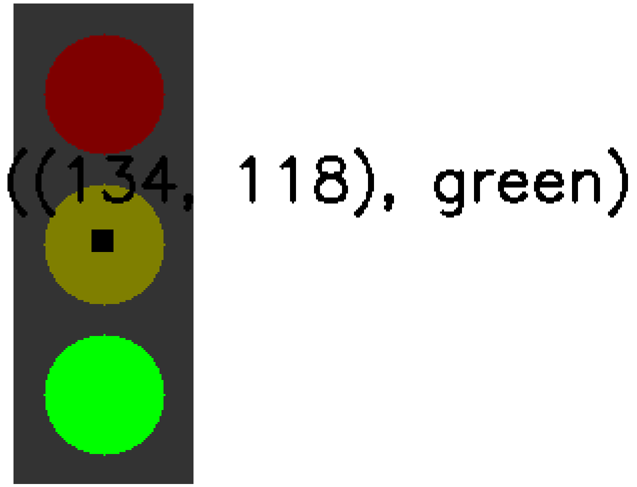
Computer Vision
Spring 2018
Problem Set #2

Xiangnan He
xhe321@gatech.edu

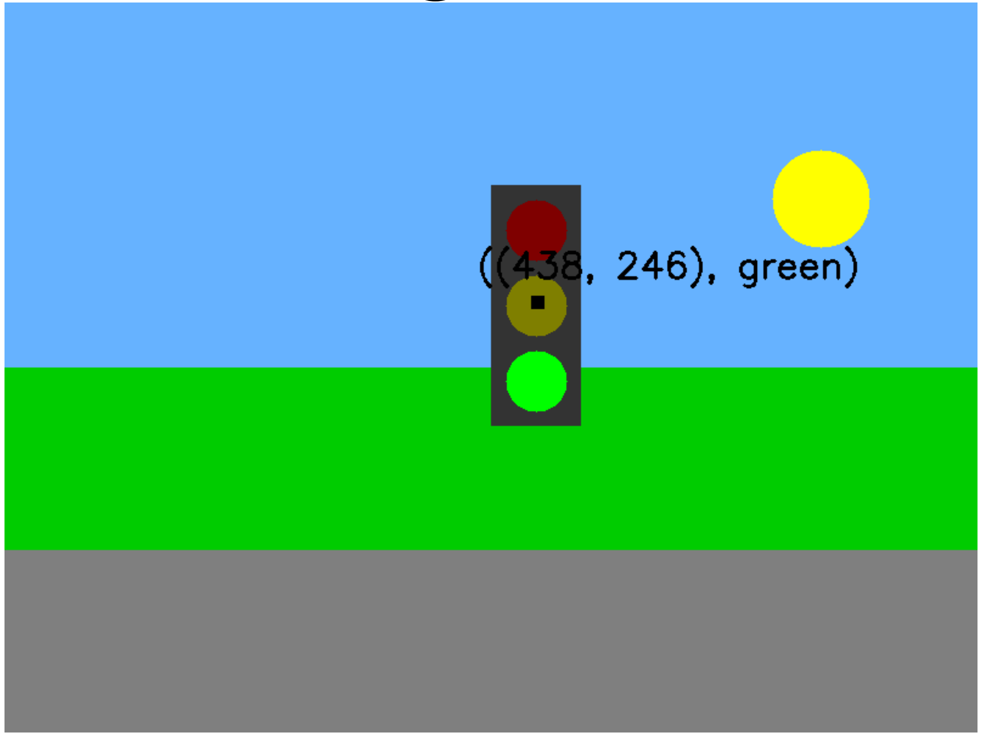
Traffic Light Detection



Coordinates and State:
((134,118), green)

ps2-1-a-1.png

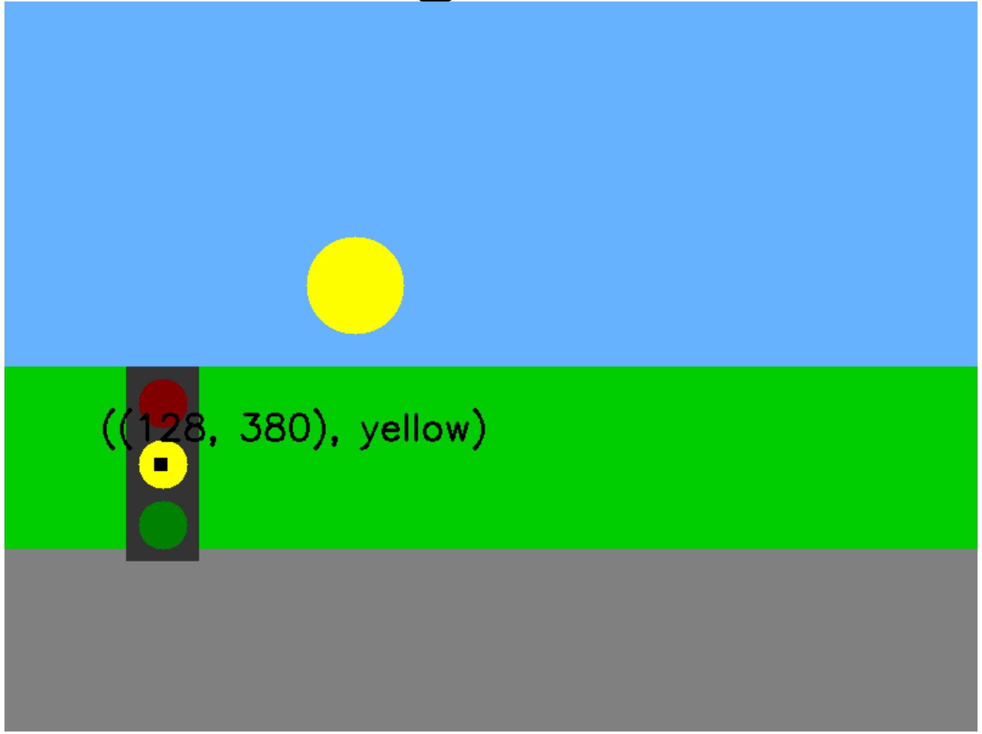
Traffic Light Detection



ps2-1-a-2.png

Coordinates and State:
(438,246), green

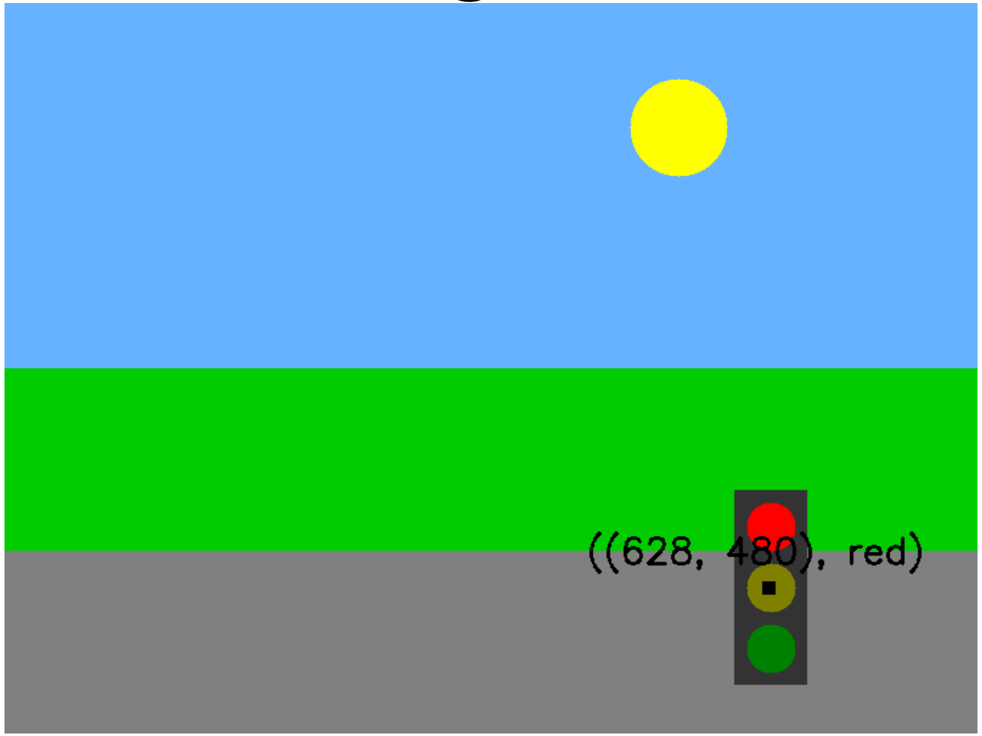
Traffic Light Detection



ps2-1-a-3.png

Coordinates and State:
(128, 380), yellow

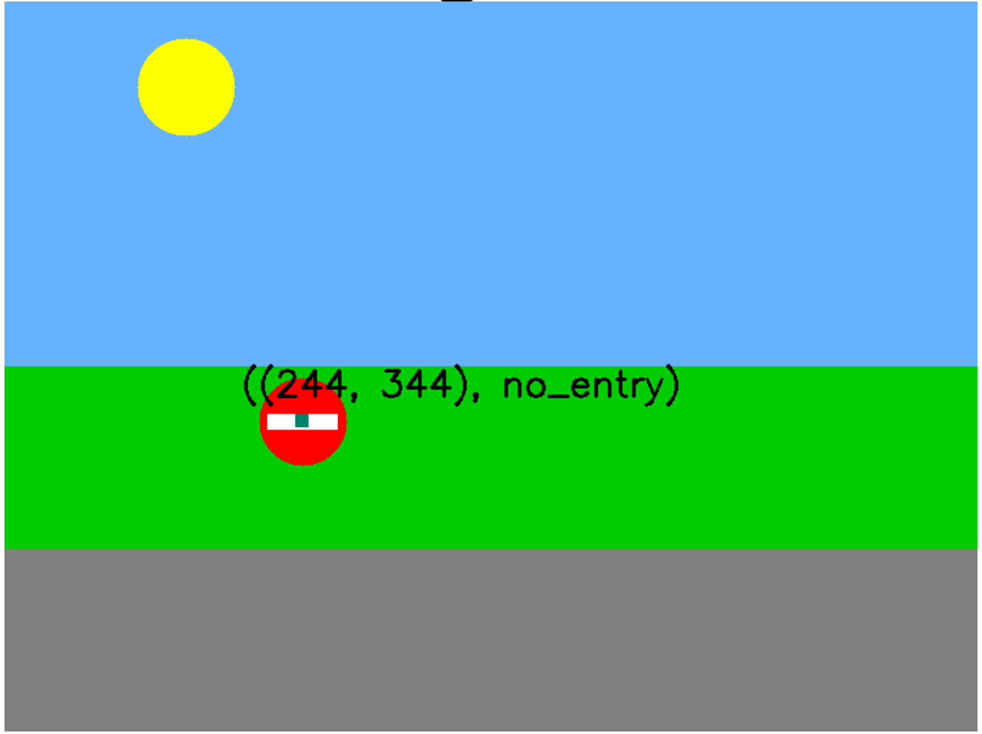
Traffic Light Detection



ps2-1-a-4.png

Coordinates and State:
(628,480), red

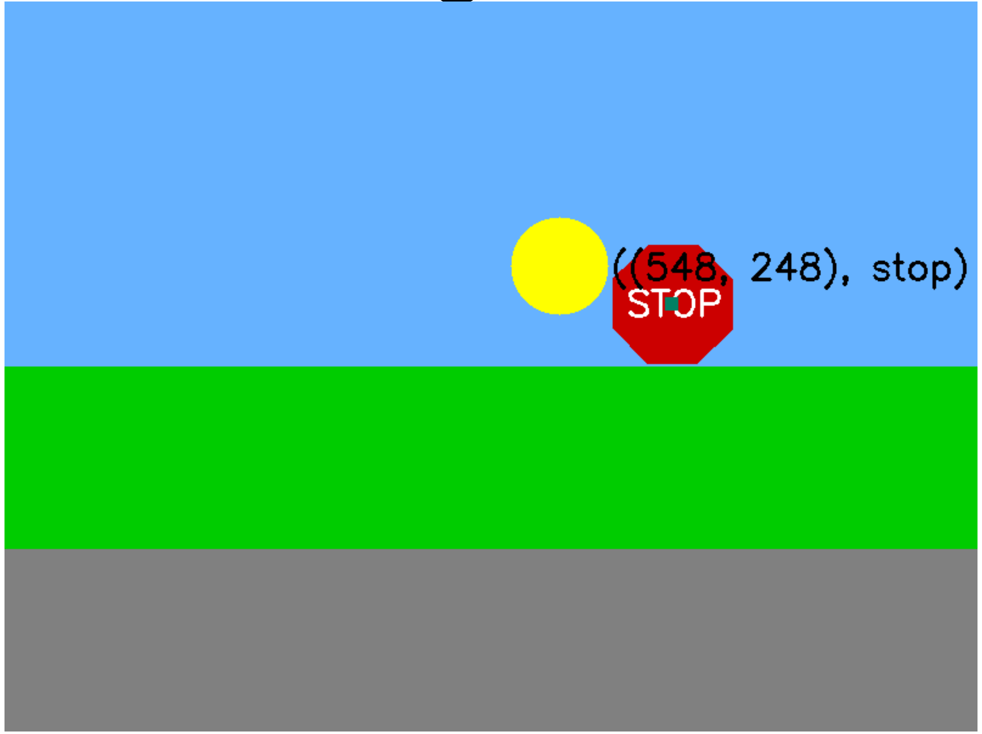
Traffic Sign Detection - Do not enter



ps2-2-a-1.png

Coordinates: (244, 344)

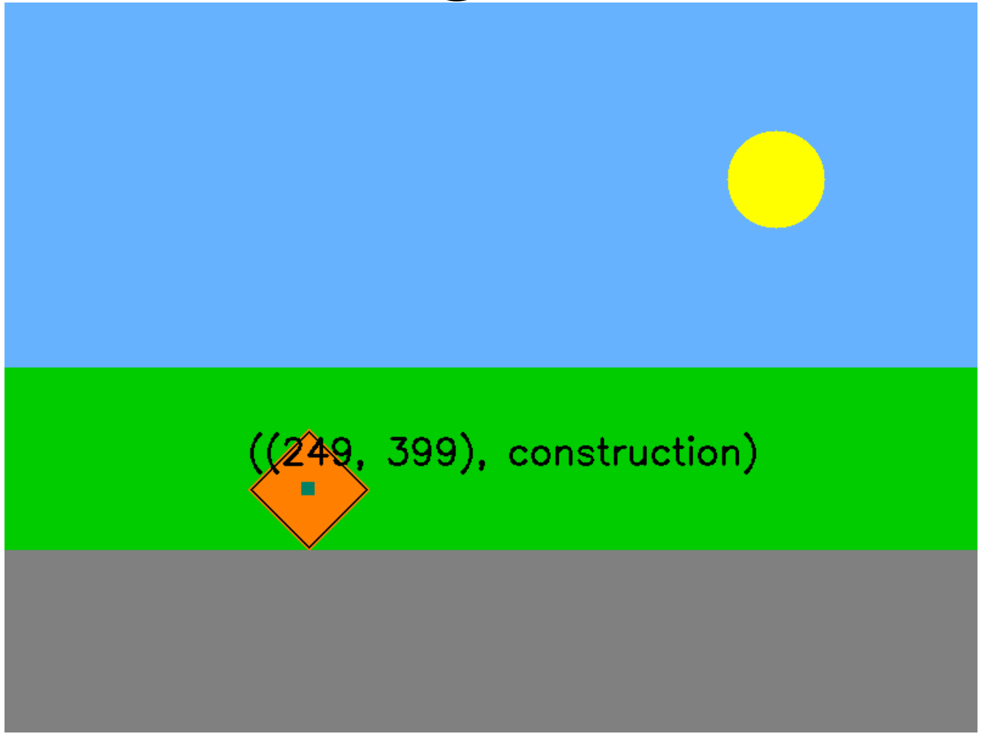
Traffic Sign Detection - Stop



ps2-2-a-2.png

Coordinates: (548, 248)

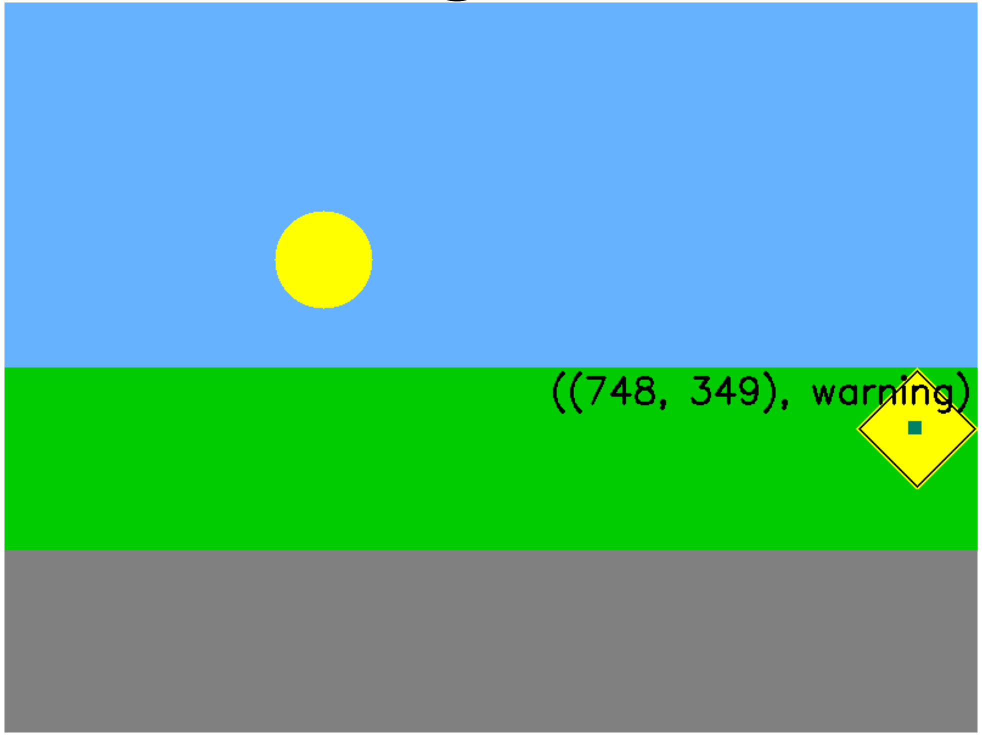
Traffic Sign Detection - Construction



ps2-2-a-3.png

Coordinates: (249,399)

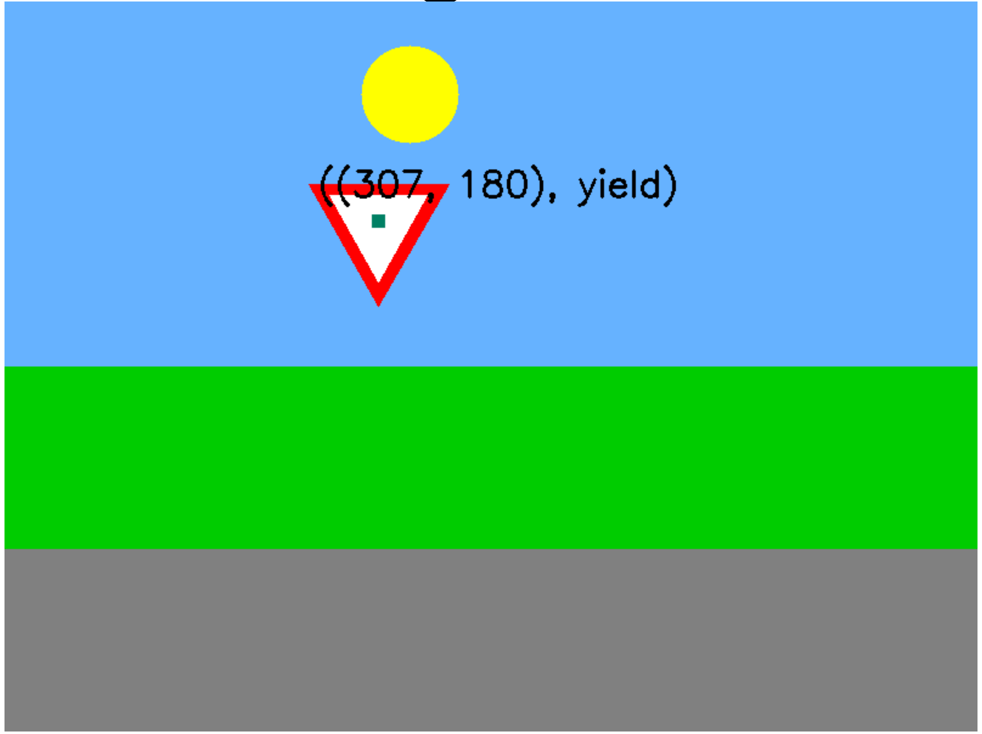
Traffic Sign Detection - Warning



ps2-2-a-4.png

Coordinates: (748,349)

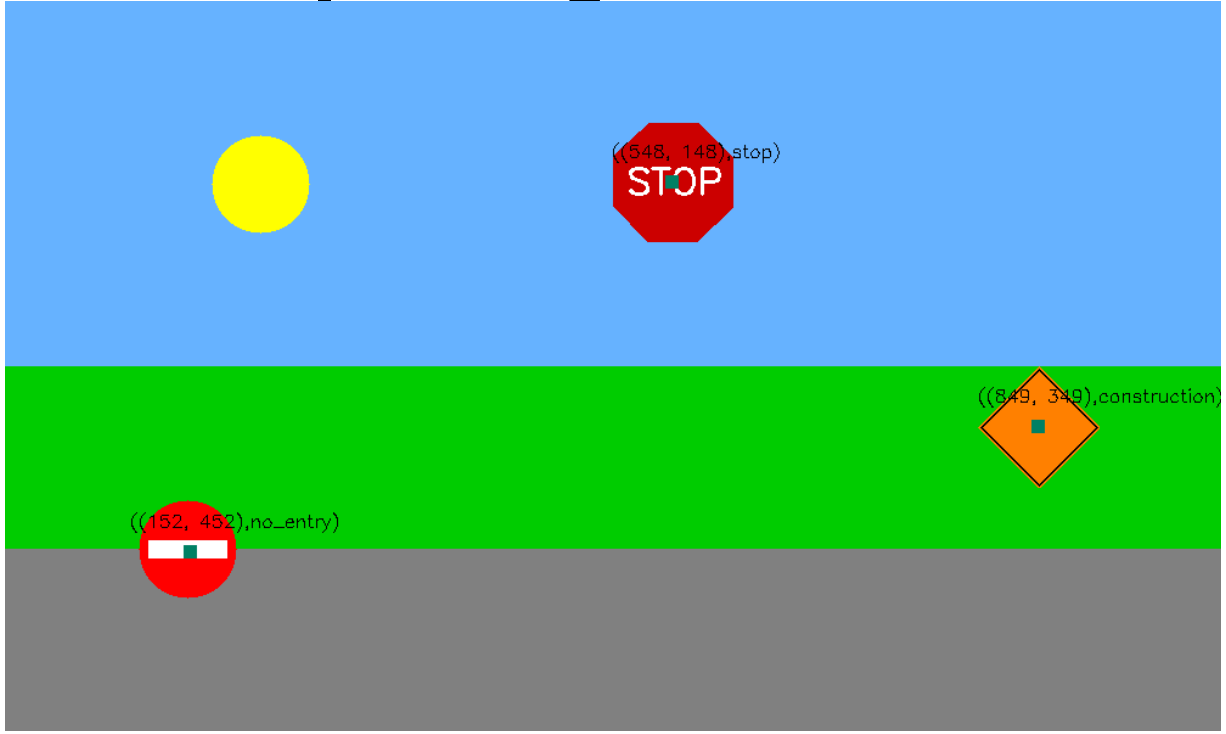
Traffic Sign Detection - Yield



ps2-2-a-5.png

Coordinates: (307,180)

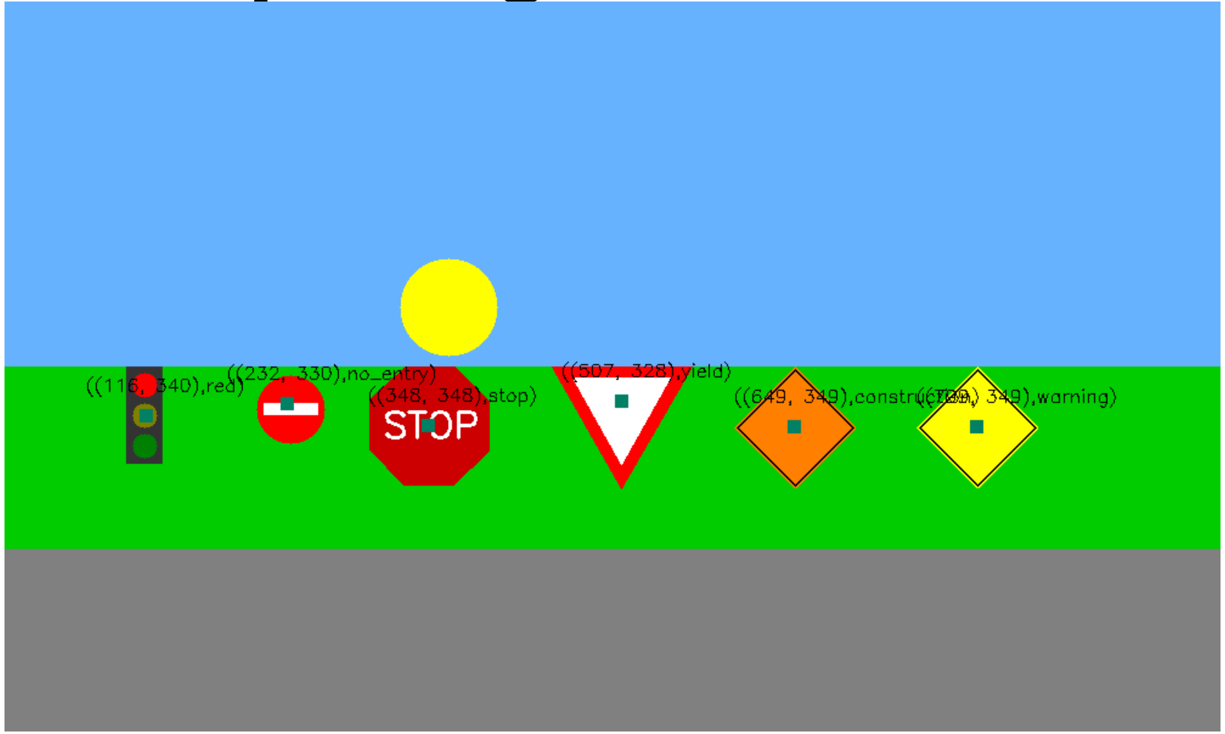
Multiple sign detection



ps2-3-a-1.png

Coordinates and Name
(548,148), stop
(152,452), no_entry
(849,349), construction

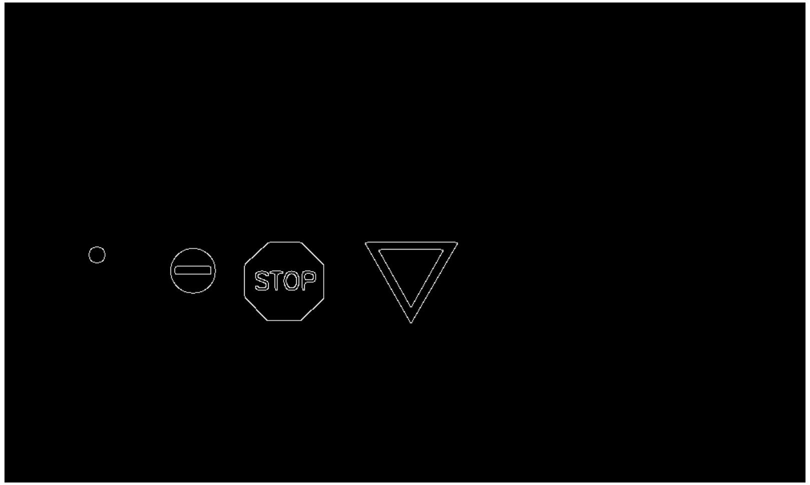
Multiple sign detection



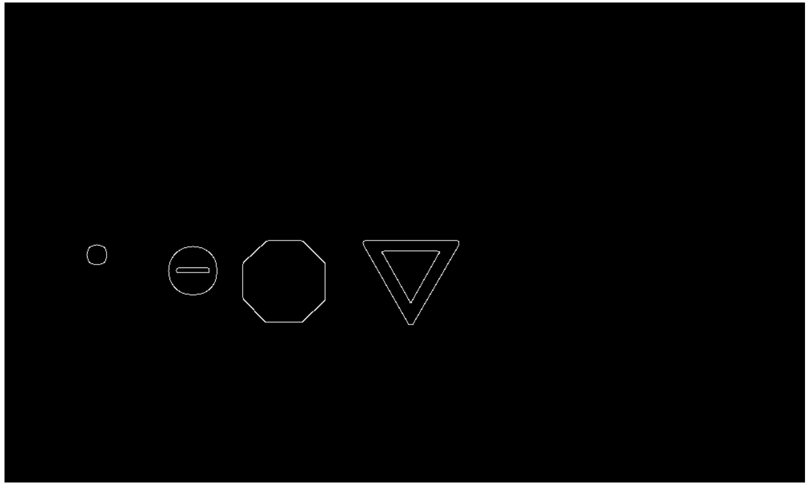
ps2-3-a-2.png

Coordinates and Name:
(116,340) red
(232,330), no_entry
(348,348), stop
(507,328), yield
(649,349), construction
(799,349), warning

Multiple sign detection – dilate effect



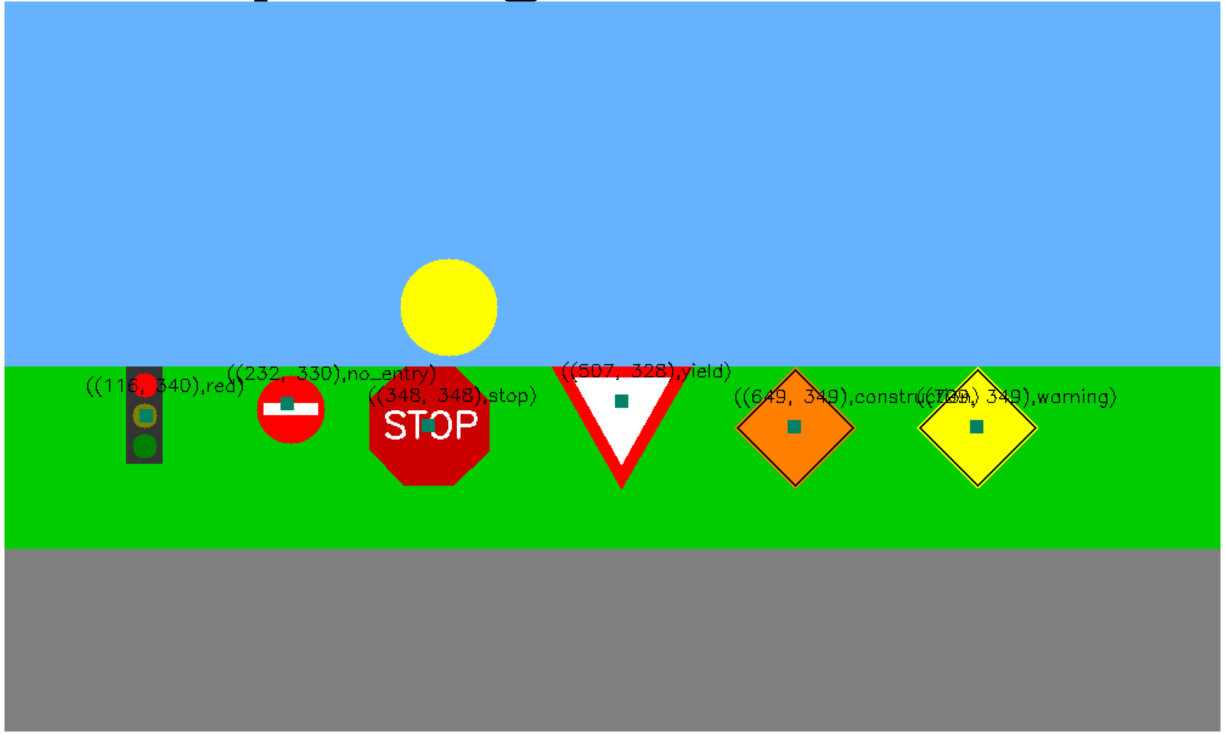
Dilate



ps2-3-a-2.png

Dilate removes “STOP”, making the later on processing easier

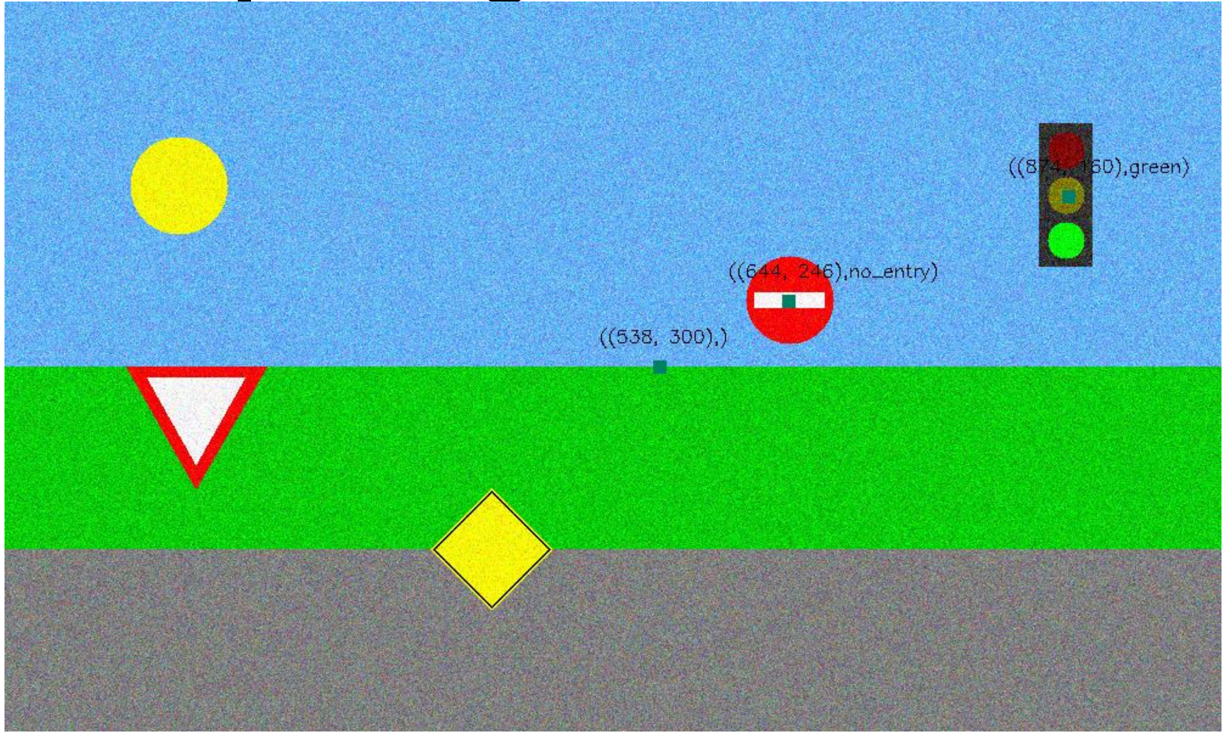
Multiple sign detection



ps2-3-a-2.png

Coordinates and Name:
(116,340) red
(232,330), no_entry
(348,348), stop
(507,328), yield
(649,349), construction
(799,349), warning

Multiple sign detection with noise

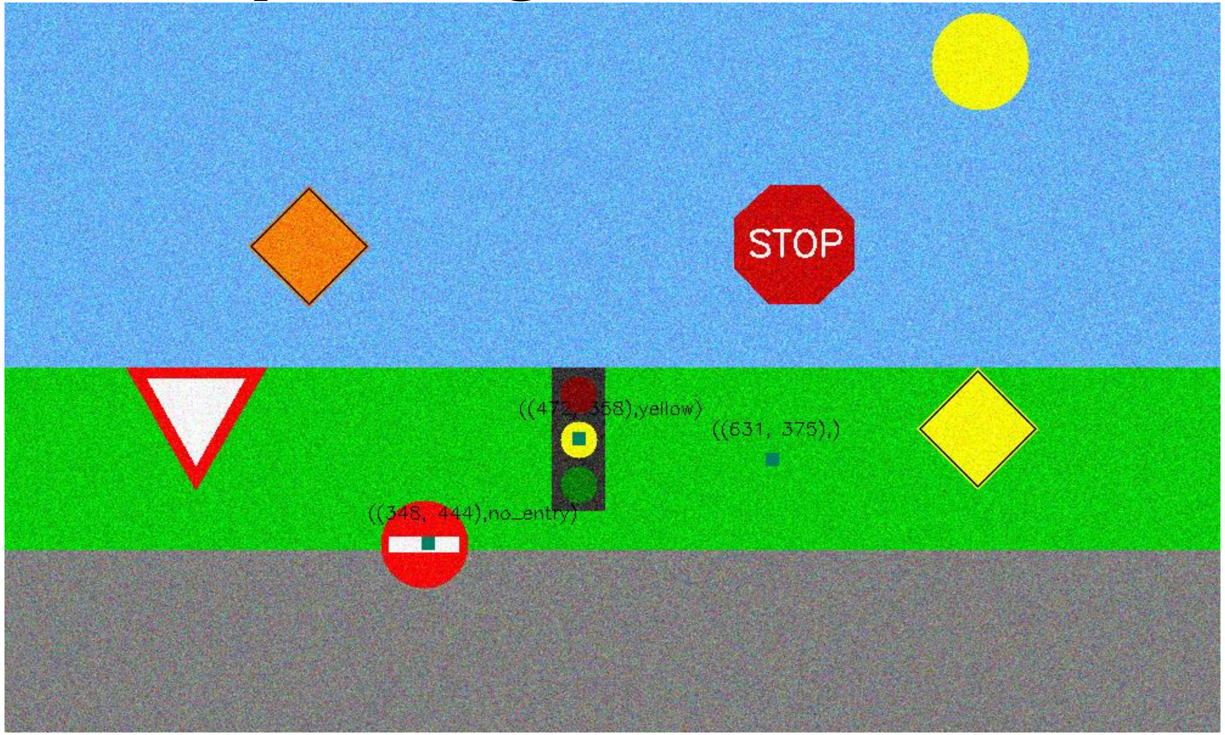


ps2-4-a-1.png

Coordinates and Name:
(644,246), no_entry
(874,160), green

Further parameters optimization is needed to detect other signs

Multiple sign detection with noise



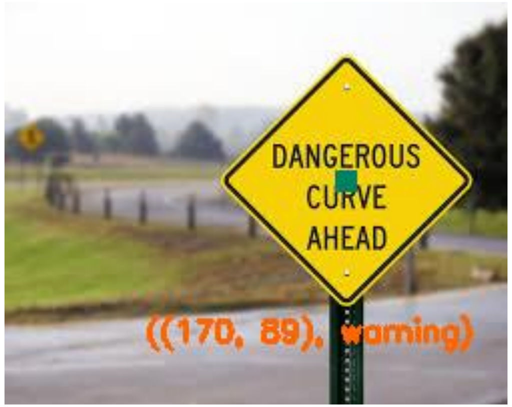
ps2-4-a-2.png

Coordinates and Name:
(348,444), no_entry
(472,358), yellow

Further parameters optimization is needed to detect other signs

Challenge Problem

Input image



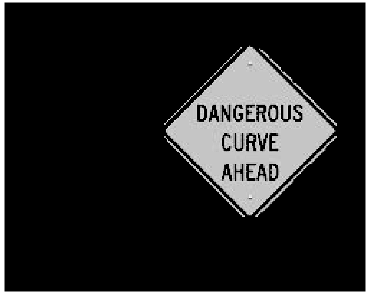
Output image

Coordinates and Name:
(170,89), warning

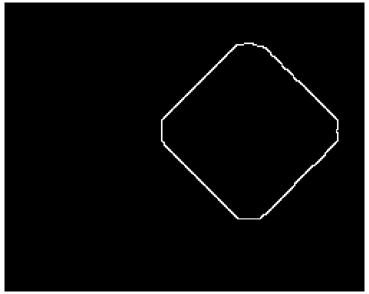
Image size: 251x201pixel

Challenge Problem – More Details

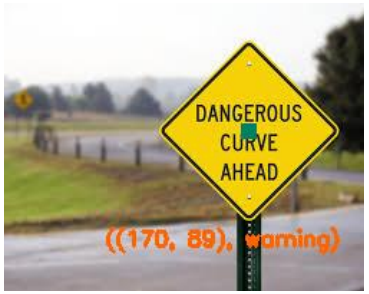
Input image



HSV, Mask

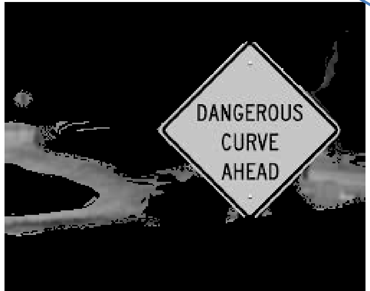


Canny



Output image

HSV color mask



HSV value range optimization

Challenge Problem

Describe what you had to do to adapt your code for this task.
Answer:
1) For traffic light, after convert to HSV image, I used two rounds of masks, one is for when the light is on, another is for all the lights, regardless of whether it is on or not, so that I can use the first one to determine the state, and the latter one to locate the traffic light;
2) For each round of masks, there are total three masks, one for red, one for yellow, and another for green. For red, we should consider two cases hue in (0,10) and (170, 180). Yellow hue is in (20, 40), while green hue in (50, 70). When determine if the light is on, I adjust the range of value from (50, 250) to (150, 250), which is brighter than the former one.
3) To find the location of the traffic light, I basically use center of yellow circle, but since there are images with sun in it, so I have to make sure the yellow circle is the right one. I used if statement, to make sure, only when the green, yellow, red has close enough (+/-5 pixels) x-axis value, and their distance is also within +/-5 pixel, and the y-axis of red, yellow, and green are in increasing order.
4) To determine the state of the traffic light, I loop through the circles from cv2.HoughCircles, and find the right sized (+/-5) of the center yellow light (center_x, center_y)

Challenge Problem

How does the difference between simulated and real-world images affect your method?
Answer:
1) Real world images normally is noisy, hard to find the clean outline of the signs, I had to add denoise processes, such as cv2.fastNlMeansDenoisingColored(image,None,20,20,7,21).
2) I also converted image to HSV and mask the image using specific color range of the sign.
3) Further, I also changed the value in HSV, to further mask the image to find the right brightness. Then I apply cv2.Canny and cv2.HoughCircles or cv2.HoughLinesP to find the correct circles and lines.
4) In “multiple sign detection with noise”, there are still signs not detectable, due to that I don't have time to further optimize the parameters.
5) In real world images, the warning sign was correctly detected by applying the methods above, given that the sign is clear in the image, for some real world images with more noise, further adjustment will be needed to detect.

Challenge Problem

If you used other functions/methods, explain why that was better(or why your previous implementation did not work)
Answer:
1) For the yield sign detection, I originally tried to calculate centroid based on the outer three lines, but found out that sometimes, the length of the line will cause major shift, therefore, I improved the algorithm by just calculate based on the outer horizontal light, since the triangle is equilateral, therefore, it is possible to just calculated the centroid based on the length and coordinate of the outer horizontal line;
2) For the single sign detection, I optimized the HoughCircles or HoughLinesP to detect with no problem, but when changed to multiple sign detection, I had to figure out how to distinguish/classify between signs, for example, when detect scene_all_signs.png, I used HSV red mask to separate out traffic red light, no_entry, yield, and stop signs, then, I defined a function to group the lines by their end point closeness (+/-5 pixel), when the group has 8 lines, it is stop sign, when 6, it is yield, I ended up use another round of HoughCircle to find no_entry.
3) For stop sign detection, I originally didn't use dilate, but found that the letters “STOP” in the center will cause the program to detect wrong lines, therefore, I used dilate, and was able to perfectly remove “STOP” from the cv2.Canny.
4) I also had to make a post process loop to remove duplicate lines for the classification algorithm to work.