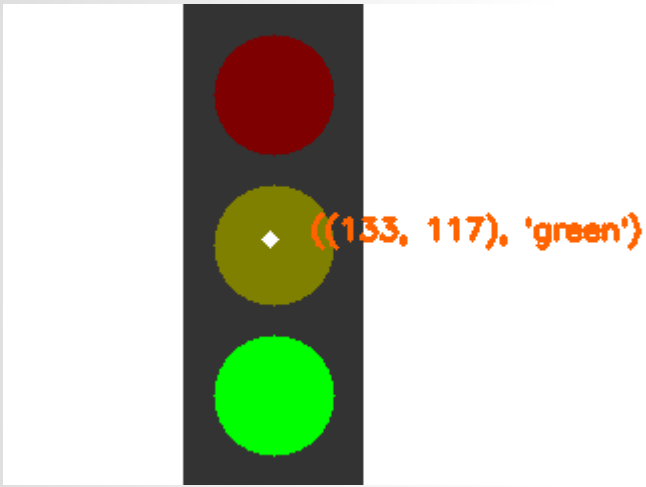
Computer Vision
Fall 2017
Problem Set #2

Zhi Zhang
zhizhang@gatech.edu

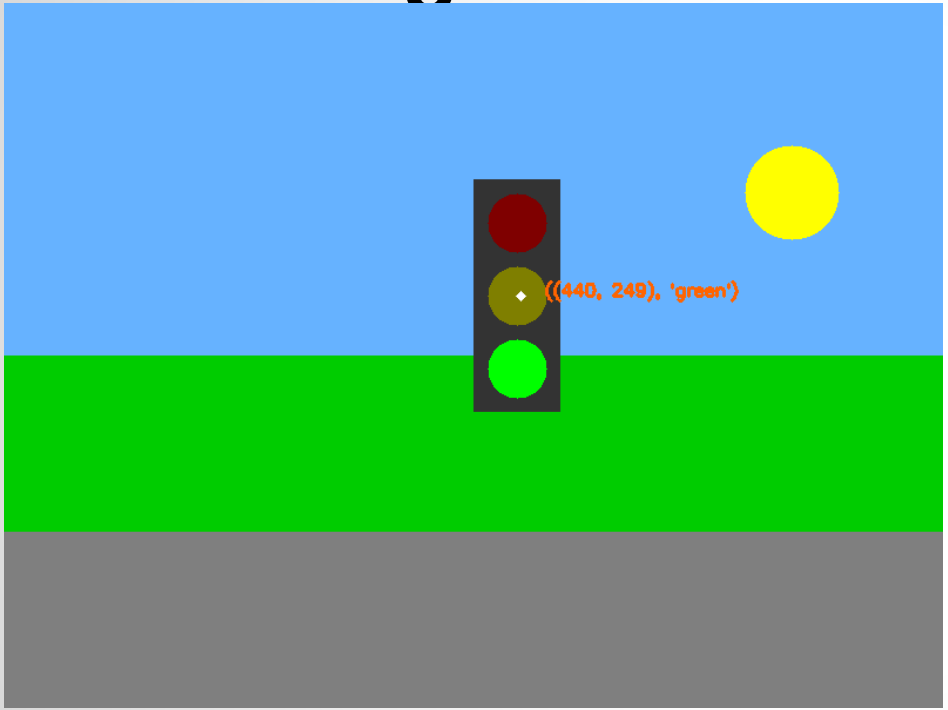
Traffic Light Detection

Coordinates and State



ps2-1-a-1.png

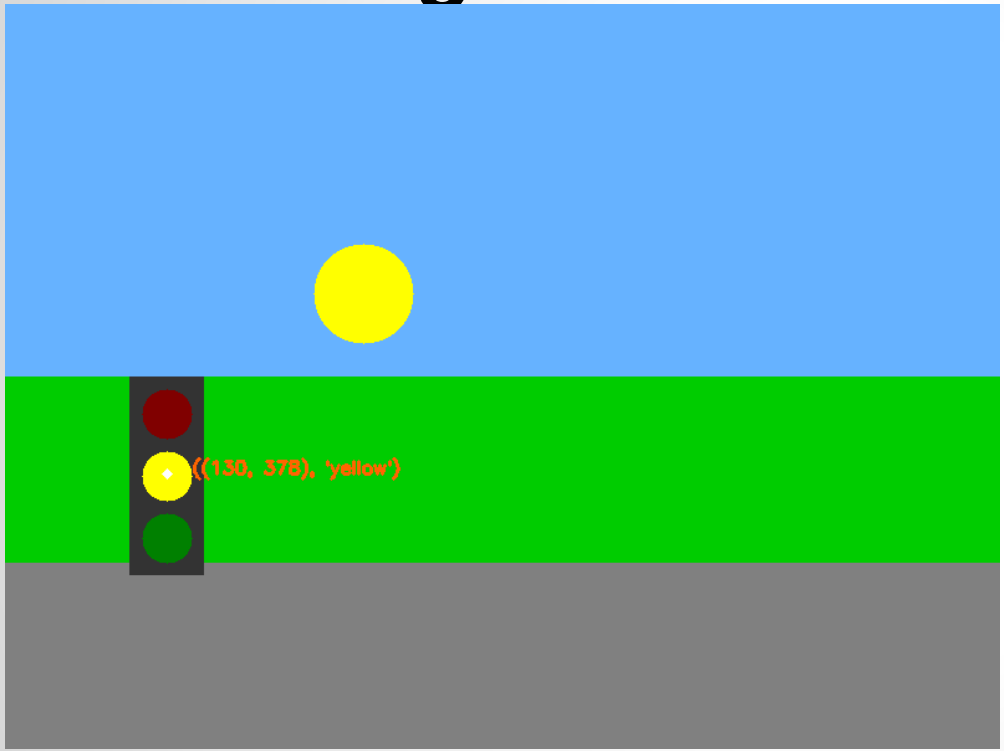
Traffic Light Detection



Coordinates and State

ps2-1-a-2.png

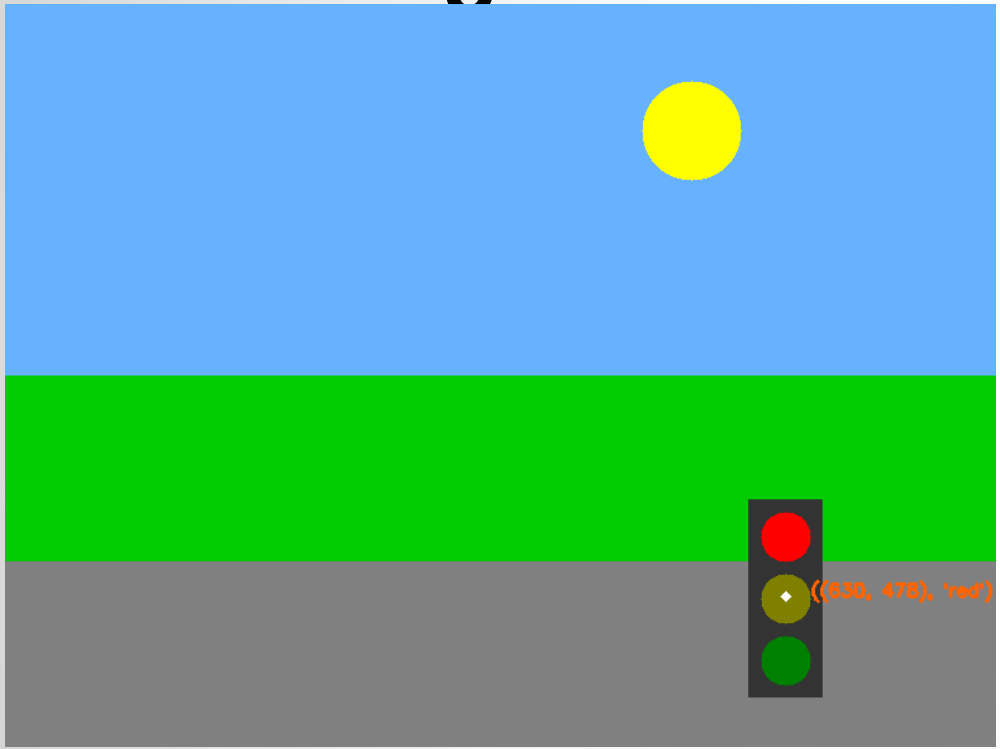
Traffic Light Detection



ps2-1-a-3.png

Coordinates and State

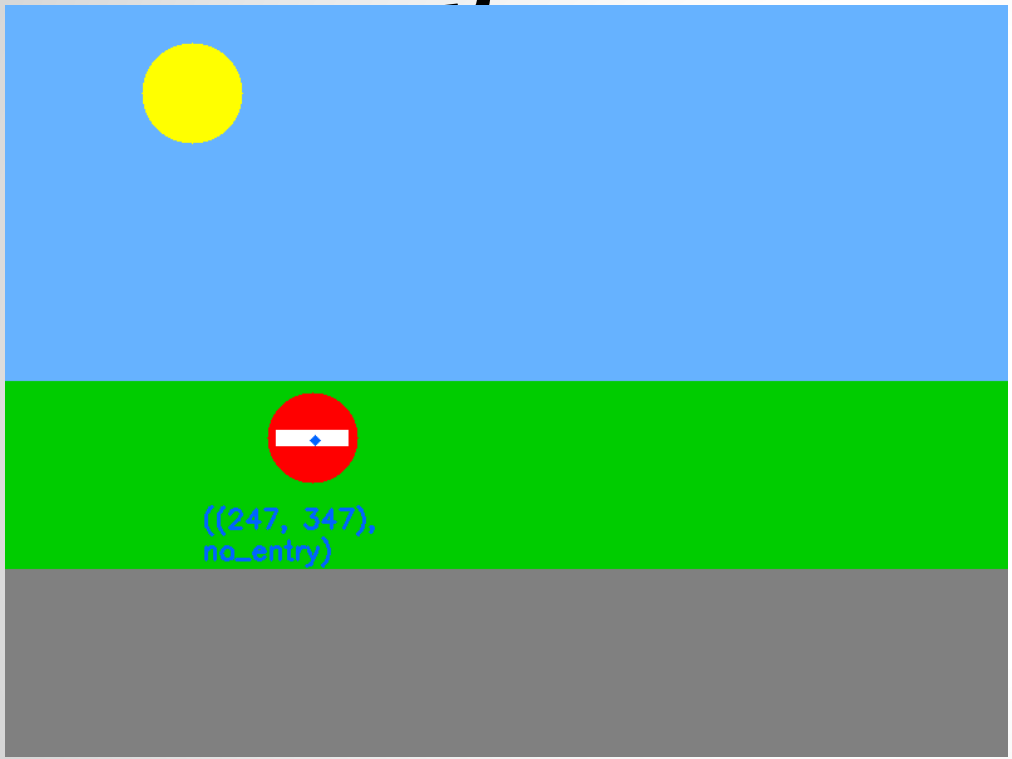
Traffic Light Detection



ps2-1-a-4.png

Coordinates and State

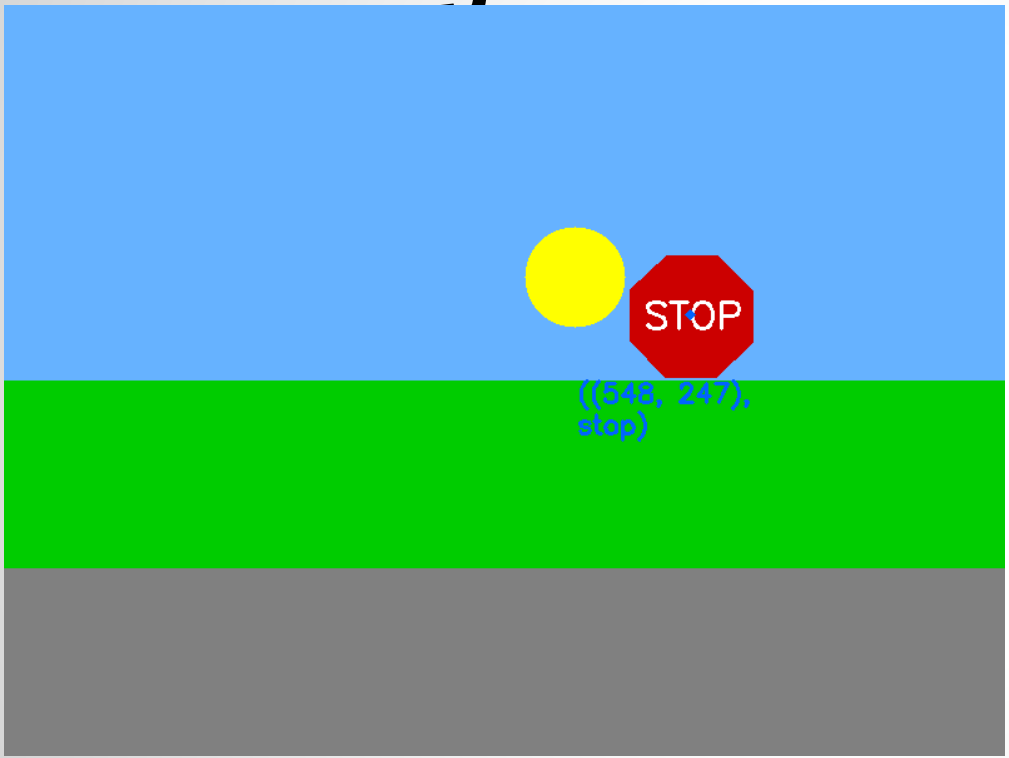
Traffic Sign Detection - Do not enter



ps2-2-a-1.png

Coordinates

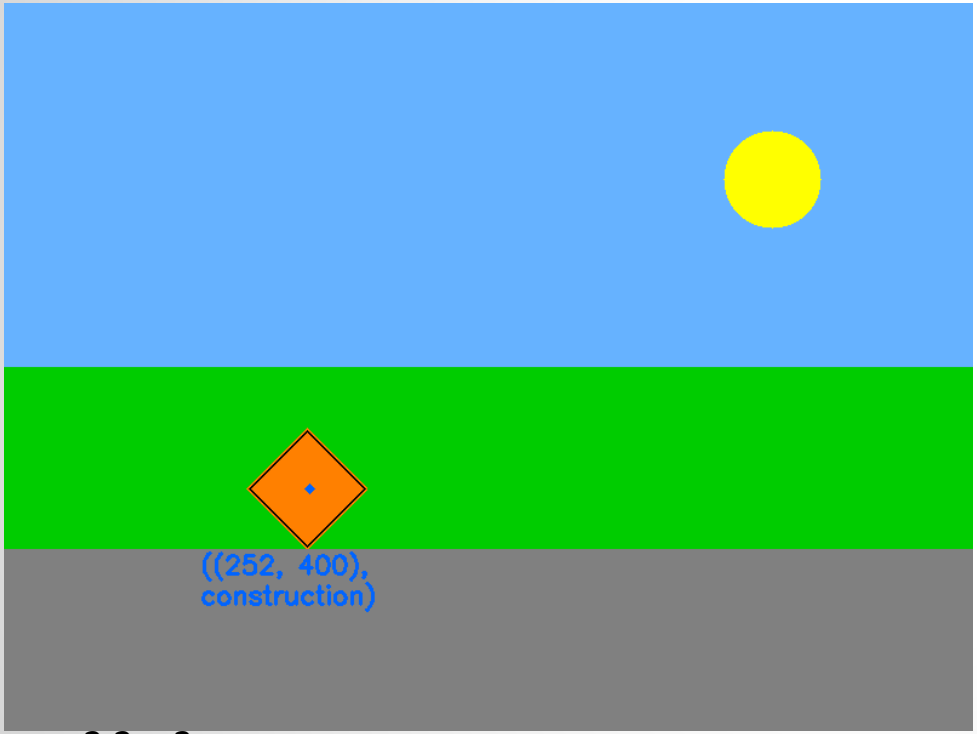
Traffic Sign Detection - Stop



ps2-2-a-2.png

Coordinates

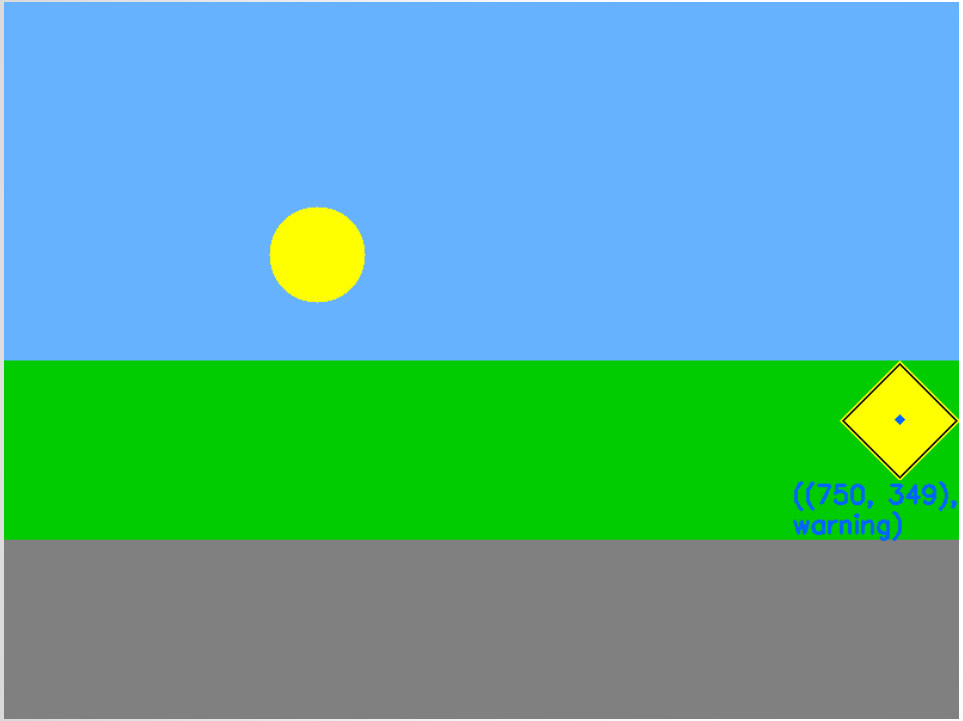
Traffic Sign Detection - Construction



ps2-2-a-3.png

Coordinates

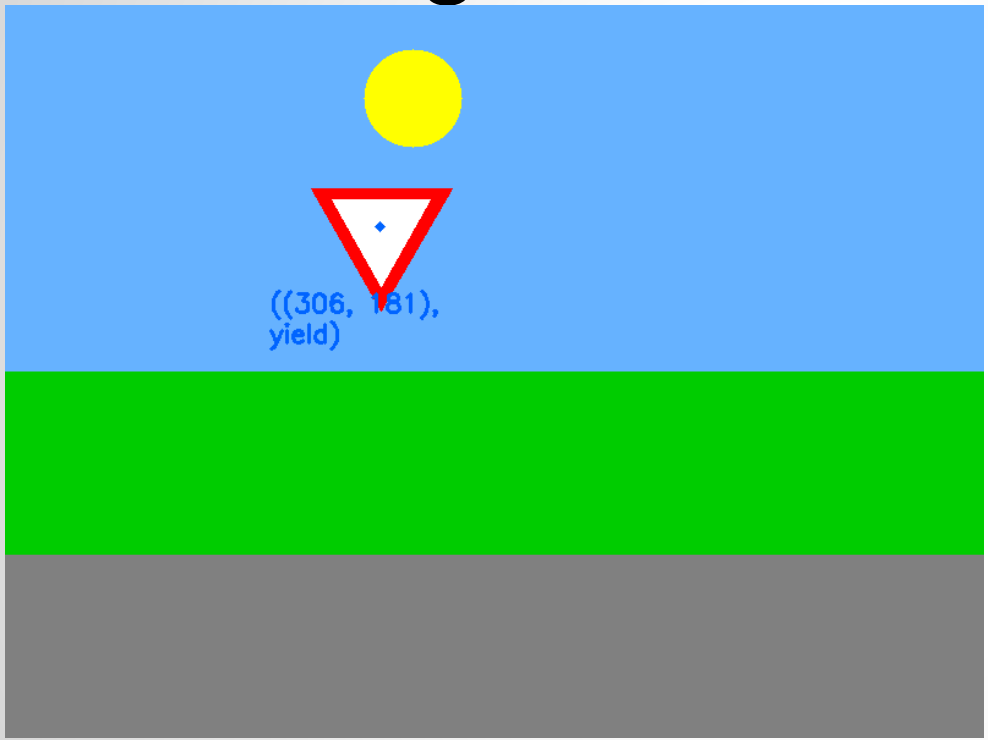
Traffic Sign Detection - Warning



Coordinates

ps2-2-a-4.png

Traffic Sign Detection - Yield

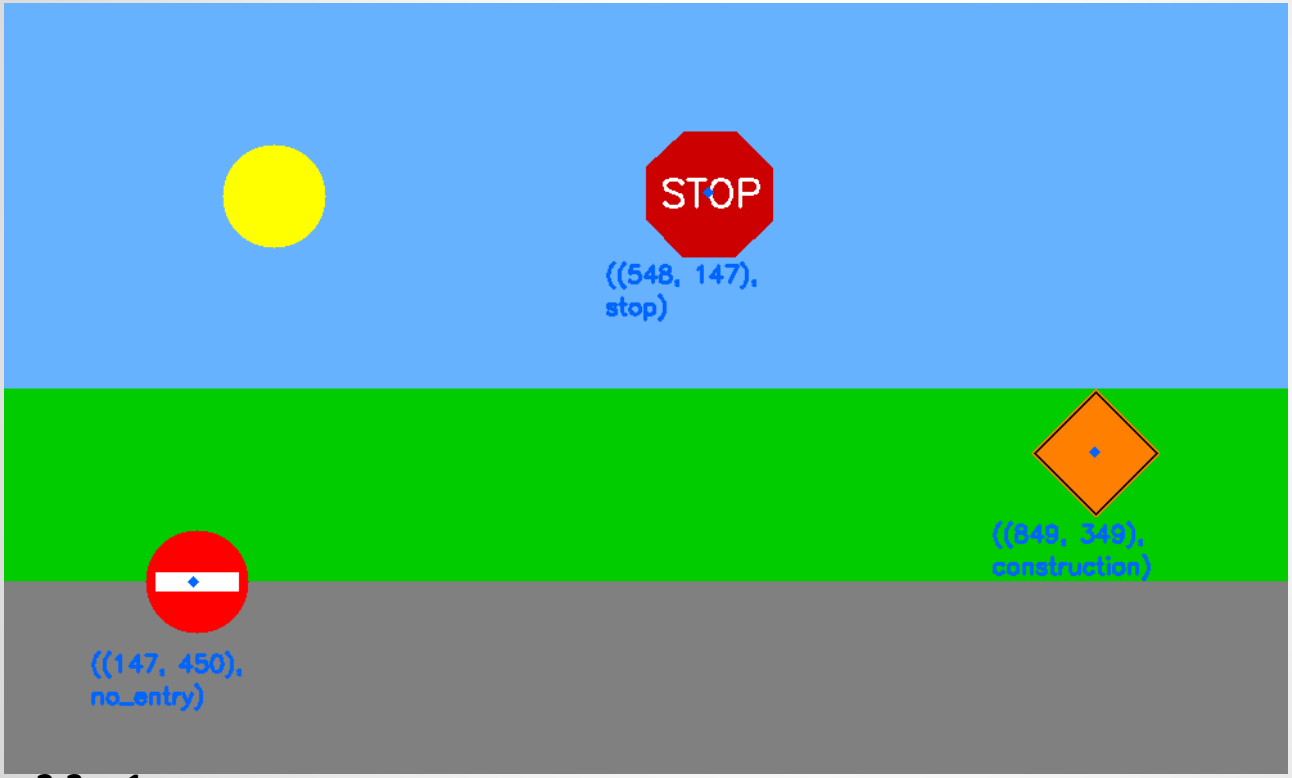


ps2-2-a-5.png

Coordinates

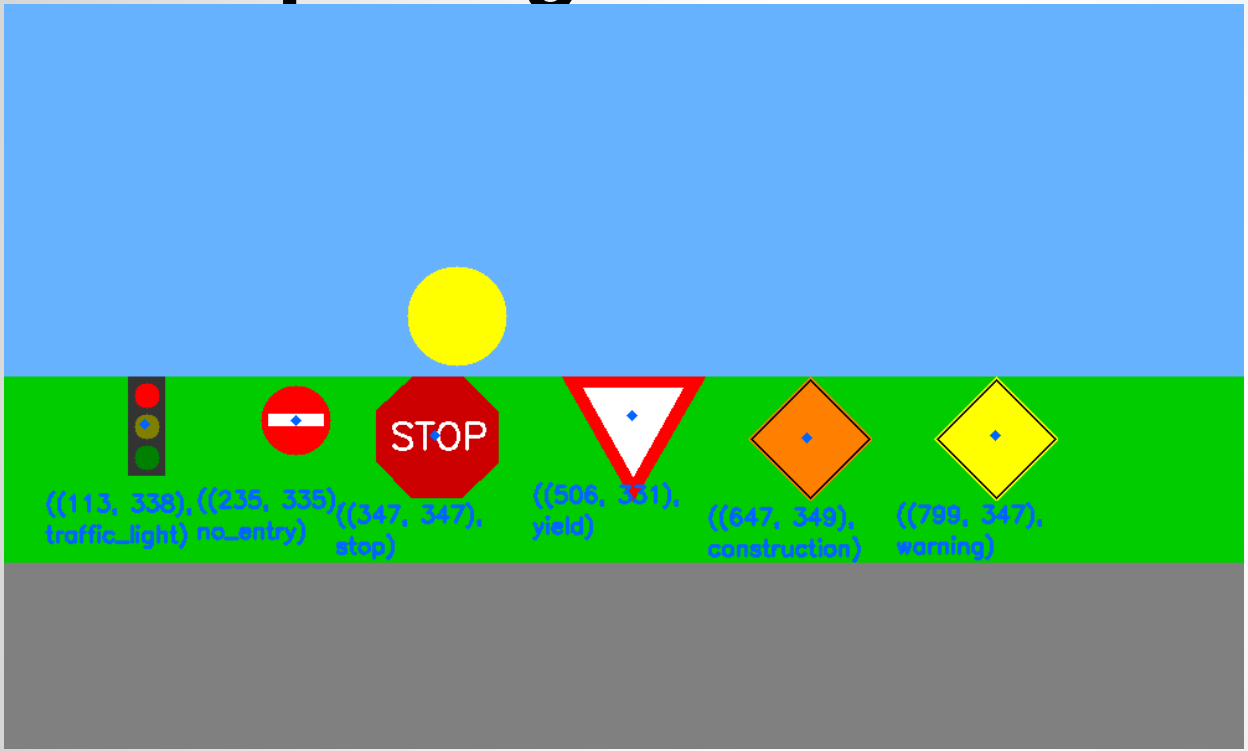
Multiple sign detection

Coordinates and Name



ps2-3-a-1.png

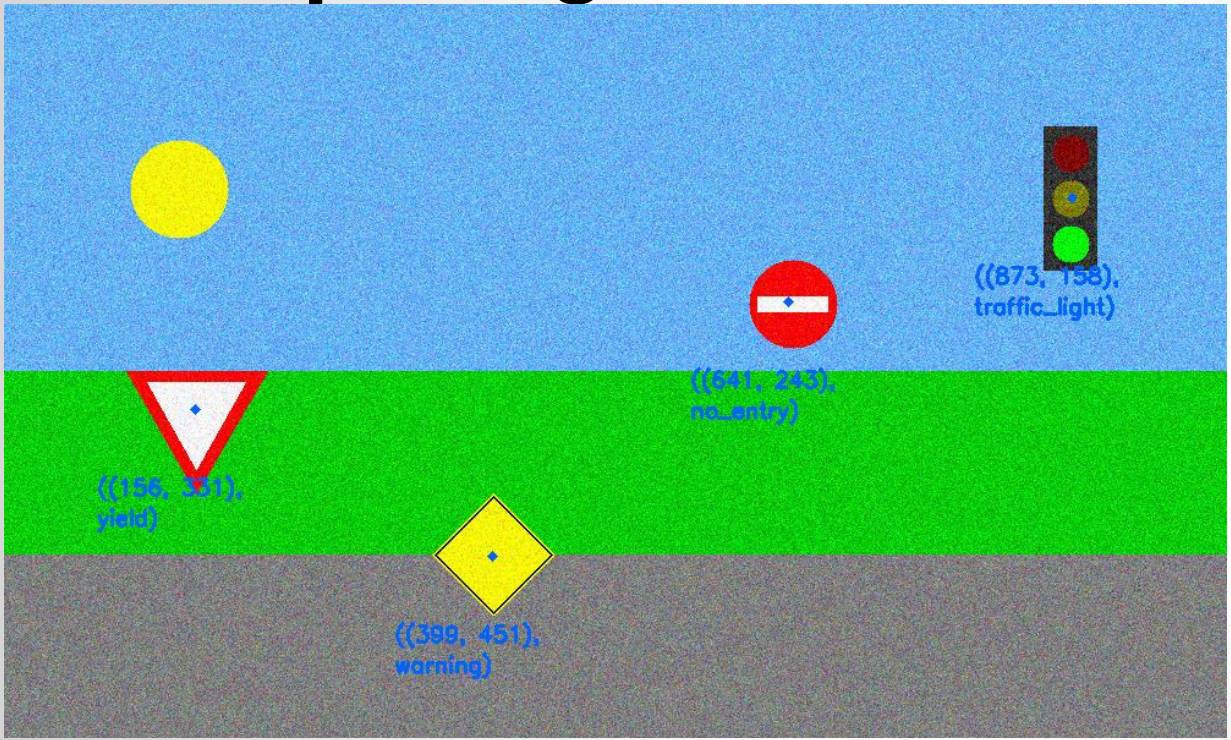
Multiple sign detection



ps2-3-a-2.png

Coordinates and Name

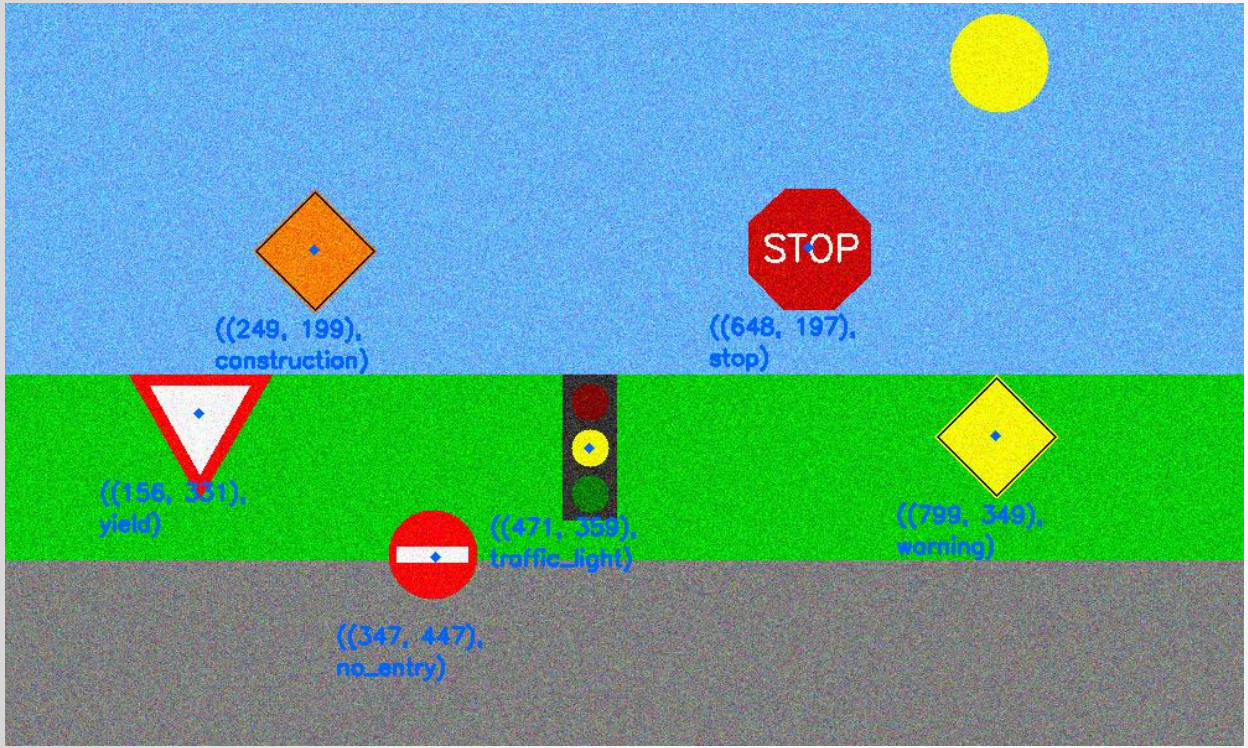
Multiple sign detection with noise



ps2-4-a-1.png

Coordinates and Name

Multiple sign detection with noise



Coordinates and Name

ps2-4-a-2.png

Challenge Problem

Coordinates and Name

Input image: img-5-a-1.png

Output image: ps2-5-a-1.png

Challenge Problem

Coordinates and Name

Input image: img-5-a-2.png

Output image: ps2-5-a-2.png

Challenge Problem

Coordinates and Name

Input image: img-5-a-3.png

Output image: ps2-5-a-3.png

Challenge Problem

Coordinates and Name

Input image: img-5-b-1.png

Output image: ps2-5-b-1.png

Challenge Problem

Describe what you had to do to adapt your code for this task. How does the difference between simulated and real-world images affect your method?

If you used other functions/methods, explain why that was better(or why your previous implementation did not work)

The things I have to adapt my code for real-world images are:

- To expand the Hue value range for the target object. In order to differentiate with other objects, the first step I did was to use the Hue value range to extract the target traffic sign. In real-world images, colors are not always presenting as they should be, for example, my img-5-a-1.png has the construction sign which falls into the red color range instead of orange, if I only check the orange, then it won't show up, so I have to expand the color range to be the warm tune colors including both orange and red.
- To give more variations to check angles between edges. My standard approach to use HoughLines to locate a polygon traffic sign is: generate HoughLines – Using known angles to select lines – Calculate intersection points between lines – further selecting objects by using other conditions (equal side length). Real-world images sometimes are skewed compared to simulated images, so the angles are not exactly 60 degree for a yield sign and 90 degree for a construction sign, so I give more variation spaces to include more possible situations.
- To apply filters to smooth the image. Real-world images contain more objects than the simulated images, these objects (lines or circles) can cause false positives, so I used the cv2.GaussianBlur() method to smooth the image to reduce the possibilities of including those false positive objects.