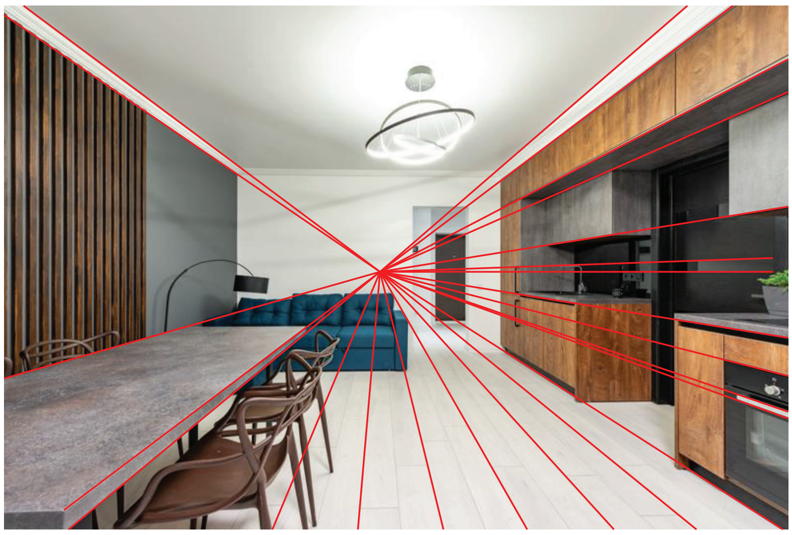
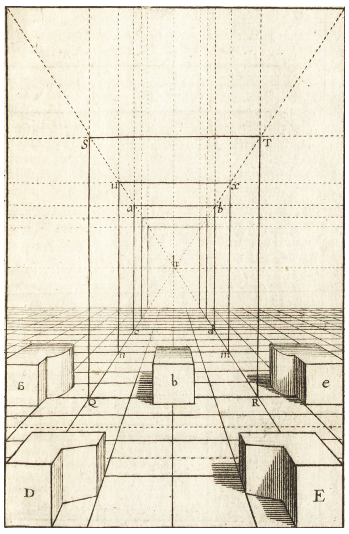
The University of Melbourne
School of Computing and Information Systems
COMP90086 Computer Vision, 2021 Semester 2

Assignment 3: Vanishing point detection

Due: 7pm, 8 Oct 2021
Submission: Source code (in Jupyter Notebook) and written responses (as .pdf)
Marks: The assignment will be marked out of 8 points, and will contribute 8% of your total mark.

In a photograph, objects farther away from the camera appear smaller in the image, and parallel lines extending into the distance appear to converge to a point. This convergence point is called a "vanishing point." This assignment will focus on scenes with a single vanishing point (one-point perspective images), such as the examples shown in Figure 1. In these scenes, lines that are parallel to the camera's viewing axis appear to converge at a single vanishing point.


(a) A photograph with one-point perspective. Parallel lines (red) formed by structures such as the edges of table and the tiles on the floor appear to converge at a single point, the vanishing point.



(b) Artist's illustration of one-point perspective, from Bosse (1665)

Figure 1: Images with one-point perspective

In this assignment, you will write a method to detect the vanishing point in one-point perspective images. Your method should follow the steps below:

1. Detect lines in the image [3 pt]

Use the Canny edge detector and Hough transform to detect lines in the image. Note that the goal of this step is not to find *all* of the lines in the image, but to find lines that can be used to detect the vanishing point (i.e., lines that converge at the vanishing point). Therefore, you will need to do some experimentation with the Canny and Hough parameters to find the most useful lines. OpenCV has two different implementations of the Hough transform (`HoughLines()` and `HoughLinesP()`); you may use whichever function seems more suitable for your implementation.

Not all lines in an image will converge at the vanishing point – vertical lines, horizontal lines, and lines produced by texture regions generally do not converge at the vanishing point. You may wish to exclude these lines in order to better detect the vanishing point.

Your written report should explain your line-detection approach, including the parameters for Canny and Hough and any additional steps you used to exclude lines, with justifications. Include images to illustrate your approach.

2. Locate the vanishing point [3 pt]

Use RANSAC to locate the vanishing point in the image from the detected lines. **You should write your own implementation of RANSAC.** Your implementation should include:

1. [1 pt] Two functions required by RANSAC: a function to find the point where lines intersect, and a function to compute the distance from a point to a line. Code for the latter function was provided in the workshops and you may re-use that code (with citation) in your implementation.
2. [2 pt] A main RANSAC loop, which should randomly sample the minimum number of lines needed to compute an intersection point and compute the number of lines in the image which pass through the intersection point ("inliers"). This function should iterate until it meets a stopping criterion and then return the intersection point with the highest number of inliers – this is the vanishing point.

You will need to find suitable values for RANSAC parameters such as the distance threshold for counting a line as an "inlier" and the stopping criterion. Your written report should explain your RANSAC method and justify your choice of parameters. Include images to illustrate your approach.

3. Main function and evaluation [2 pt]

$\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$

Your submission should include a main function that can run your vanishing point detection method on a folder of images, return the (x,y) locations of the vanishing points, and evaluate the result. A folder of images and ground truth vanishing point locations have been provided, which you can use to develop and evaluate your result. Use mean squared Euclidean distance between the predicted and ground truth vanishing point location as a measure of accuracy. You are not required to split the provided dataset into a training and test set. $\frac{1}{n}\sum_{n}[(x_2-x_1)^2+(y_2-y_1)^2]$ ← n個 sample

Your written report should present the results of your method on the provided images. You should critically evaluate your method, discussing strengths, weaknesses, and any opportunities for improvement. Include images, figures, and/or tables to illustrate your results.

Submission

You should make two submissions on the LMS: your code and a short written report explaining your method and results. Your report on each section of the code (1, 2, and 3) should be no more than 500 words per section.

Submission will be made via the Canvas LMS. Please submit your code and written report separately under the **Assignment 3: Code** and the **Assignment 3: Report** links on Canvas.

- Your **code** submission should include the Jupyter Notebook (please use the provided template) with your code and any additional files we will need to run your code. You do not need to include the provided images or ground truth file in your submission.
- Your written **report** should be a .pdf with your answers to each of the questions. The report should address the questions posed in this assignment and include any images, diagrams, or tables required by the question.

Evaluation

Your submission will be marked on the correctness of your code/method, including the quality and efficiency of your code. You should use built-in Python functions where appropriate and use descriptive variable names. Your written report should clearly explain your approach and any experimentation used to produce your results, and include all of the specific outputs required by the question (e.g., images, diagrams, tables, or responses to sub-questions).

Late submission

ok

The submission mechanism will stay open for one week after the submission deadline. Late submissions will be penalised at 10% of the total possible mark per 24-hour period after the original deadline. Submissions will be closed 7 days (168 hours) after the published assignment deadline, and no further submissions will be accepted after this point.

Updates to the assignment specifications

ok

If any changes or clarifications are made to the project specification, these will be posted on the LMS.

Academic misconduct

ok

You are welcome — indeed encouraged — to collaborate with your peers in terms of the conceptualisation and framing of the problem. For example, we encourage you to discuss what the assignment specification is asking you to do, or what you would need to implement to be able to respond to a question.

However, sharing materials — for example, showing other students your code or colluding in writing responses to questions — or plagiarising existing code or material will be considered cheating. Your submission must be your own original, individual work. We will invoke University's Academic Misconduct policy (`http://academichonesty.unimelb.edu.au/policy.html`) where inappropriate levels of plagiarism or collusion are deemed to have taken place.