COMPUTER VISION REPORT - LAB 3

S6019595 Sebastian Jitaru, S4718267 Vlad Muscoi

Introduction

To get a clear two-dimensional (*2D*) image projection of a three-dimensional (*3D*) scene each point on the image plane has to receive light from only one ray in the environment. To achieve this pinhole and lenses (simulating eyes) are used, which make all rays pass through a single center of projection before reaching the image plane. Mathematically given a point $P(x_w, y_w, z_w)$ in *3D* space maps to $\big(u, v\big)$ on the *2D* image plane via the relationships

$$u \;=\; f\frac{x_w}{z_w}, \quad v \;=\; f\frac{y_w}{z_w}.$$

Where *focal length* f is the distance between the pinhole (optical center O_c) and the image plane. These equations show how moving an object closer to O_c makes its image larger, while points farther away project closer to the center of the image.

An important question in 3D vision involves *inverting* the projection process: to what extent can we recover an object's orientation or even aspects of the camera's internal parameters from a single view? One method to approach this is by identifying *vanishing points*, where parallel *3D* lines intersect in the *2D* image space. In particular, when 3D edges that are known to be parallel are imaged, their corresponding lines on the image plane converge at a single vanishing point this is due to the fact that the angle between the rays gets smaller and smaller as the field increases in other words, the object is further away causing lines to intersect. By detecting and relating these intersections, we can learn the directions of the lines in 3D. This, allows us to estimate both the focal length f of the camera and the orientation of planes in the scene. For instance, observing a rectangle (with two orthogonal sets of edges) allows us to solve for f using the orthogonality (dot product = 0) constraint on vanishing points, then recover 3D direction vectors for the rectangle's edges and compute the normal vector of the planar patch. The following sections explore this process through the provided exercises,

Exercise 1

This exercise's goal is to get the vanishing points, focal length, f, direction vectors and normal of a planar image space from a single perspective image of a rectangle.

1. **Data Acquisition:** Using Python matplotlib tool, the endpoints of the edges of the rectangle are collected. We gather 2 sets of parallel lines (See Figure 1 for a visual aid of the rectangle with parallel lines on top.)
2. **Line Fitting and Vanishing Points:** For each set, the four clicked points (two per line) are saved in np arrays (e.g. A_1 and B_1 for set 1, and A_2 and B_2 for set 2). Each pair of points is used to fit a line using `np.polyfit`. The intersection of the two best-fit lines gives us the vanishing point for that set. In our implementation, the computed vanishing points are: $(u_1, v_1) \approx (2031.47, 1294.16)$ and $(u_2, v_2) \approx (-1775.30, 1129.03)$. Figure 2 shows the extended lines intersecting at these points.
3. **Computing the Focal Length:** Assuming that the principal point is at the center of the image (in our case, $(u_0, v_0) = (320, 240)$), the focal length is computed using the orthogonality of the two sets of vanishing points. This is based on the fact that the 3D directions corresponding to the rectangle's edges are perpendicular. The focal length is then obtained by the following equation derived from the geometric principles presented in [1]
$$f = \sqrt{-\Big[(u_1 - u_0)(u_2 - u_0) + (v_1 - v_0)(v_2 - v_0)\Big]}.$$
resulting in $f \approx 1627.53$.
4. **Obtaining the Direction Vectors:** Each vanishing point can be back-projected to obtain a 3D direction vector. In the pinhole camera model, a vanishing point (u, v) corresponds to a 3D direction given (up to scale) by
$$(u - u_0,\ v - v_0,\ f).$$
Normalizing this vector results in the unit direction vectors for each set: $\mathbf{w}_1 \approx [0.66173,\ 0.40759,\ 0.62928]$ $\mathbf{w}_2 \approx [-0.74882,\ 0.31772,\ 0.58165]$.
5. **Computing the Normal of the Plane:** Finally, the normal of the planar (i.e. the rectangle) is computed as the cross product of the two direction vectors:
$$\mathbf{n} = \frac{\mathbf{w}_1 \times \mathbf{w}_2}{\|\mathbf{w}_1 \times \mathbf{w}_2\|} \approx [0.03714,\ -0.85611,\ 0.51546].$$

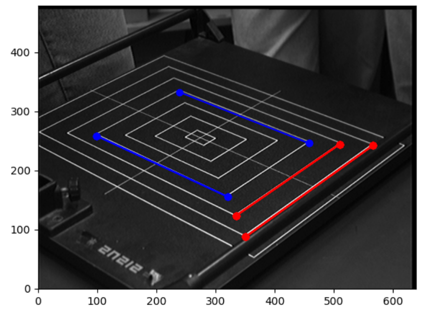

Fig. 1. 2 Sets of parallel lines 1 in blue and 1 in red on top of rectangle.tif image.

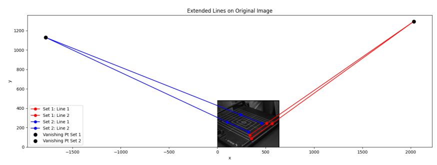

Fig. 2. Extended lines showing the intersections (vanishing points) for each set of parallel lines.

Exercise 2

Figure 3 shows the two sets of parallel lines of which end-points have been extracted.

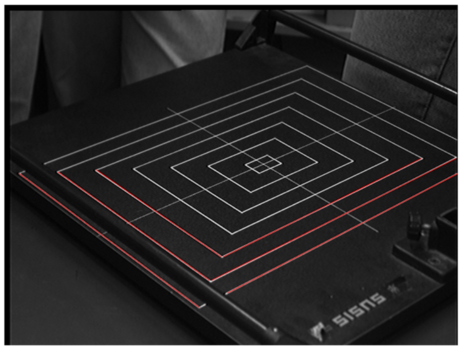

Fig. 3. Selected pairs of end-points

Exercise 3

The function provided in the 'par_lines.m' file is designed to determine the orientation of N parallel 3D lines. The function begin by reading data from a file. This file is structured in four columns, each containing the endpoint coordinates of the parallel lines. The code then sets the following variables and objects:

- n - number of dimensions
- f - a temporary value
- m - number of lines in the input (needs to be higher than the number of dimensions)
- a - a matrix based on the difference of the input end-point's coordinates

Expanding more on the matrix, each row of it represents a line within the image. In this matrix, a system of linear equations will be stored. The rows of the matrix will be used to compute the vanishing point. To do this, we must first compute the line equation which can be written as: $ax + by + c = 0$:

- The variable a in the previous equation is the first column of matrix a which will be populated with the horizontal components: $y_2 - y_1$
- The variable b in the previous equation is the second column of matrix a which will be populated with the vertical components: $-(x_2 - x_1)$
- The variable c in the previous equation is the third column of matrix a which will be populated with the constant term: $(x_1 \cdot a - y_1 \cdot b)$

Once we have the matrix computed, we need to compute the least squared solution by applying Singular Value Decomposition (SVD) to matrix a. This yields a vector with three values which are placed in U, S, V. U and S are unused. The V matrix is already sorted from largest to smallest, so we can extract the minimum of effortlessly. If all our minimums are smaller then 0, we flip their sign, making them positive.

Lastly the vector $wvec$ (normalized) represents the direction or orientation of the parallel lines in the perspective image. It gives the direction in the null space of the matrix a.

Exercise 4

We have modified the `par_line.m` into a new code file called `rectangle.m`. The major change lies in the addition of the function `rectangle(file1, file2)` which accepts two files as input. The flow of the new function is as follows:

1. First, the direction vectors of the two rectangles defined in the two input files are computed. These direction vectors are computed using a slightly adapted version of the `par_line` function.
2. The camera constant f is computed by doing the dot product of the two direction vectors and normalizing the result.
3. We then print the f value and the direction vectors of the sides of the rectangles.
4. The normal of the planar path is computed, normalized and printed.

1. REFERENCES

[1] Dong Hoon Lee, Kyung Jang, and Soon Jung, "Intrinsic camera calibration based on radical center estimation.," 01 2004, pp. 7–13.