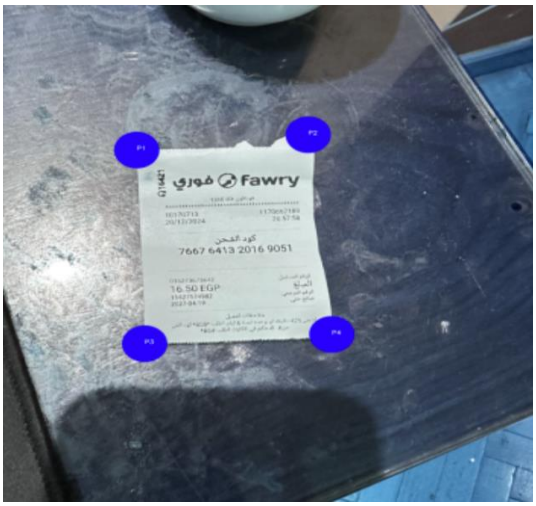
b. One of the main problems of this approach is the white noise present on the surface, unfortunately since it shares the same color as the receipt, it gets extracted as well from K-means, now in order to reduce it , we need to perform some morphological operations, so as a result, image is first converted to greyscale and here appears one of the biggest challenges, what threshold should be used ? For images with shadows, a low threshold (e.g., using the minimum_threshold function) helps capture as much brightness as possible without being influenced by shadows. Conversely, for images with significant white noise, a high threshold is necessary to eliminate noise during binary conversion. Through experimentation with various thresholds, including Otsu and minimum_threshold, it became evident that thresholds are case-specific. After testing multiple values (e.g., 0.4, 0.5, 0.6), a threshold of **0.6** was found to generalize best across different scenarios.

c. The next step involved applying morphological operations to improve the results. After experimenting with different combinations, sizes, and iterations of binary opening, closing, and erosion, the most effective approach was to apply binary opening once with a 2x2 window and 15 iterations. This method effectively reduced white noise around the receipt while preserving the integrity of the image.

d. Once the white region of the receipt is separated from the rest of the image, the next step is to identify the largest contour surrounding the receipt. This is achieved using the `find_contours` function. Based on experimentation, selecting the contour with the maximum area consistently yielded better results compared to selecting the longest contour, even in subsequent stages.

e. After obtaining the contour, ideally a rectangular one, the next step is to approximate it to a rectangle. This allows us to extract the four corners needed for a perspective transformation, providing a bird's-eye view of the receipt. This was achieved using cv2.approxPolyDP by iteratively reducing the epsilon parameter passed to the function until the approximated polygon had exactly four points, indicating a rectangle.
It is worth noting that, sometimes this polygon cannot be obtained, if the contour is very noisy and not taking any rectangular shape, this occurs in cases of too much white noise

f. Once the four corners of the polygon are obtained, a final adjustment may be needed. Occasionally, three corners align well with the receipt, while the fourth deviates significantly, requiring outlier detection and correction. Outlier detection is performed by calculating the mean coordinates (x,y)(x, y)(x,y) of all four points. The outlier is identified as the point furthest from the mean. To correct it, the unit vector from the outlier to the mean is calculated, and the outlier is adjusted in this direction by a factor equal to the difference between the outlier's distance and the average distance of the other points. An alternative approach could involve predicting the outlier's position based on geometric relationships (e.g., estimating the top-left point using the bottom-left and top-right points). However, the implemented method has proven effective and did not require further modification.



g. After identifying the correct four points, they can be used for the perspective transformation step. This process corrects any slight skew in the image, ensuring the receipt is properly aligned.

Problems with this approach :

- Hard to determine thresholds that generalize well in different lighting conditions.
- It is the most prone to white noise, as receipt and noise have same color, making receipt detection harder.
- Determining the correct combination of morphological operations to remove white noise.
- Obtaining the correct rectangle around the receipt.
- Choosing the right dimensions to crop on, before detecting the code or price
- Cropping the digits without getting a contour on 2 digits at the same time

2.2 Cropping the 16 digits Number

a) Apply Erosion and Dilation to the transformed image to get the 16 as digits a single contour
b) get all contours and filter by the aspect ratio in addition to the contour width to get only the correct contour
c) Crop the image and return the 16 digits as one single image
d) Find Contours again on the resulted image to get each digit image separately for the template matching module

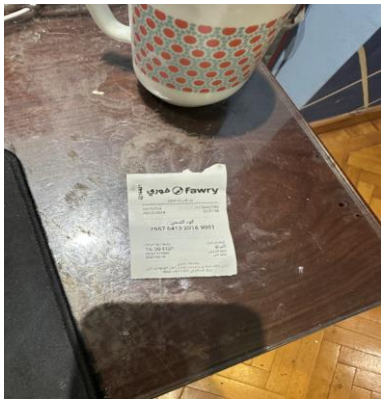
3. Performance & accuracy

a. The selected approaches effectively handle standard test cases without noise or minimal skewing. However, they encounter challenges with severely skewed images or those affected by intense flash lighting.
b. The segmentation techniques accurately extract the 16-digit code but occasionally struggle with the price, as the transformed image sometimes crops part of the price, preventing successful template matching.
c. These corner cases were intentionally introduced to evaluate the robustness and accuracy of our methods under extreme conditions. In practical applications, users aiming to extract numbers from a card are unlikely to capture photos that are heavily rotated or taken from extreme perspectives.

Here's an example of our two failing test cases :

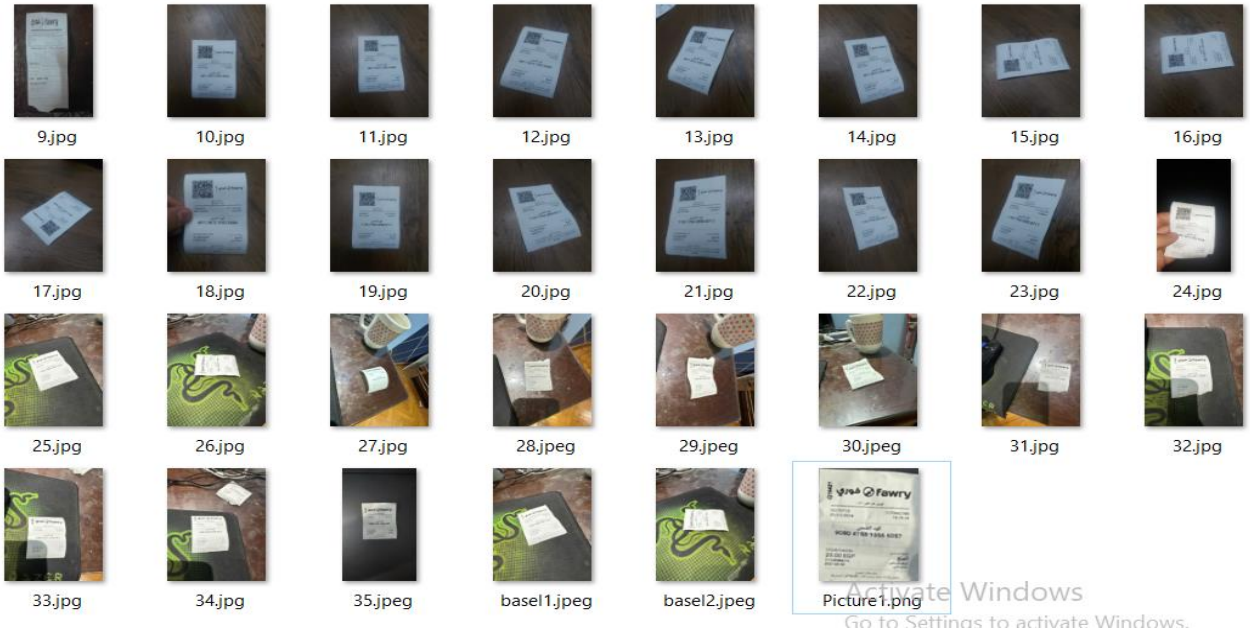


However , our code successfully passes these test cases :


4. Measuring accuracy

Using a dataset of 30 receipts captured from various perspectives, we tested our project against PyTesseract's pretrained OCR model and evaluated its performance by calculating recall, accuracy, and precision.



Results:

- Total Processed Images: 15
- Correct Predictions: 12
- Accuracy: 80%

	Our Implementation	Pytesseract
Accuracy	80 %	90%

5. How to operate

First method :

1) You can use the GUI by running this command

   streamlit run .\code\gui\app.py in the root project directory.

   (detailed packages needed to run is provided in the readme file)

2) Drag your receipt image
3) wait for a couple of seconds until the image is processed and you will get the 16 digits code and the price . (the price is a plus , it's not always gets extracted in noisy images)

Second method :

Open Fawry.ipynb notebook and add the path to the image you want to test in the testing cell provided .

6. References

[1]. pyimagesearch.com: credit-card-ocr-with-opencv-and-python

[2]. Template Matching Advances and Applications in Image Analysis: https://arxiv.org/abs/1610.07231

[3]. An Overview of the Tesseract OCR Engine: 33418.pdf

[4]. https://stackoverflow.com/questions/7263621/how-to-find-corners-on-a-image-using-opencv

[5]. https://stackoverflow.com/questions/64860785/opencv-using-canny-and-shi-tomasi-to-detect-round-corners-of-a-playing-card