Assignment 05

Please refer to MyCourses/Gradescope for assignment deadlines.

The goal of this assignment is to give you hands-on practice with the following concepts:

- Canny edge detection
- Hough transform for line detection
- Hough transform for circle detection

There is a lot here, so each part has a large portion done in the provided code. There are just a few key parts for you to implement.

Part 1 of 3: Canny edge detection

As we saw in class, the main steps of the Canny algorithm are:

1. Conversion to a single-channel image (usually just grayscale)
2. Gaussian blurring
3. Gradient calculation with the Sobel operator
4. Conversion to magnitude and direction
5. Direction-dependent non-maximal suppression
6. Double thresholding to find strong, weak, and non-edges
7. Hysteresis: promoting weak edges to strong edges if they are connected to strong edges

In practice, you would generally use `cv.Canny()` to do this. If you look at the OpenCV documentation, you'll see that there are two overloaded signatures for `cv.Canny()`: one that takes in a single `uint8` image (the result of blurring in step 2), and one that takes in two `int16` images `dx` and `dy` (the result of the Sobel operators in step 3). The latter gives you a bit more control over how you compute the gradient.

In this part of the assignment, you will implement steps (2) and (3) above using a single derivative-of-Gaussians filter. You will then pass the results to the `dx` and `dy` arguments of `cv.Canny()` and confirm that it gives you the same result as if you had used `cv.Canny()` with the blurred single-channel image. (In practice, there will be a few minor differences in the outputs due again to our old friend, floating-point precision and rounding. Arguably, using the DoG filter is more accurate because it avoids the intermediate step of converting to `uint8` in between the blur operation and the gradient calculation.)

Instructions: Implement the missing code in `edge_detection.py`. Verify that you get the same (or nearly the same, considering possible rounding errors) whether using `my_canny_1` or `my_canny_2`.

The `bricks.jpg` file has been provided as a test image for this part of the assignment.

Part 2 of 3: Hough transform for line detection

The `hough_lines.py` file contains a large python class that shows the inner-workings of the Hough Line-Detection algorithm using primarily numpy operations. In practice, you would normally be using `cv.HoughLines()` or `cv.HoughLinesP()` to do this. But, this assignment is about understanding the underlying algorithm.

Instructions:

1. Implement the missing code in `hough_lines.py` according to the docstrings and comments.
2. Tune the parameters of the `main()` function such that your code detects as many of the **horizontal lines** in the `house.jpg` image as possible, without detecting any of the other lines. You will need to look carefully at the code and use your debugger to determine what the effect of changing each parameter is. Using the `config.py` file, record the parameters that you used.

Suggestions:

- Use your debugger and set breakpoints in `hough_lines.py`, but run the code using `run_hough.py` as the entry point. This will automatically apply the values in the `config.py` file to the house image and save out the result as a new image.
- Think carefully about what each parameter does. For example, if you're detecting too many edges in the tree region of the image, you might want to adjust the canny blur parameter. The output will also be sensitive to the size and resolution of the accumulator array and how this interacts with the chosen parameters for non-maximal suppression.

Expected Output:

Figure 1: Example output having tuned the parameters for horizontal-line detection

Part 3 of 4: Hough transform for circle detection

Just like in Part 2, you will be doing a small amount of coding and a lot of parameter tuning.

Instructions:

1. Implement the missing code in `hough_circles.py` according to the docstrings and comments.
2. Tune the parameters of the `main()` function separately for each of the 4 types of coins. As before, use your debugger to better understand the inner-workings of the circle detector and use the `config.py` file to record the parameters that you used.

Suggestions: While it's unavoidable that you'll need to adjust the 'radius' parameter for each coin, see if you can find values for all the other parameters that work for all coins. This isn't strictly necessary, but it's nice for your sanity (and ours) if the logic is as repeatable as possible.

Example Outputs:

Figure 2: Example output having tuned the parameters for pennies

Figure 3: Example output having tuned the parameters for nickels

Figure 4: Example output having tuned the parameters for dimes

Figure 5: Example output having tuned the parameters for quarters

Collaboration and Generative AI disclosure

Did you collaborate with anyone? Did you use any Generative AI tools? Briefly explain what you did in the `collaboration-disclosure.txt` file.

Reflection on learning objectives

This is optional to disclose, but it helps us improve the course if you can give feedback.

- what did you take away from this assignment?
- what did you spend the most time on?
- about how much time did you spend total?
- what was easier or harder than expected?
- how could class time have better prepared you for this assignment?

Give your reflections in the `reflection.txt` file if you choose to give us feedback.

Submitting

Use `make submission.zip` to generate a zip file that you can upload to Gradescope. You can upload as many times as you like before the deadline. To account for late penalties, contact the instructor or grader directly if you plan on uploading any further revisions after the deadline. (This is to prevent us from having to grade your work twice. Let us know so that we don't start grading files that will be replaced later).