CS 4476/6476: Computer Vision, Spring 2020
PS4

Instructor: Judy Hoffman

Due: Friday, March 13, 11:58 pm

1 Instructions

General Instructions:

Set up the environment as you did for the previous assignments.

1. Install Miniconda. It doesn't matter whether you use 2.7 or 3.6 because we will create our own environment anyway.
2. Create a conda environment, using the appropriate command. On Windows, open the installed "Conda prompt" to run this command. On MacOS and Linux, you can just use a terminal window to run the command. Modify the command based on your OS ('linux', 'mac', or 'win'):

   ```
   conda env create -f proj4_env_<OS>.yml
   ```

3. This should create an environment named 'proj3'. Activate it using the following Windows command:

   ```
   activate proj4
   ```

   or the following MacOS / Linux command:

   ```
   source activate proj4
   ```

4. Install the project package, by running `pip install -e .` inside the repo folder.
5. Run the notebook for color quantization using:

   ```
   jupyter notebook ./Quantization-Student.ipynb
   ```

   and for Hough Transform using:

   ```
   jupyter notebook ./notebook.ipynb
   ```

6. Generate the submission once you've finished the project using:

   ```
   python zip_submission.py
   ```

Points Distribution:

1. You can score a maximum of 100 points on this assignment of which 75 points are regular credit for CS 4476 and 85 points are regular credit for CS 6476.
2. Thus, students in CS 6476 must complete at least 10 points worth of the extra credit in addition to Sections 2 and 3.

Instructions for Hough Transform

You can code directly in the notebook. All submissions will be via Gradescope. If you're completing this assignment in Jupyter Notebook, you must run the `notebook2script.py` file to export your work to a python file. To generate your submission file, run the command `python notebook2script.py submission` and your file will be created under the 'submission' directory.

2 Color Quantization with K-Means (25 points)

For this problem you will write code to quantize an image using k-means clustering and experiment with two different color spaces — RGB and HSV.

Complete the functions defined below in `quantization_student.py`. Use `Qunatization-Student.ipynb` to run your implemented functions and generate quantized images with 3, 5 and 10 clusters. Put these images in your report.

1. (5 points) Given an RGB image, quantize the 3-dimensional RGB space, and map each pixel in the input image to its nearest k-means center. That is, replace the RGB value at each pixel with its nearest cluster's average RGB value. Use the following form:

   ```
   [outputImg, clusterCenterColors] = quantizeRGB(origImg, k)
   ```

   where `origImg` and `outputImg` are MxNx3 matrices of type `uint8`, `k` specifies the number of colors to quantize to, and `clusterCenterColors` is a k x 3 array of the k centers. (You can use the Scikit-Learn implementation of the k-means algorithm for this question.)

2. (5 points) Given an RGB image, convert it to HSV, and quantize the 1-dimensional **Hue** space. Map each pixel in the input image to its nearest quantized Hue value, while keeping its Saturation and Value channels the same as the input. Convert the quantized output back to RGB color space. Use the following form:

   ```
   [outputImg, clusterCenterHues] = quantizeHSV(origImg, k)
   ```

   where `origImg` and `outputImg` are MxNx3 matrices of type `uint8`, `k` specifies the number of clusters, and `clusterCenterHues` is a k x 1 vector of the hue centers. (You can use the Scikit-Learn implementation of the k-means algorithm for this question.)

3. (3 points) Write a function to compute the SSD error (sum of squared error) between the original RGB pixel values and the quantized values, with the following form:

   ```
   [error] = computeQuantizationError(origImg, quantizedImg)
   ```

   where `origImg` and `quantizedImg` are both RGB images, and error is a scalar giving the total SSD error across the image. Write down the logarithmic error (base e) for all the generated RGB and HSV images (3, 5 and 10 clusters) in your report. Note: The function should **not** return logarithmic error.

4. (12 points) Answer the following questions in your report. Please try to restrict your answers to 1-2 sentences of fairly high-level explanations.

   (a) How do the results vary with the number of quantization bins?

   (b) How are the **qualitative** results between RGB and HSV color spaces different? How would you explain this difference?

   (c) Name and explain 2 metrics (apart from SSD) used for comparing the generated/modified image(here the modified image is the quantized image) with the original one.

3 Hough Transform on generated images (50 points)

Detecting Traffic Signs and Lights

Learning Objectives

— 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 and noise in an image.

— Identify what challenges real-world images present over simulated scenes.

Note: For all sub-parts in this section, be sure to include the images generated in the IPython notebooks in the corresponding sections in your report.

Your question starts here-

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 the objectives.

1. Traffic Light [15 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.

Figure 1: Image of generic traffic light (Left: In order - red, yellow, green light is ON)

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 notebook.ipynb` such that `traffic_light_detection` returns the traffic light center coordinates (x, y) i.e. (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

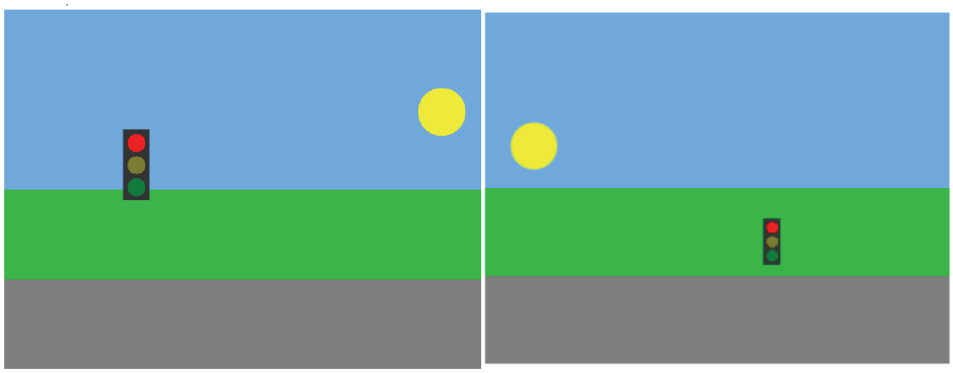

Figure 2: Example of generated scene

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.

Deliverables:

(a) Code:
Complete `traffic_light_detection(img_in, radii_range)`

2. Traffic Signs one per scene [25 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. Implement a way to recognize these signs:

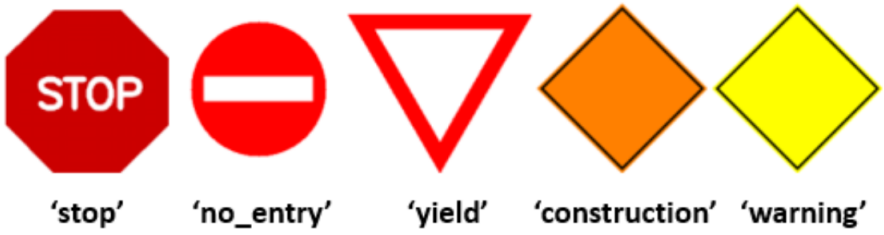

Figure 3: Traffic Signs

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.

Deliverables:

(a) Code:
Complete the following functions. Read their documentation in `notebook.ipynb` 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)`

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, Fig. 4 is a randomly generated example:

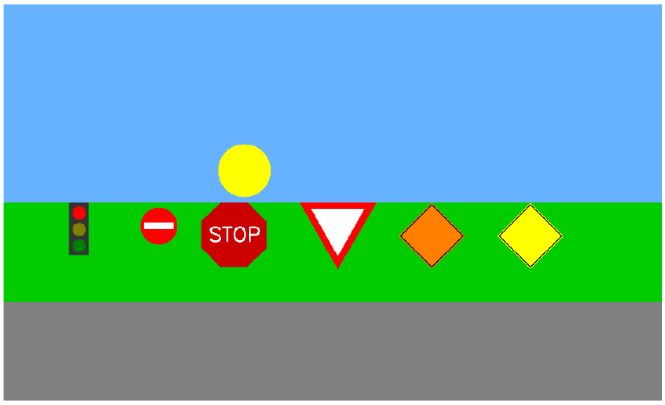

Figure 4: Generated Scene with Signs

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.

Deliverables:

(a) Code:
Complete `traffic_sign_detection(img_in)`.

4 Extra Credit (15 + 10 points)

4.1 Hough Transform on Real Images with known radius (10 points code + 5 points report

Let's try your code using real world images. Implement a Hough Transform circle detector that takes an input image and a fixed radius, and returns the centers of any detected circles of about that size and the Hough space used for finding the centers. Include a function with the following form:

```
[centers,hough_space] = detectCircles(im, radius, useGradient)
```

where `im` is the input image, radius specifies the size of circle we are looking for, and `useGradient` is a flag that allows the user to optionally exploit the gradient direction measured at the edge points.

The output `centers` is an N x 2 matrix in which each row lists the (x,y) position of a detected circle's center and `hough_space` is a `NumPy` array (height x width matrix) of Hough accumulator array (height and width from the image). Save this function in a file called `detectCircles.py`.

Answer the following in your report:

In Circle Hough Transform, circle candidates are produces by voting in the parameters space and then finding local maxima in the space. This can be done using techniques like Thresholding, Non-Maximum Suppression and Mean Shift.

Experiment with thresholding as a post-processing technique to determine circle parameters from the accumulator array. In your report include images of the selected circles along with the corresponding Hough Space for low (≤ 0.4), mid-range (≈ 0.7) and high (≥ 0.95) thresholds. Explain how your results vary with increasing thresholds and why that may be the case.

4.2 Hough Transform on Real Image with unknown radii (10 points)

Extend your Hough circle detector implementation to detect circles of any radius. Optimize your code for the image `jupiter.jpg`. You would be awarded points proportional to the number of circles detected. Detecting all of the circles gives you the maximum credit. Be sure to include the results generated by the notebook in your report.

Note: We will be testing your code on a hidden image which will **not** have circles of the same radii as `jupiter.jpg`. However, the hidden image is similar `jupiter.jpg` and the circles will be in the same range (approximately 15 to 200 pixels). Therefore your parameters tuned for `jupiter.jpg`, should work reasonably well for the held-out image as well.