Computervision Lab 3
Image formation and transformations

David Van Hamme

1 The pinhole camera model

Essential to understanding image formation is the pinhole camera model, that describes the projection of the world onto the image sensor in the simplest form. It is a projection model from 3D world geometry onto a 2D plane. Rays of light originating (i.e., reflected by) the world geometry pass through an infinitesimally small hole and then intersect with a plane. The coordinates of this point of intersection determine the image coordinates. The process is illustrated in Figure 1.

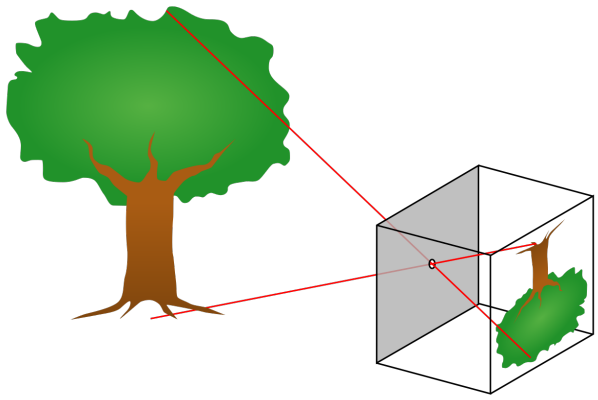

Figure 1: Pinhole camera image formation.

The hole is called the focal point of the projection, and the line through the focal point, perpendicular to the image plane is called the principal axis. The mathematical description of the projection is very straightforward using similar triangles, as shown in Figure 2.

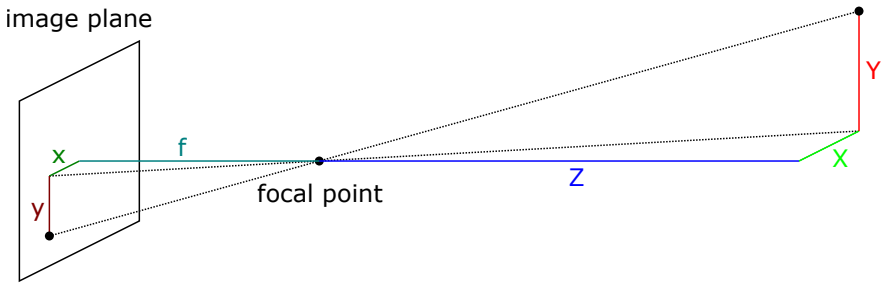

Figure 2: Similar triangles in the pinhole camera model.

As can be seen from the figure, the following relationships hold:

$$x/X = f/Z$$
$$y/Y = f/Z$$

or written differently:

$$x = f\frac{X}{Z}$$
$$y = f\frac{Y}{Z}$$

In an actual camera, the image plane is made up from photosensitive elements that accumulate the incoming light within a rectangular (usually square) region; they are essentially photon counters. These elements are called pixels (short for picture elements), and the images are described in discrete pixel coordinates rather than the continuous x and y expressed in meters in the equations and figure above. Note that because of the similar triangles, the scale and units of the focal distance and image coordinates are not important, only their ratio is. We can therefore express the focal distance f in pixels, e.g., if the focal distance is $5mm$ and the physical size of one photosensitive square on the sensor s $5\mu m$, we can use 1000 as focal distance and the quantities x and y will then be the horizontal and vertical deviation, in pixels, from the principal axis. They now describe pixel coordinates relative to the principal point (the place where the principal axis intersects the image plane). They differ from the image coordinates you use in python to address the images only in a translation: since the origin $(0, 0)$ of the image in python is at the top left corner, we have to add half the horizontal and vertical size of the image (in pixels) to x and y.

The scaling and translation operation, as well as the division by Z are elegantly described in homogeneous coordinates by the following matrix operation:

$$s\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} X \\ Y \\ Z \end{bmatrix}.$$

Note that the actual projection is in the conversion from homogeneous coordinates back to regular coordinates. After multiplying the matrix with the 3D coordinates, the third value is simply the Z value, i.e., the distance to the object. To convert to regular coordinates, you divide by the third value, meaning an object that was twice as far, appears half as large.

The 3x3 matrix in the middle is called the *intrinsic camera matrix*. For a sensor with rectangular pixels or a less than perfect lens, the horizontal and vertical focal distances may be different. c_x and c_y represent the coordinates of the principal point, in pixels. This point is usually near the center of the sensor, but tolerances in camera assembly can shift it; no two cameras are exactly the same.

Exercise 1

Write a simple camera simulator that projects a 3D wireframe model to image coordinates.

Question 1 What is the camera matrix for a 1080p camera with a horizontal field of view of 90 degrees?

Assignment 1

- Create a virtual 3D cube with a side of 1 meter, defined as an 8x3 matrix containing the 3D coordinates of the vertices of the cube relative to the camera (you can choose the exact position, picture the situation in your head or make a quick freehand diagram), and a 12x2 array of edges described by pairs of vertex indices that need to be connected by lines;
- Project the 3D vertex coordinates to 2D image coordinates using your camera matrix from Question 1, and visualize the result by drawing the vertices and edges on an empty 1080p image. Pay attention to the dimensions of your matrices, transpose as necessary and round the image coordinates to integer pixels.

Figure 3 shows an example of a cube projected like this.
Tip: use numpy for the mathematics and the opencv drawing functions **line** and **circle**.

Question 2 If you double the focal distance, what happens to the picture?

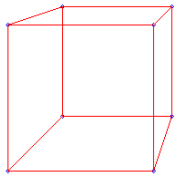

Figure 3: Virtual cube rendered with pinhole camera model.

2 Camera calibration

More advanced tasks in computer vision, such as stereo vision, pose estimation and structure from motion build on the pinhole camera model and therefore require us to determine the camera matrix for a given camera and lens combination. Important to note is that the pinhole model is a poor approximation in practice: lenses do not behave like an infinitesimally small hole. A lens achieves the same purpose (focusing light coming from one direction onto one place on the image sensor) by refracting incoming light beams, which allows for a lot more light to come through (the aperture can be much larger) but also induces distortion: the simple relation defined by the similar triangles no longer holds.

There are various types of distortion, but the dominant effect is *radial distortion*, where the light rays are deflected radially: incident rays that form a large angle with the principal axis, seem to have a different focal distance than rays that are closer to the principal axis. Rather than have a focal distance that depends on the incidence angle, this type of distortion is compensated for by a transformation that converts the real image coordinates into pinhole model coordinates. The transformation is given by a polynomial correction. Let r be the distance of the original image point (x, y) to the principal point, i.e. $r = \sqrt{(x_{orig} - c_x)^2 + (y_{orig} - c_y)^2}$. The corrected coordinates of the point are then:

$$x_{pinhole} = c_x + (x_{orig} - c_x)(1 + k_1 r^2 + k_2 r^4)$$
$$y_{pinhole} = c_y + (y_{orig} - c_y)(1 + k_1 r^2 + k_2 r^4).$$

This transformation moves the point along the radial line through the principal point along a distance specified by the distortion coefficients k_1 and k_2. More than two coefficients can be used (for higher powers of r) but in practice this is not usually required.

Question 3 Why are there only even powers in this polynomial in r?

The most common method to simultaneously determine the camera matrix and the distortion coefficients is Zhang's method, which is an iterative procedure that estimates the intrinsic matrix from many different views of a known object assuming no distortion, then estimates the distortion and then refines all parameters

using a non-linear optimization technique.

Exercise 2

Calibrate a camera based on checkerboard pattern views.

Assignment 2 Determine the intrinsic matrix and distortion parameters of the gopro camera used to shoot the calibration sequence you find on Ufora as **calibration_frames.zip**. Print the matrix and coefficients in your report.
Assignment 3 Do the calibration procedure for 5 different random subsets of 20 frames. Print the camera matrix for each set.
Question 4 How can you find out which calibration is the best? Look in the opencv documentation.
Assignment 4 Use your best calibration result to undistort one of the frames from the sequence. Lines that are straight in reality should be straight in your rectified image now.

Tip: there is a camera calibration example in the opencv documentation.

3 Geometric transformations

In this lab we discuss some types of two-dimensional geometric transformations. These are transformations that convert a plane into another plane. Examples are translation, rotation, scaling, affine transformation and perspective transformation.

Translation, rotation and scaling speak for themselves. Affine transformations are all transformations that maintain collinearity and distance relationships. Each affine transformation can be seen as a combination of translation, rotation, scaling, and shear. Perspective transformations (also called homographies) are transformations that only retain linearity.

Figure 4: Original image (left), shear transformation (central), perspective transformation (right).

Exercise 3

Write a program that applies a horizontal shear transform to an image. The transformation matrix to shear horizontally has this form:

$$\begin{bmatrix} 1 & m & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

where m is the shear factor. Note the dimensions: this is an image-to-image transform, but it has 3 columns, meaning it applies to *homogeneous coordinates*.

Assignment 5 Shear **shadow.png** so that the photographer's shadow becomes vertical. Size your target image so that it will be large enough to accommodate the sheared image, and make sure that all parts of

the original image are visible. You can add translation by placing pixel offsets in the third column.

New functions: **warpAffine**.

Exercise 4

Write a program that applies a perspective transform to obtain a perpendicular view on the ground plane, also called a *bird's eye view*. Unlike a rotation, scaling or shear transform, a perspective transform is not trivial to define manually. Usually, you will determine a perspective transform by solving for it: you specify a set of source coordinates and target coordinates for a set of points, and the transformation that maps one onto the other is determined by solving the following system:

$$s \begin{bmatrix} x_{target} \\ y_{target} \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x_{orig} \\ y_{orig} \\ 1 \end{bmatrix}.$$

This system has 8 degrees of freedom (it is only defined up to a scale factor, which does not matter in homogeneous coordinates), so you need source and target coordinates for at least 4 points (resulting in 4 equations in x and 4 equations in y).

Assignment 6 Apply a perspective transform to **shadow_box.png** so that the photographer in not only stands vertically, but is also proportionally correct. In your program you click on the 4 corners of the tetragon that you want to transform into a rectangle, after which the right perspective transformation is searched for and executed. In order to make an image display window clickable, you have to set a mouse callback function that is called anytime your mouse pointer interacts in some way with the window.

Tip: an example of a mouse callback function can be found in `callback.py` on Ufora. Functions you need: **setMouseCallback, getPerspectiveTransform, warpPerspective**.

Figure 5: Original image (left), bird's eye view (right).

Question 5 Can you do this for 5 point correspondences instead of 4? How could such an overdetermined system be solved?

4 Report

Write a brief report about this lab session in which you answer each question and provide the output image of each assignment. Append your source code.