EECS 504 Foundations of Computer Vision: HW1

Term: Fall 2018
Instructor: Jason J. Corso, EECS, University of Michigan
Due Date: 10/5 23:59 Eastern Time

Constraints: This assignment may be discussed with other students in the course but must be written independently. Programming assignments should be written in Python using the open-source library ETA. Over-the-shoulder Python coding is strictly prohibited. **Web/Google-searching for background material is permitted. However, everything you need to solve these problems is presented in the course notes and background materials, which have been provided already.**

Goals: Test mathematical basics and deepen the understanding of images as functions.

Data: You need to download `hw1.zip` to complete this assignment. All paths in the assignment assume the data is off the local directory.

Problem 1 (20): Estimate homography with Linear Least Squares
We discussed least squares estimation in lecture. As you may know, affine transformation preserves parallelism. For example, a square could become a parallelogram after applying an affine transformation, but it can never be a trapezoid. Now, we consider a more general case where only straight lines are preserved in the transformed space, while other geometrical properties are not guaranteed. This type of transformation is called a projective transformation, or a homography. The transformation can be described mathematically as the following:

$$
w \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \tag{1}
$$

where $\mathbf{p} = [x, y, 1]$ and $\mathbf{p}' = [x', y', 1]$ are homogeneous coordinates of the original and the transformed point respectively. w is a constant to make sure the third element of $\mathbf{p}'$ is 1.

(a) (6) Assume we know a list of pairs of points under homography, $\{(x_1, y_1), (x'_1, y'_1)\}, \{(x_2, y_2), (x'_2, y'_2)\}, ..., \{(x_n, y_n), (x'_n, y'_n)\}$. Derive the least-squares formulation for estimating the parameters of the homograph.

$$
\min \|\mathbf{A}\mathbf{x} - \mathbf{y}\|_2^2. \tag{2}
$$

Write down the values of $\mathbf{A}$,$\mathbf{x}$ and $\mathbf{y}$, given the points above. Also include your derivation in the writeup.

(b) (4) What is the solution to our estimation problem? How many pairs of points do you think are needed to ensure that the solution is unique? Write down your answer and include a brief explanation in your writeup.

(c) (10) Projective transformations are very common in our real world. When you take pictures of the same object from different viewing angles, the objects in the pictures are projections of the real one. Based on this, we can make useful applications. For example, American football games are played on a rectangular field marked with end lines, side lines and goal lines which are 10 yards inward from each end line, as shown in the Fig. 1. In the NFL broadcasts, virtual yellow lines are drawn on top of real scenes to create a better view for TV audience (see more in How the NFL's magic yellow line works). In this problem, you are expected to implement this based on the homography estimation you derived in previous questions.

To simplify the problem, you will work on two images instead of a video. `football1.jpg` and `football2.jpg` are two screenshots captured from a football game and are provided to you. We use the first image as a baseline (though it is not a good one). The yellow line for the marker 33 has been highlighted, as shown in Fig. 2. You need to select a couple of correspondences in two images first. Then, estimate the transformation matrix and draw the corresponding line in the second picture.

Fill in the code in `hw1p1.py`. The code and images are given in the folder `hw1p1`. Include your code as verbatim and report the output image in your pdf writeup. Also submit your original program files on Canvas.
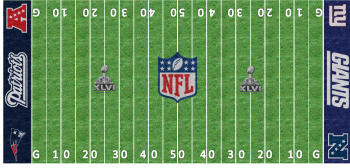
Figure 1: football field

Figure 2: yellow line at marker 33

Problem 2 (6): Lambertian Model

The Lambertian model of reflectance is $R(\mathrm{x}) = \rho \boldsymbol{\ell}(\mathrm{x})^{\mathrm{T}} \mathbf{n}(\mathrm{x})$, as shown in Fig. 3.
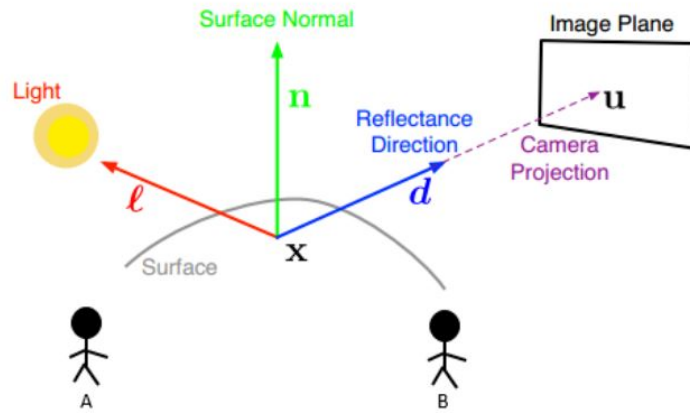

Figure 3: Lambertian model

(a) (3) There are two people standing at location A and B. Who will observe stronger reflection at point x? Why?

(b) (3) Tell one drawback of Lambertian model and name a specific object in the real world whose reflectance is not well modeled by the Lambertian model.

Problem 3 (15): Potts model

(a) (3) Convolution is a simple mathematical operation which is fundamental to many common image processing operations like blurring, edge-detection etc. Discrete convolution in 1-D is given by:

$$y[n] = x[n] * h[n] = \sum_{m=-\infty}^{\infty} x[m] \cdot h[n-m] \tag{3}$$

Think how would you perform convolution on the boundary values. Perform a 2-D convolution on the following image **A** with kernel **B**:

$$\mathbf{A} = \begin{bmatrix} 4 & 8 \\ 2 & 3 \\ 1 & 6 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 1 & 2 \\ 2 & 0 \end{bmatrix} \tag{4}$$

(b) (6) Recall the Potts model we discussed in the lecture. Based on that model, the energy of an image I is

$$E(I) = \beta \sum_{s=1}^{n-1} \sum_{t=1}^{n} (\mathbf{1}(I(s,t) \neq I(s+1,t))) + \beta \sum_{s=1}^{n} \sum_{t=1}^{n-1} (\mathbf{1}(I(s,t) \neq I(s,t+1))) \tag{5}$$

Implement a function to compute the energy of the image `data/Michigan_letter.jpg` using this model and submit the value of the resultant energy. Use $\beta = 1$ in your implementation. You need to fill in the missing parts of the functions `_create_x_derivative_kernel()`, `_create_y_derivative_kernel()` and `_convolve()` in `modules/convolution.py` and `_calculate_potts_energy()` in `modules/_calculate_potts_energy.py`.

Compute the complexity O(n) of this algorithm.

You can run the the pipeline using the following command:
```
eta build -r requests/potts_energy_request.json --run-now
```

(c) (3) Perform a 2-D convolution on the given image using the _create_gaussian_kernel() in modules/convolution.py and calculate its Potts energy. You need to use the pipeline provided in pipelines/potts_energy_gaussian.json to do this. Explain the difference in energies for the original and final image. (Hint: Think in the direction as to what the given kernel is doing to the image).

You can run the the pipeline using the following command:
```
eta build -r requests/potts_energy_gaussian_request.json --run-now
```

(d) (3) You are given two images data/img_1.jpg and data/img_2.jpg. You need to modify the requests/potts_energy_request.json to input these images. Compute the energies of these images using the function you implemented in part (b). How are the energies of the two images related? Explain your observations.
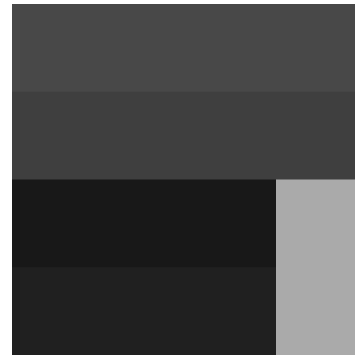

Figure 4: img_1.jpg

Figure 5: img_2.jpg

Problem 4 (14): Transformation

Imagine you are in Lego world. You are given 3 blocks - red, blue and green. The blocks are arranged as shown in Fig 6. You have to rearrange and stack the lego blocks to form structure shown in Fig 7. The images are 20x20 and the top-left corner is (0,0). The edge-length (in number of pixels) of the blocks are provided in the figure.
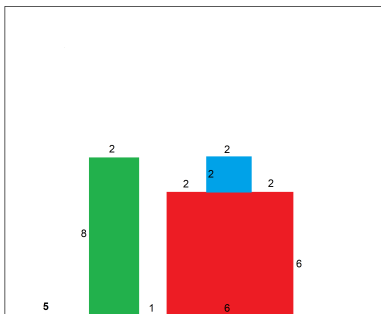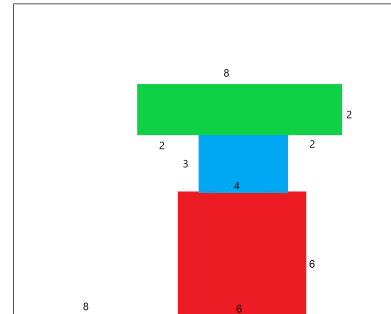

Figure 6: Current state

Figure 7: Final state

(a) (10) Explain your approach in detail. Provide the transformation matrix (matrices) you have created to achieve Fig 7.

(b) (4) State which transformation(s) is (are) used in part (a). Do these transformation matrix(matrices) preserve orientation, angle and parallelism of line segments?

Problem 5 (25): Edge Detection

An edge is a place where image intensity changes abruptly. Edge detection is the method of finding these boundaries of objects within images. Through this problem, you are going to implement a well-known algorithm called Canny Edge Detector. You need to run the detector on `data/Canny_input.jpg` by using `pipelines/edge_detection.json`.

You can run the the pipeline using the following command:
```
eta build -r requests/edge_detection_request.json --run-now
```

The first step of the Canny Edge Detector is to convolve the input image with a Gaussian kernel. This removes unwanted high frequency components (noise) in the image that could be falsely detected as an edge.

This is the first step in the pipeline, and is already given to you. However, the value of sigma is currently set to 0.001. This value is so small that Gaussian convolution does nothing and the result of such convolution is the original image. We want you to tune the value of sigma to obtain the best result from your Canny Edge Detector. Try increasing and decreasing sigma, and observe its effect on the output. Then, choose the value of sigma that gives you the best result from your edge detector.

(a) (4) The second step of the Canny Edge Detector is to get the gradient images. One of the most popular ways to compute the gradient images is by using vertical and horizontal sobel operators. Fill out the missing part of the functions `_create_sobel_horizontal_kernel()` and `_create_sobel_vertical_kernel()` in file `modules/convolution.py` to create the kernels, and function `_create_intensity_orientation_matrices()` in file `modules/canny_edge_detector.py` to compute the intensity and orientation matrices of the gradients.
Report the gradient intensity image in your writeup. Observe the edges extracted from this. Does it give a sharp edge? Explain your observation.

(b) (7) Implement an algorithm to get a sharp edge response from the output of part (a). This process is called Non-Maximum Suppression. Fill in the missing part of the function `_non_maximum_suppression()` in the file `modules/canny_edge_detector.py` and report the output image in your writeup.
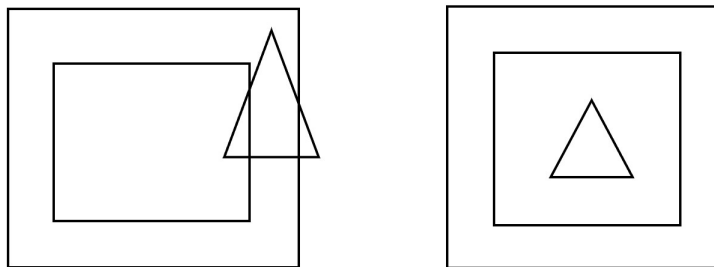

Figure 8: Canny_input.jpg

(c) (10) Observe the output from part (b). The edge-pixels can be considered as strong, weak, or suppressed based on the intensity of their gradients. You should then perform Edge Tracking by connecting each weak-edged pixel to any neighboring strong-edged pixel (if any) to get the complete edge. Fill in the missing part of the code for the functions `_double_thresholding()` and `_hysteresis()` in the file `modules/canny_edge_detector.py` to implement double thresholding and perform edge tracking by hysteresis, respectively. Include the output image in your report.

(d) (4) Is the Canny Edge Detector rotation invariant? Provide a mathematical proof or counter argument.

(e) (**Extra Credits: 10**) Observe the line-segments formed by intersecting edges in the output of part (c). Write a function to get the coordinates of these line-segments. Be creative! For this problem, you can use `pipelines/line_segments.json` and modify `modules/find_line_segments.py`. You can run the the pipeline using the following command:

```
eta build -r requests/line_segments_request.json --run-now
```

You should provide the output as a .json file called `output_points.json` (You can use `eta.core.serial.write_json()` to write into .json format). Your `output_points.json` should be in the follow-ing format:

```
{
    "Line_segments": {
                        {
                            "No": 1
                            "coordinates": [(x1, y1), (x1',y1')]
                        }
                        {
                            "No": 2
                            "coordinates": [(x2, y2), (x2',y2')]
                        }
                        .
                        .
                        .
    }
```

Submission Process: Submit a single pdf with your answers to these problems, including all plots and discussion. Submit the pdf to Gradescope.

For coding assignments, include your code verbatim in your writeup. Pack the original program files into one zip file and upload it to Canvas. **Code should be well-commented for grading**

Grading and Evaluation: The credit for each problem in this set is given in parentheses at the stated question (sub-question fraction of points is also given at the sub-questions). Partial credit will be given for both paper and python questions. For python questions, if the code does not run, then limited or no credit will be given.