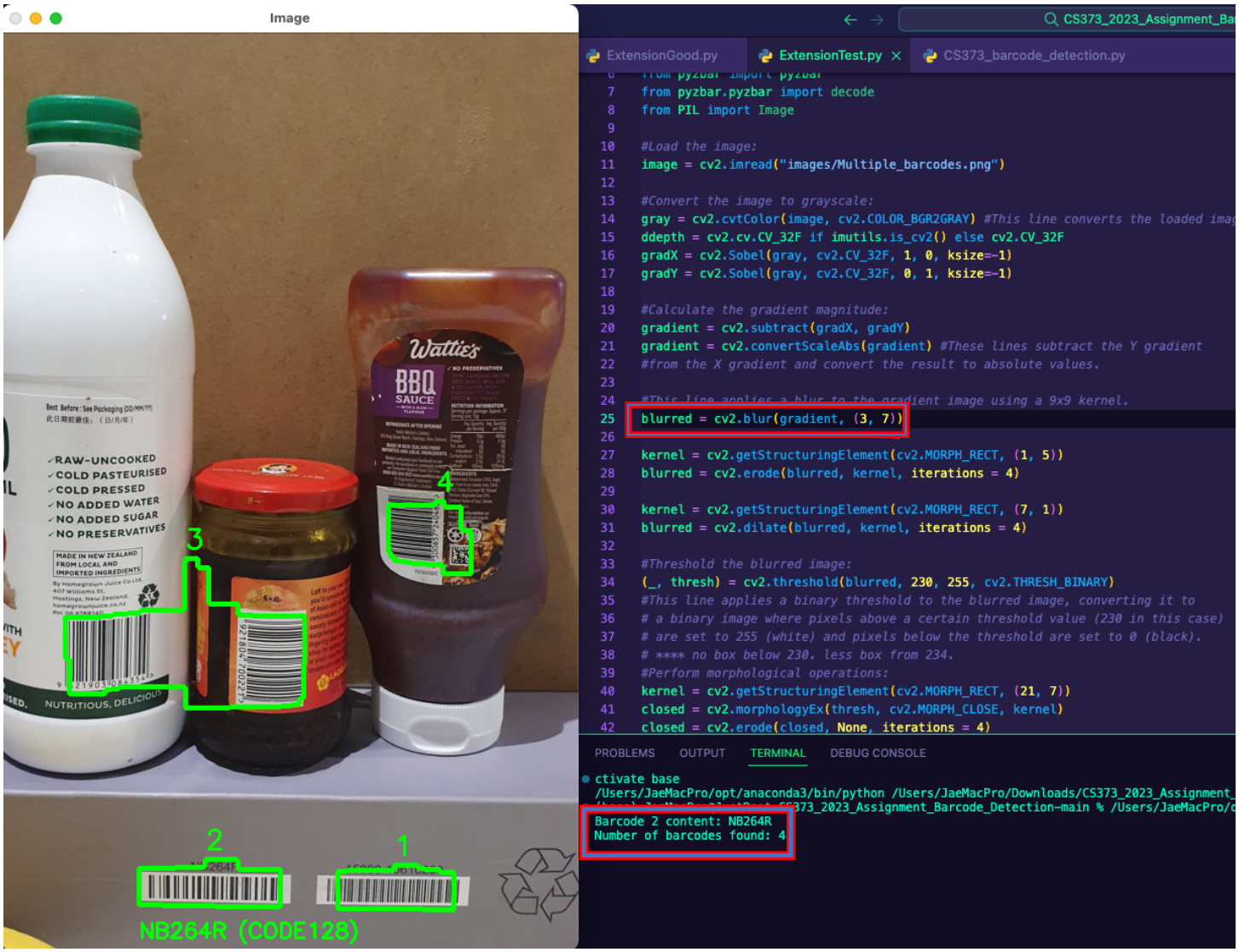
Jae Kim #466217270
Extension of Barcode Detector

The purpose of this extension is to enhance the barcode detection and decoding functionality in an image. The extension builds upon the existing codebase, utilizing computer vision techniques, OpenCV, and the pyzbar library to improve the accuracy of barcode detection and content extraction. This report presents an overview of the extended code, including the modifications made, the experimental results, and the limitations encountered.

Multiple_barcodes.png (blurring image with 3 x 7 kernel)



Libraries used:

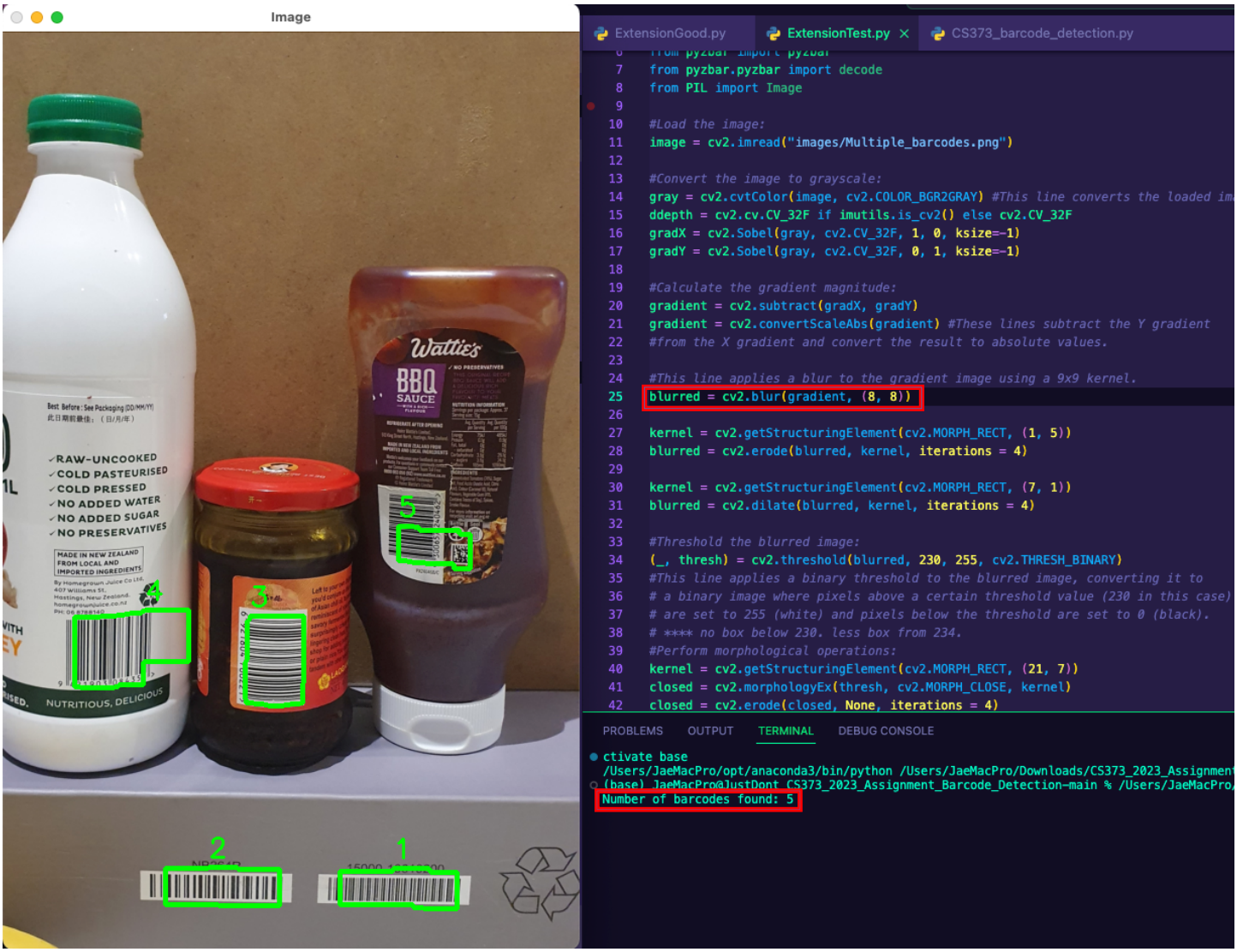
```
import numpy as np
import imutils
import cv2
from pyzbar import pyzbar
from pyzbar.pyzbar import decode
from PIL import Image
```

Once the Multiple_barcodes.png is loaded, the image is then converted to grayscale, and the gradient magnitude is calculated using the Sobel operator. To enhance barcode detection, the gradient image is blurred using different kernel sizes. Morphological operations are applied to further refine the barcode regions.

Contours are found in the binary image after thresholding and filtering. The code filters the contours based on their area and draws the filtered contours on the original image. For each detected barcode, the code puts a coloured index number on the rectangle and decodes the barcode content using the pyzbar library. Finally, the image with drawn contours, index numbers, and barcode content is displayed as well as the number of barcodes are printed out on the terminal.

Several modifications were made to improve the barcode detection and decoding accuracy. The size of the kernel used for blurring the gradient image was adjusted. By using an 8x8 kernel, the code successfully detected five separate barcodes. However, it failed to read the content of one barcode due to erosion causing loss of detail. On the other hand, using a 3x7 kernel allowed the code to successfully decode one barcode's content but resulted in four barcode detections instead of five. The selection of kernel size was a trade-off between the number of detected barcodes and the ability to read barcode content.

Multiple_barcodes.png (blurring image with 8x8 kernel)



Further modifications were made to the morphological operations. The size of the structuring element used for erosion and dilation was adjusted to separate well-separated barcodes. However, altering these parameters resulted in either no barcode detection or overlapping grouping of barcodes with text boxes in the image.

```
24  #This line applies a blur to the gradient image using a 9x9 kernel.
25  blurred = cv2.blur(gradient, (8, 2)) # (8,8) & (3,7), it only detects all barcodes but counts 4 with 1 barcode content.
26  blurred = cv2.blur(gradient, (3, 7)) #(3,7) by itself, it only detects all barcodes but counts 4 with 1 barcode content.
27
28  kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (1, 5))
29  blurred = cv2.erode(blurred, kernel, iterations = 4)
30
31  kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (7, 1))
32  blurred = cv2.dilate(blurred, kernel, iterations = 4)
33
34  #Threshold the blurred image:
35  (_, thresh) = cv2.threshold(blurred, 230, 255, cv2.THRESH_BINARY)
36  #This line applies a binary threshold to the blurred image, converting it to
37  # a binary image where pixels above a certain threshold value (230 in this case)
38  # are set to 255 (white) and pixels below the threshold are set to 0 (black).
39  # **** no box below 230. less box from 234.
40  #Perform morphological operations:
41  kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (21, 7))
42  closed = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, kernel)
43  closed = cv2.erode(closed, None, iterations = 4)
44  closed = cv2.dilate(closed, None, iterations = 4)
45
46  kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (9, 5))
47  closed = cv2.dilate(closed, kernel, iterations = 3)
48
49  kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (9, 1))
50  closed = cv2.erode(closed, kernel, iterations = 7)
51
52  kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (9, 5))
53  closed = cv2.dilate(closed, kernel, iterations = 4)
54
55  kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (9, 1))
56  closed = cv2.erode(closed, kernel, iterations = 3)
```

I updated Line 28 by changing the structuring element (SE) size from (1, 7) to (1, 5) since it effectively separated the barcodes and improved their detection.
For Line 31, I kept the SE size as (1, 7) because any modifications to it resulted in either no detection of barcodes or overlapping grouping of barcodes with boxes around text in the image. Thus, the original SE size was optimal for the desired barcode detection and separation.

Thresholding the blurred image was another critical step. Various threshold values were tested, and I found that a value around 230 or 231 provided the best results. Values below 230 or above 234 resulted in either no barcode detection or a reduced number of detected barcodes.

I made modifications to Line 46, 47, 49, and 50 by applying iterations of 3 and 7 before performing another set of dilation and erosion. This adjustment improved the overall performance; however, finding the optimal balance between separating objects and preserving barcode details remained a challenge. When separating the two objects through iterations of erosion and dilation, the barcode located in the bottom left corner loses its necessary details, making it impossible for pyzbar to decode its content. On the other hand, if I avoid this separation, the accuracy of the total number of detected barcodes is compromised.

Based on the limitations encountered in the barcode detection and decoding process, it may be more suitable to redefine the purpose of this program. Instead of trying to both count the number of barcodes and extract their content in a single image, it could be more effective to split the functionalities into separate programs. The revised approach would involve having one program solely dedicated to counting the number of barcodes present in an image. This program would utilize image processing techniques, contour analysis, and thresholding to accurately determine the count of barcodes.

On the other hand, a separate program would focus exclusively on reading the content of a single barcode. This program could utilize the pyzbar library or other barcode decoding methods to extract the information encoded within a barcode.

By separating these functionalities, we can better address the specific requirements and challenges associated with barcode detection and content extraction. This approach allows for more flexibility, optimisation, and adaptability in each program, ensuring accurate barcode counting and reliable barcode content retrieval.

Multiple_barcodes.png (blurring image with9x5, 3x7, 8x8 kernels)