COS 470 Image Processing and Computer Vision
2024 Fall
Taylor Brookes
10/2/2024
Assignment 2

Task 1: Object size measurements
Q1.1: Detection and Measurement -- Measure dimensions of the book (height and width in cm). Describe methodology and rationale of the approach.
A1.1:

Step 1: transform paper from trapezoid to rectangle and fill image with paper, then scale image down so it can fit on my screen

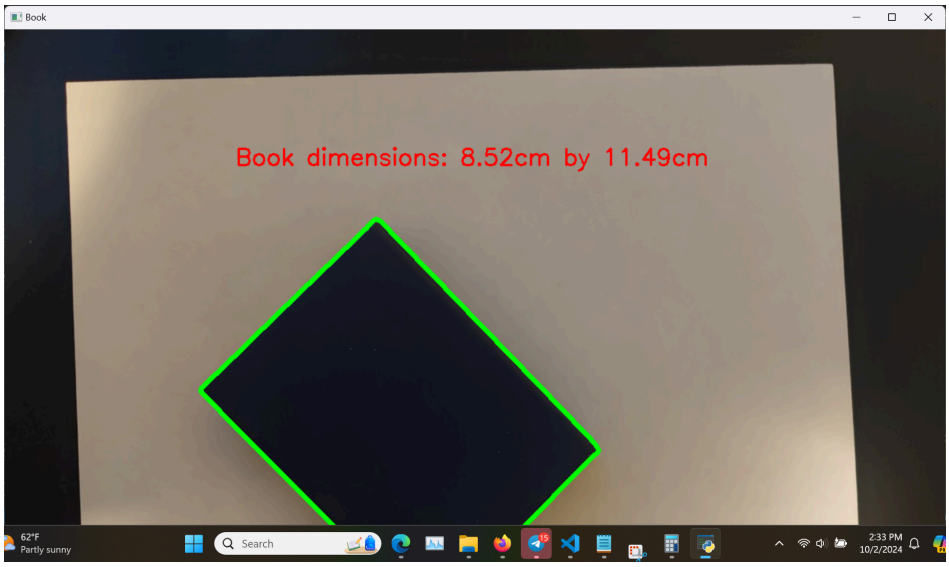
Step 2: locate the corners of the book in the new image

Step 3: measure from corner to corner of the book and average the results

Step 4: convert book pixel lengths to cm with scale factor

I used BirdView2 as a template to automatically locate the corners of the paper and book with contours. This calculated the corners very quickly, while my Harris Corner Detector implementation that I chose not to use was taking about 10 seconds each detection cycle. I probably could have gotten more accuracy with the Harris Corner Detector but didn't have enough time to program the second detection method. In addition, I believe that using the FAST corner detection method and using the value of the pixels in its detection circle to determine which corners are for the paper and which are for the book would be helpful. The book would have most of its corner's neighbours be white, while the paper would have most of its corner's neighbours be black.

Q1.2: Annotation -- Draw a rectangle around the book. Display the measured width and height on the image.
A1.2:

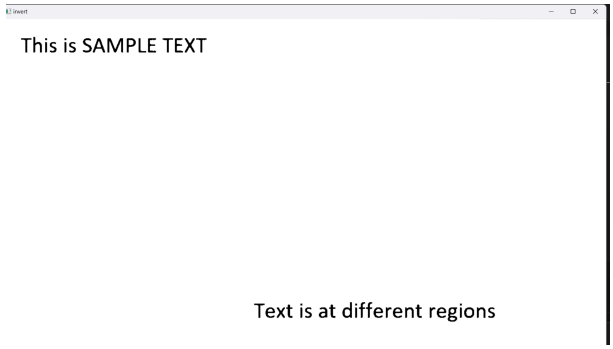


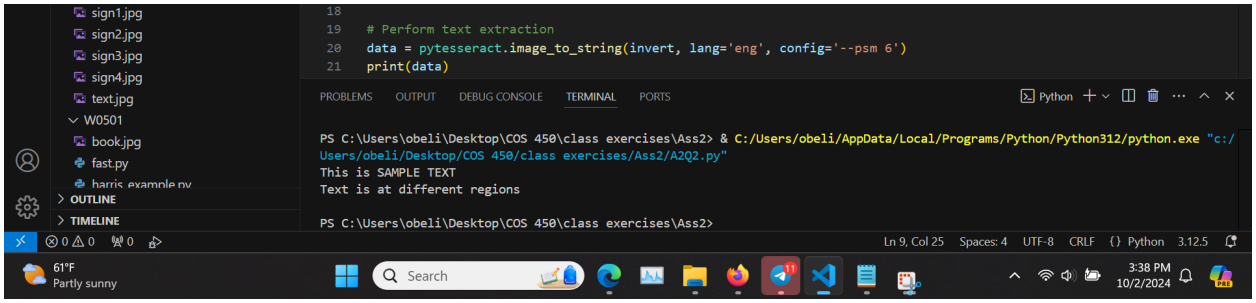
Q1.3: Comparison and Documentation -- Compare your measurements to the actual measurements (8cm x 10.6cm). Aim for <10% error.
A1.3:

My answer: 8.52cm x 11.49cm (6.5% and 8.4% error)

Because I used contours instead of actual corner detection, I believe this is why my answers are somewhat off but still within the margin of error. Changing the scale factor of the image for the contours does not significantly change the result of the calculation (around 0.02cm difference).

Task 2: Text recognition -- use OpenCV and Python-tesseract
Q2.1: Text Analysis -- Write code to recognize the text in the image “text.jpg”.
A2.1:





Q2.2: Sign Recognition -- Apply methods to the 4 signs to recognize the text on the signs. Document any additional methods to improve accuracy if detection fails. Explain strategies and summarize accuracy (= # correct recognition / # total letters). EX: “stop” vs “step” = 75% accuracy. HINT: use detection algorithms to find regions of interest (RoI) and apply PyTesseract to the regions. Image preprocessing can help such as thresholding or noise reduction. Can also tune parameters of specific functions.
A2.2:

I probably could have used contouring (RETR_EXTERNAL with a -1 fill or something) to create a less noisy binary image, and I also manually created the regions of interest for each sign instead of having the computer create the region of interest, so it’s not very transferable. I found the following code and implemented it to the signs to see its accuracy. I was not entirely sure what getStructuringElement and morphologyEx do, but after looking it up they appear that getStructuringElement creates a mask for morphologyEx, and morphologyEx uses erosion (sharpens the image, thickness/size of object is decreased) and dilation (opposite of erosion),

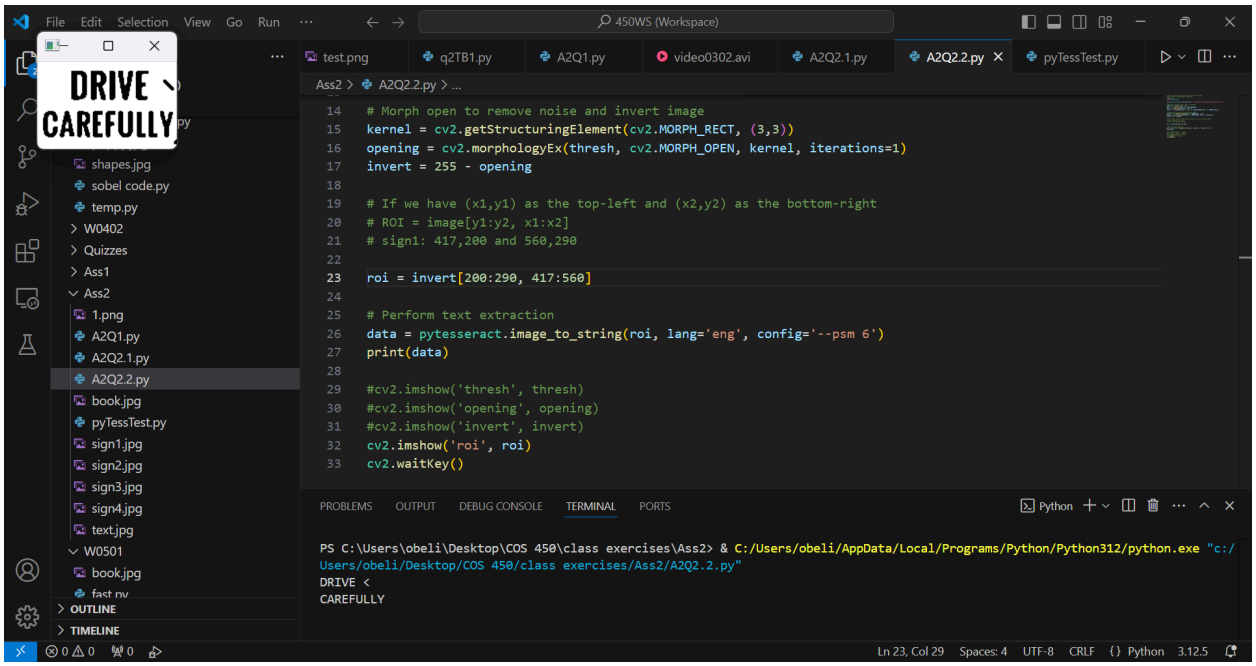
with the specific morph function being used as MORPH_OPEN, which first uses erosion and then uses dilation. (https://docs.opencv.org/3.4/d4/d76/tutorial_js_morphological_ops.html).

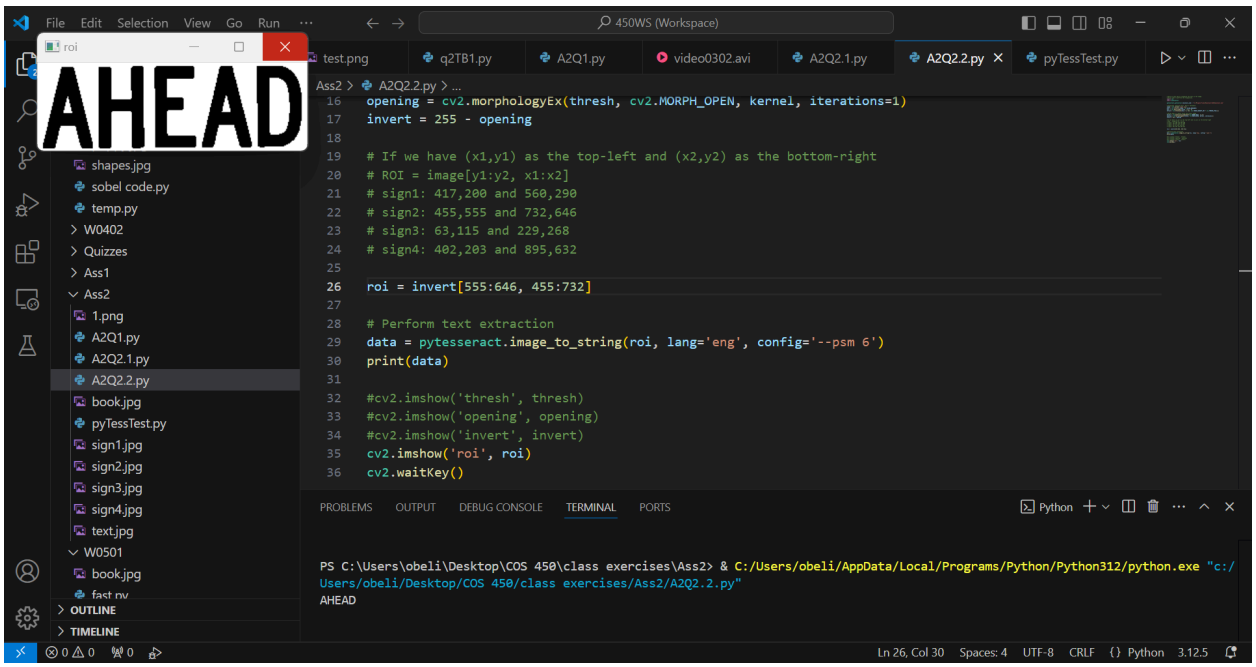
```
# Morph open to remove noise and invert image
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3,3))
opening = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel, iterations=1)
invert = 255 - opening
```

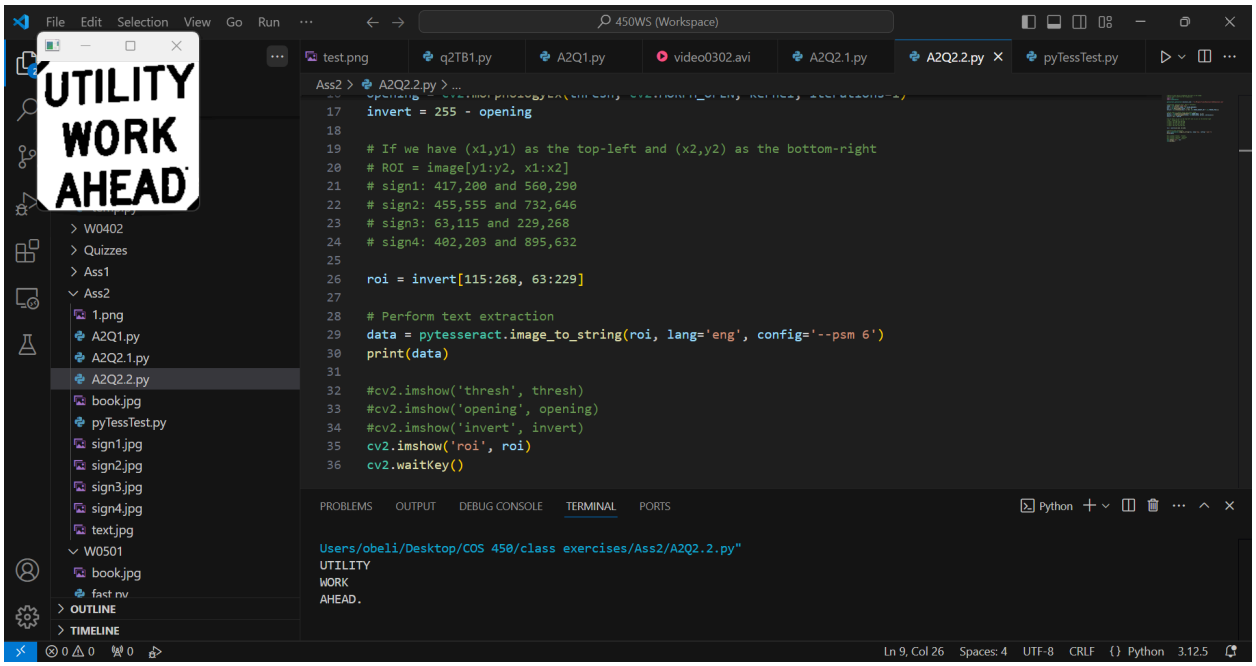
TABLE OF RESULTS:

accuracy	no morph no ROI	morph no ROI	no morph ROI	morph ROI
sign1	100%	82%	87%	93%
sign2	2%	0%	100%	100%
sign3	38%	50%	88%	94%
sign4	86%	86%	68%	72%

MORPH WITH MANUAL ROI
Sign1: accuracy = 93% (14/15)
DRIVE < CAREFULLY
vs
DRIVE CAREFULLY
Sign2: accuracy = 100% (5/5)
AHEAD
vs
AHEAD
Sign3: accuracy = 94% (16/17)
UTILITY WORK AHEAD.
vs
UTILITY WORK AHEAD
Sign4: accuray = 72% (13/18)
N0O*” PASSING ZONE gq
vs
NO PASSING ZONE


DRIVE
CAREFULLY
# Morph open to remove noise and invert image
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3,3))
opening = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel, iterations=1)
invert = 255 - opening
# If we have (x1,y1) as the top-left and (x2,y2) as the bottom-right
# ROI = image[y1:y2, x1:x2]
# sign1: 417,200 and 560,290
roi = invert[200:290, 417:560]
# Perform text extraction
data = pytesseract.image_to_string(roi, lang='eng', config='--psm 6')
print(data)
#cv2.imshow('thresh', thresh)
#cv2.imshow('opening', opening)
#cv2.imshow('invert', invert)
cv2.imshow('roi', roi)
cv2.waitKey()
PS C:\Users\obeli\Desktop\COS 450\class exercises\Ass2> & C:/Users/obeli/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/obeli/Desktop/COS 450/class exercises/Ass2/A2Q2.2.py"
DRIVE <
CAREFULLY


roi
AHEAD
opening = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel, iterations=1)
invert = 255 - opening
# If we have (x1,y1) as the top-left and (x2,y2) as the bottom-right
# ROI = image[y1:y2, x1:x2]
# sign1: 417,200 and 560,290
# sign2: 455,555 and 732,646
# sign3: 63,115 and 229,268
# sign4: 402,203 and 895,632
roi = invert[555:646, 455:732]
# Perform text extraction
data = pytesseract.image_to_string(roi, lang='eng', config='--psm 6')
print(data)
#cv2.imshow('thresh', thresh)
#cv2.imshow('opening', opening)
#cv2.imshow('invert', invert)
cv2.imshow('roi', roi)
cv2.waitKey()
PS C:\Users\obeli\Desktop\COS 450\class exercises\Ass2> & C:/Users/obeli/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/obeli/Desktop/COS 450/class exercises/Ass2/A2Q2.2.py"
AHEAD



The code that I used in these examples seems to have correctly identified all the letters that are visible, but also added letters that were not there because of the noise of the image. If I remove the region of interest and process Sign1 again, this is the result:

MORPH WITHOUT MANUAL ROI

Sign1: accuracy = 82% (14/17)

DRIVE CAREFULLY," 7

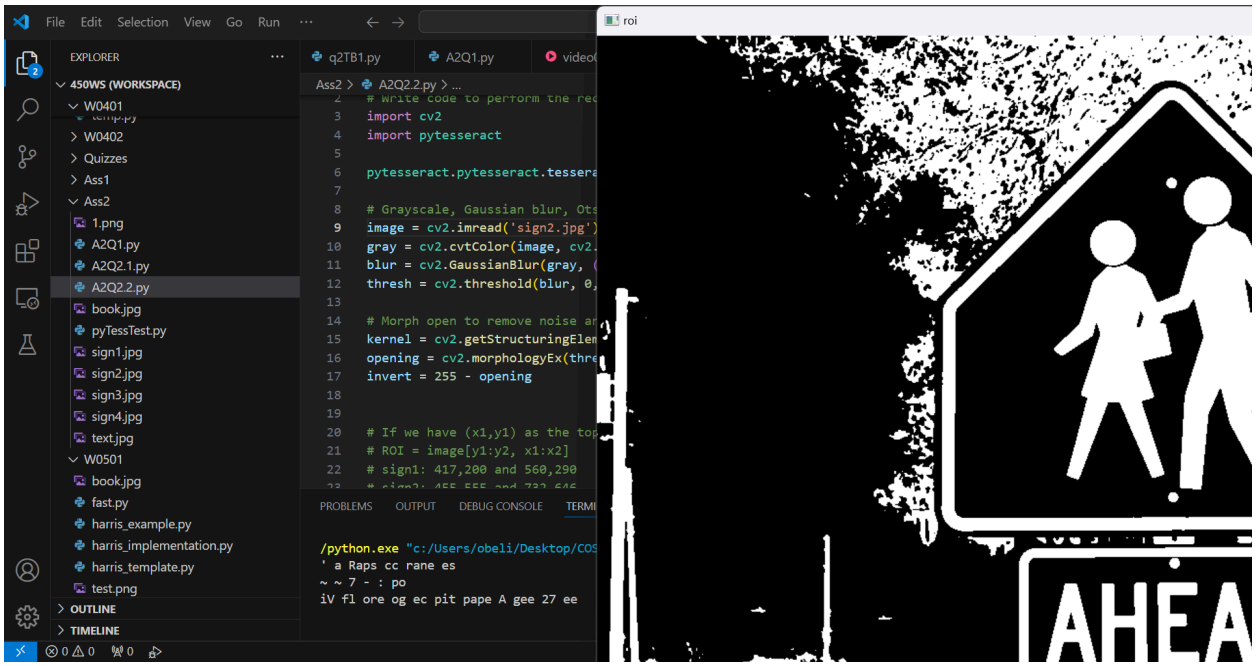
vs

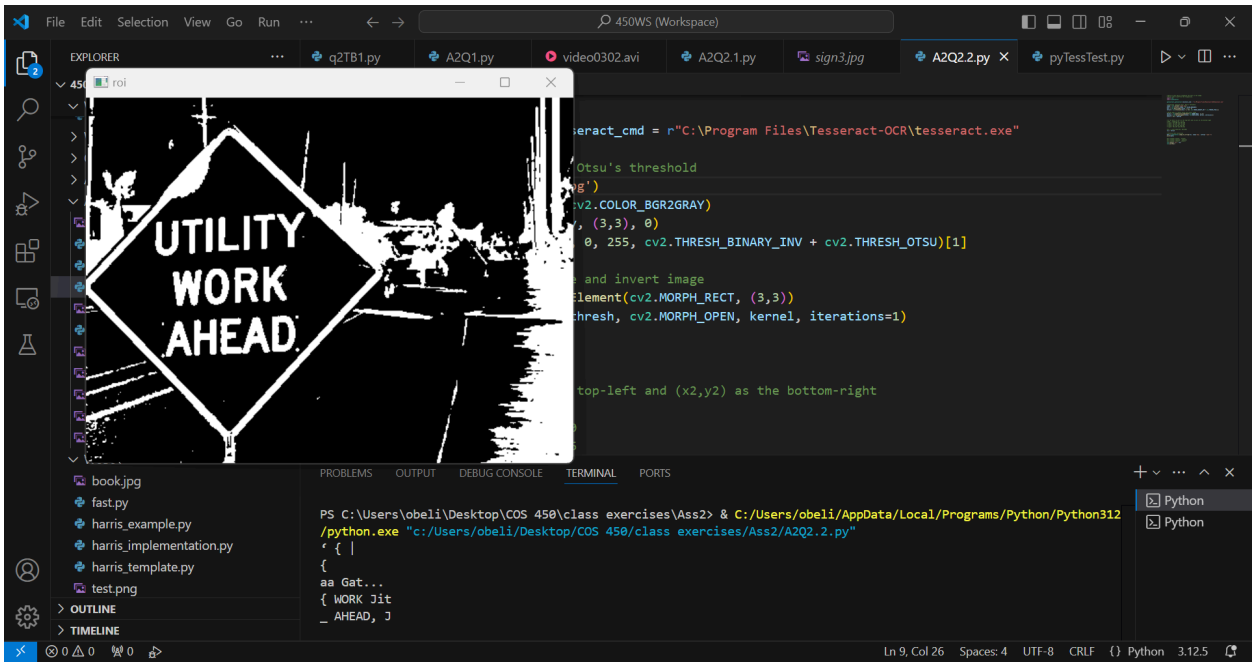
DRIVE CAREFULLY

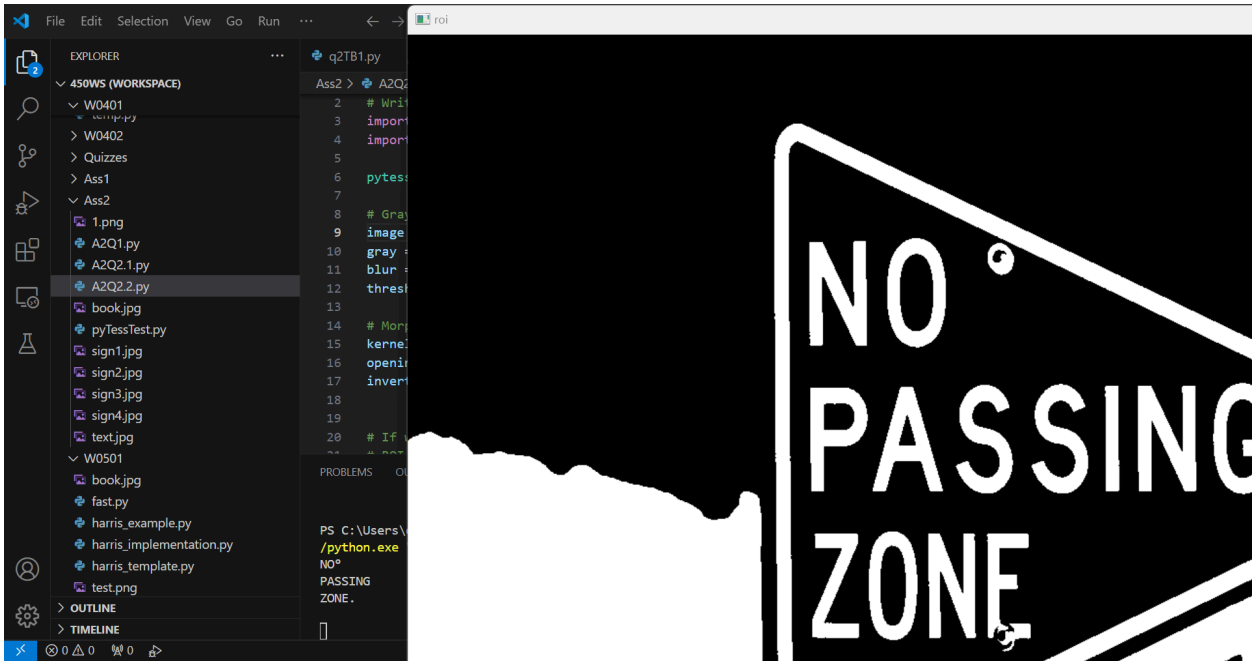
Here are the results without the manual ROI and without the morphological functions:
NO MORPH NO MANUAL ROI:
Sign1: accuracy = 100% (14/14)
DRIVE CAREFULLY
vs
DRIVE CAREFULLY
Sign2: accuracy = 2% (1/47)
' a Raps cc rane es
~ ~ 7 - : po
iV fl ore og ec pit pape A gee 27 ee
vs
AHEAD
Sign3: accuracy = 32% (9/28)
‘ { |
{
aa Gat...
{ WORK Jit
_ AHEAD, J
vs
UTILITY
WORK
AHEAD
Sign4: accuracy = 86% (13/15)
NO°
PASSING
ZONE.

vs

NO
PASSING
ZONE




roi
UTILITY
WORK
AHEAD
A2Q2.2.py
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
PS C:\Users\obeli\Desktop\COS 450\class exercises\Ass2> & C:/Users/obeli/AppData/Local/Programs/Python/Python312
/python.exe "c:/Users/obeli/Desktop/COS 450/class exercises/Ass2/A2Q2.2.py"
Python
Python
Ln 9, Col 26
Spaces: 4
UTF-8
CRLF
Python
3.12.5


roi
NO
PASSING
ZONE
450WS (WORKSPACE)
W0401
W0402
Quizzes
Ass1
Ass2
1.png
A2Q1.py
A2Q2.1.py
A2Q2.2.py
book.jpg
pyTessTest.py
sign1.jpg
sign2.jpg
sign3.jpg
sign4.jpg
text.jpg
W0501
fast.py
harris_example.py
harris_implementation.py
harris_template.py
test.png
OUTLINE
TIMELINE