Problem Set 2: Detecting Traffic Signs and Lights

Description

Problem Set 2 is aimed at introducing basic building blocks of image processing. Key areas that we wish to see you implement are: loading and manipulating images, producing some valued output of images, and comprehension of the structural and semantic aspects of what makes an image. Relevant Modules are 1-2.

For this and future assignments, we will give you a general description of the problem. It is up to the student to think about and implement a solution to the problem using what you have learned from the lectures and readings. You will also be expected to write a report on your approach and lessons learned.

Learning Objectives

- Identify how images are represented using 2D and 3D arrays.
- Learn the representation of color channels in 3D arrays and the predominance of a certain color in an image.
- Use Hough tools to search and find lines and circles in an image.
- Use the results from the Hough algorithms to identify basic shapes.
- Understand how objects can be selected based on their pixel locations and properties.
- Address the presence of distortion / noise in an image.
- Identify what challenges real-world images present over simulated scenes.

Problem Overview

Methods to be used: In this assignment you are to use methods that work with the Hough Transform and line finding. You should also analyze how each image is composed and what patterns can be used to identify objects in a scene.

RULES: You may use image processing functions to find color channels, load images, find edges (such as with Canny). Don't forget that those have a variety of parameters and you may need to experiment with them. There are certain functions that may not be allowed and are specified in the assignment's autograder Piazza post.

Refer to this problem set's autograder post for a list of banned function calls.

Please do not use absolute paths in your submission code. All paths should be relative to the submission directory. Any submissions with absolute paths are in danger of receiving a penalty!

Obtaining the Starter Files:

```
Obtain the starter code from canvas under files.
```

Programming Instructions

Your main programming task is to complete the api described in the file **ps2.py**. The driver program **experiment.py** helps to illustrate the intended use and will output images to verify your results. Additionally there is a file **ps2_test.py** that you can use to test your implementation.

Write-up Instructions

Create **ps2_report.pdf** - a PDF file that shows all your output for the problem set, including images labeled appropriately (by filename, e.g. ps1-1-a-1.png) so it is clear which section they are for and the small number of written responses necessary to answer some of the questions (as indicated). For a guide as to how to showcase your results, please refer to the powerpoint template for PS2. Remember to update the template to this term and also to ensure your name and email are correct on the output submitted.

How to submit

Two assignments have been created on Gradescope: one for the report - **PS2_report**, and the other for the code - **PS2_code**.

- **Report:** the report (PDF only) must be submitted to the **PS2_report** assignment.
- **Code: all files** must be submitted to the **PS2_code** assignment. **DO NOT** upload zipped folders or any sub-folders, please **upload each file individually**. Drag and drop all files into Gradescope.

Notes:

- You can only submit to the autograder **10** times in an hour. You'll receive a message like "You have exceeded the number of submissions in the last hour. Please wait for 36.0 mins before you submit again." when you exceed those 10 submissions. You'll also receive a message "You can submit 8 times in the next 53.0 mins" with each submission so that you may keep track of your submissions.
- If you wish to modify the autograder functions, create a copy of those functions and **DO NOT** mess with the original function call.

YOU MUST SUBMIT your report and code separately, i.e., two submissions for the code and the report, respectively. Only your last submission before the deadline will be counted for each of the code and the report.

Grading

The assignment will be graded out of 100 points. The last submission before the time limit will only be considered. The code portion (autograder) represents **60**% of the grade and the report the remaining **40**%.

The images included in your report must be generated using experiment.py. This file should be set to be run as is to verify your results. **Your report grade will be affected if we cannot reproduce your output images.**

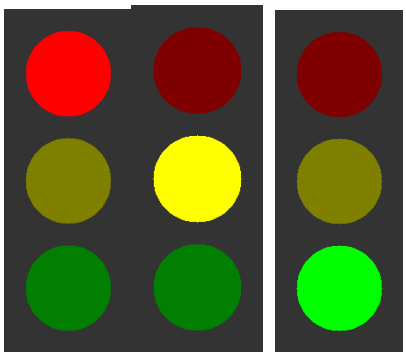
The report grade breakdown is shown in the question heading. As for the code grade, you will be able to see it in the console message you receive when submitting.

Assignment Overview

You have just started working for a self-driving car company. As your first assignment, you are asked about how the car would process traffic rules. That is, you are tasked with detection of both traffic lights and traffic signs. Your job is to design and implement a program that would solve both aims.

1. Traffic Light [25 points]

First off, you are given a generic traffic light to detect from a scene. For the sake of the problem, assume that traffic lights are shown as below: (with red, yellow, and green) lights that are vertically stacked. You may also assume that there is no occlusion of the traffic light.

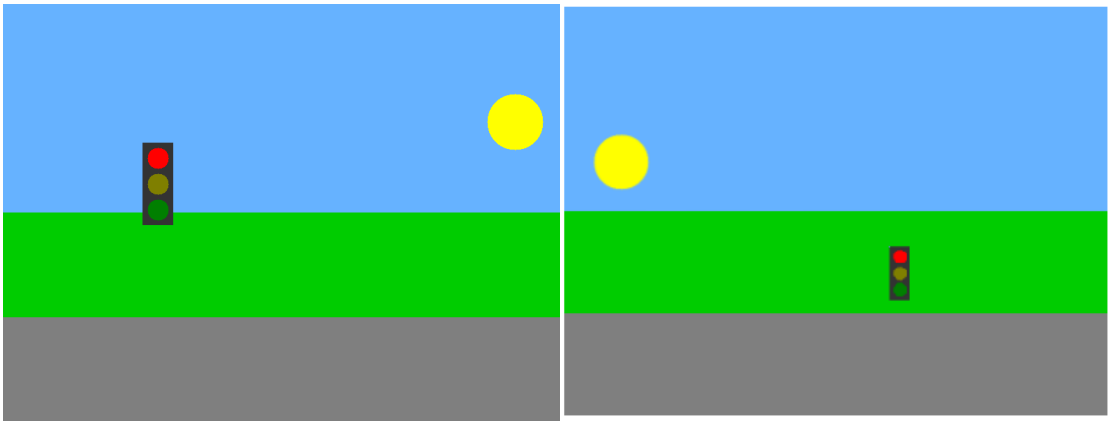


((x-coordinate, y-coordinate), 'color')

It is your goal to find a way to determine the state of each traffic light and position in a scene. Position is measured from the center of the traffic light. Given that this image presents symmetry, the position of the traffic light matches the center of the yellow circle.

Complete your python ps2.py such that traffic_light_detection returns the traffic light center coordinates (x, y) ie (col, row) and the color of the light that is activated ('red', 'yellow', or 'green'). Read the function description for more details.

Testing:
A traffic light scene that we will test will be randomly generated, like in the following pictures and examples in the github repo.



Functional assumptions:
For the sake of simplicity, we are using a basic color scheme, but assume that the scene may have different color objects and backgrounds [relevant for part 2 and 3]. The shape of the traffic light will not change, nor will the size of the individual lights relative to the traffic light. Size range of the lights can be reasonably expected to be between 10-30 pixels in radius. There will only be one traffic light per scene, but its size and location will be generated at random (that is, a traffic light could appear in the sky or in the road--no assumptions should be made as to its logical position). While the traffic light will not be occluded, the objects in the background may be.

Code: Complete `traffic_light_detection(img_in, radii_range)`

Report: For each of the following images, find the traffic light. Place its coordinates and state using cv2.putText before saving your output images.

- Input: **simple_tl.png**. Output: **ps2-1-a-1.png**
- Input: **scene_tl_1.png**. Output: **ps2-1-a-2.png**
- Input: **scene_tl_2.png**. Output: **ps2-1-a-3.png**
- Input: **scene_tl_3.png**. Output: **ps2-1-a-4.png**

2. Traffic Signs one per scene [50 points]

Now that you have detected a basic traffic light, see if you can detect road signs. Below are five common road signs that you would see in the United States (apologies to those outside the United States)

Implement a way to recognize these signs:

Similar to the traffic light, you are tasked with detecting the sign in a scene and finding the (x, y) i.e (col, row) coordinates that represent the **polygon's centroid**.

Functional assumptions:
Like above, assume that the scene may have different color objects and backgrounds. The size and location of the traffic sign will be generated at random. While the traffic signs will not be occluded, objects in the background may be.

Code: Complete the following functions. Read their documentation in ps2.py for more details.

- `yield_sign_detection(img_in)`

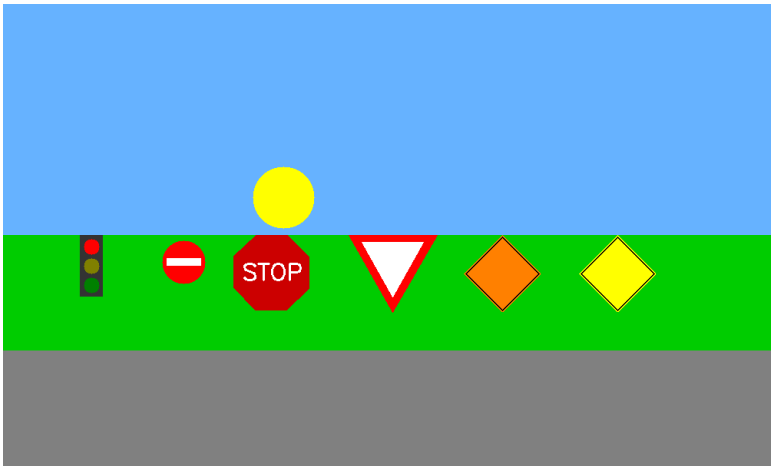
- `stop_sign_detection(img_in)`
- `warning_sign_detection(img_in)`
- `construction_sign_detection(img_in)`
- `do_not_enter_sign_detection(img_in)`

Report: For each of the following images, find the traffic sign. Place its coordinates using cv2.putText before saving your output images.

- Input: **scene_dne_1.png.** Output: **ps2-2-a-1.png**
- Input: **scene_stp_1.png.** Output: **ps2-2-a-2.png**
- Input: **scene_constr_1.png.** Output: **ps2-2-a-3.png**
- Input: **scene_wrng_1.png.** Output: **ps2-2-a-4.png**
- Input: **scene_yld_1.png.** Output: **ps2-2-a-5.png**

3. Multiple signs in a scene [10 points]

The next task is to detect multiple traffic signs and the traffic light in one scene. Find where each sign is in the scene, below is a randomly generated example:



Functional assumptions:
Like above, assume that the scene may have different color objects and backgrounds. There will be n instances of each sign and/or traffic light, where n is 0 or 1. The size and location of each will be generated at random. While the traffic signs will not be occluded, objects in the background may be.

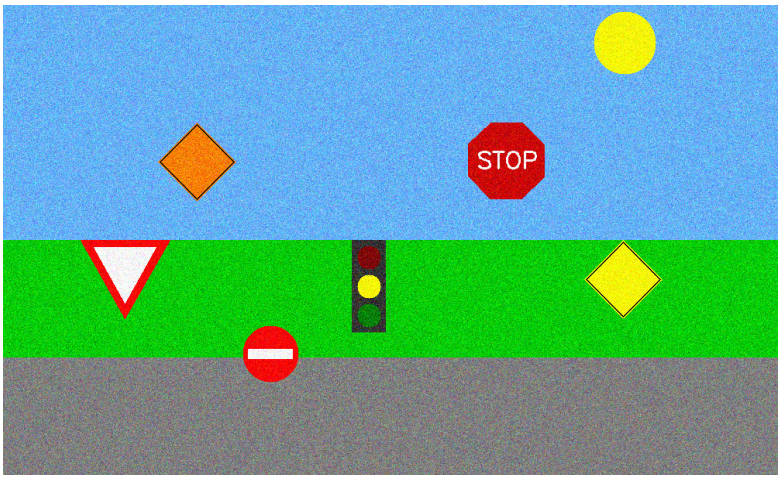
Code: Complete `traffic_sign_detection(img_in)`.

Report: Find all traffic signs and the traffic light using the images below. Place their coordinates and name (in a easy to read area) using cv2.putText before saving your output images.

- Input: **scene_some_signs.png.** Output: **ps2-3-a-1.png**
- Input: **scene_all_signs.png**. Output: **ps2-3-a-2.png**

4. Poor Weather Conditions [10 points]

Anyone with driving experience knows that signs and lights may be harder to detect in times of inclement weather. Though, as a CV student, you are unphased. Your program will be given a scene like below. Test your traffic light and sign detection against images that have a certain amount of noise.



Code: Complete `traffic_sign_detection_noisy(img_in)`.

Report: Find all traffic signs and the traffic light using the images below. Place their coordinates and name using cv2.putText before saving your output images.

- Input: **scene_some_signs_noisy.png.** Output: **ps2-4-a-1.png**
- Input: **scene_all_signs_noisy.png**. Output: **ps2-4-a-2.png**

5. Challenge problem: Using a real image [5 points]

Let's try your code using real world images. In this section you are encouraged to go out and take some photographs of street signs like the ones above. See how well your method performs with these. In case you are not able to obtain such images, you should look for these images online. You are to use two type of images:

A. Three images with one street sign in the scene. Label them **img-5-a-1.png, img-5-a-2.png, img-5-a-3.png.** (Will be collected with your submission)
B. Three images with multiple street signs. Label them **img-5-b-1.png, img-5-b-2.png, img-5-b-3.png.** (Will be collected with your submission)

Find all traffic signs using the images below. Place their coordinates and name using cv2.putText before saving your output images.

Code: Ideally you would use the same functions in ps2.py you worked on. Adding other methods is allowed but make sure you include them in ps2.py and experiment.py where needed.
Complete `traffic_sign_detection_challenge(img_in).`

Report:
Create and show your output images and name them the following way:

- **ps2-5-a-1.png, ps2-5-a-2.png, ps2-5-a-3.png** (output images from part 5a)
- **ps2-5-b-1.png, ps2-5-b-2.png, ps2-5-b-3.png** (output images from part 5b)

Text answer: 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? Include your answer in the report.