Radial Distortion Estimation, Image Undistortion
COMP 776, Fall 2017
Due: September 20, 2017
Assignment Data: https://drive.google.com/open?id=0BzP2GHNy5xrfVGg5ZlhVMXY3UEk

Summary

The goal of this reading/assignment is for you to familiarize yourself with lens distortion. For the assignment: Given an image with a known focal length and principal point, but unknown distortion parameters, you will write code to estimate the amount of radial distortion. Here, you will achieve this by manually selecting points on a curve in the image that corresponds to a straight line in the real world (e.g., the edge of a building or a curb). Once the distortion coefficient is estimated, you will use it and the known intrinsics to undistort the image.

Distortion

Related Reading: Szeliski, S2.1.6; Hartley & Zisserman, S7.4
Recall that *camera obscura* images have an inherent trade-off between brightness and clarity: Small camera apertures reduce blur but make the image dimmer. For centuries, *lenses* have been used to address this problem by decreasing the spread of light rays that fall on a given point in the image plane. A simple depiction of the optical properties of lenses is shown in Fig. 1.

In practice, however, the refractive properties of the lens cause distortion in the path (and color) of light entering the camera. *Spatial lens distortion*, in particular, causes the outgoing light rays from the lens to strike the image plane at different points than they would have under regular perspective projection. The resulting projection is *nonlinear*, meaning that the magnitude of spatial distortion is different for different incoming light rays. A simple depiction of the optics of distortion is shown in Fig. 2.

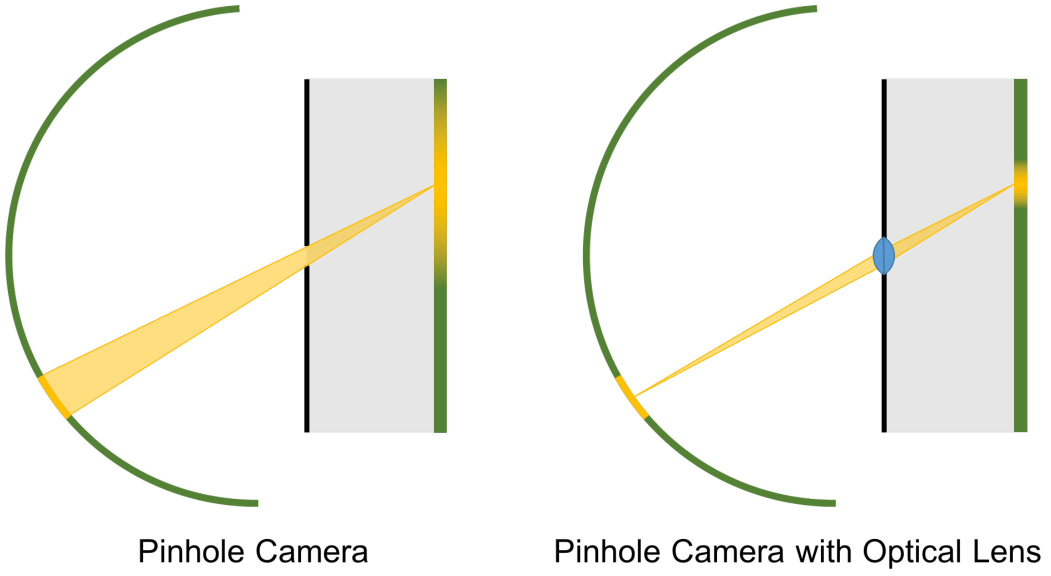

Figure 1. In a lens-less camera (left), a range of input light rays contribute to the color of a single point on the image plane. Larger apertures lead to more input light rays per point, causing increased image blur. Smaller apertures decrease the number of rays, which leads to less overall light exposure for points in the image. When a lens is added (right), outgoing light from a single point in the world space converges to a single point on the image plane. This removes image blur without decreasing the number of input light rays per point. Image clarity, however, further relies on both the image sensor and the world point being the correct distance from the lens.

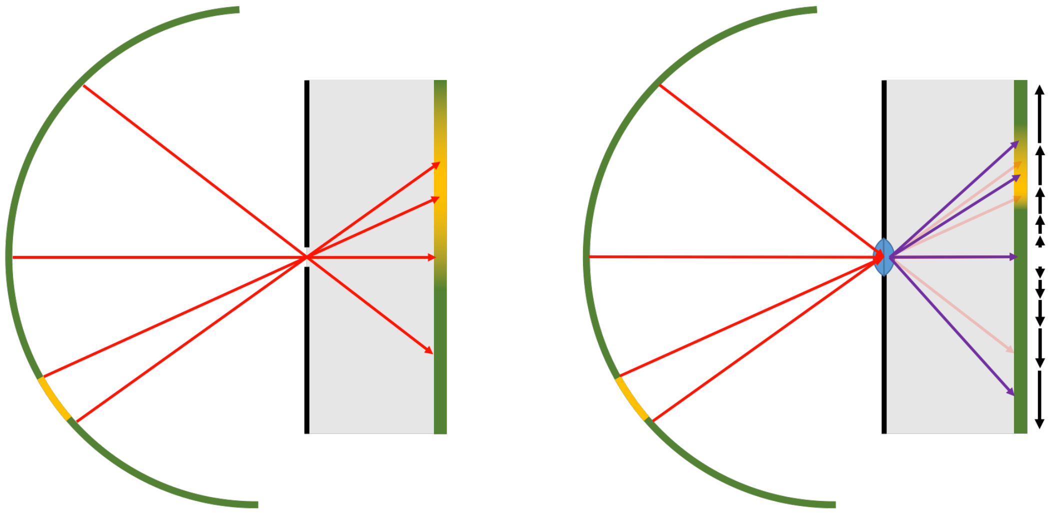
Pinhole Camera (no distortion)
Optical Lens (pincushion distortion)

Figure 2. (Left) The paths of light rays (red) entering a pinhole camera are unaffected by the aperture. (Right) When a lens is added, the path of the outgoing rays (purple) is affected because the light rays *refract* through the lens – that is, each ray's direction changes as it enters and exits the lens. The amount of refraction is related to the shape and material of the lens itself. This figure shows *pincushion radial distortion*, where incoming rays are pushed further from the center of the image with increasing magnitude.

There are two main types of lens distortion. We'll focus on ***radial distortion***, which is the spatial distortion of incoming/outgoing rays and is caused primarily by lens refraction. Another type of distortion is ***tangential distortion***, caused by the lens being rotated or shifted w.r.t. the camera axis. (Lens shifting is also known as "decentering.") Aberrations in lens shape, as well as the lens material itself, can also contribute to both types of distortion.

Obviously, the physical properties governing lens distortion can be very complex. Fortunately, relatively simple models can be adopted for many modern cameras, without much loss of accuracy. Possibly the most widely used distortion model is the ***Brown-Conrady distortion model***,[1] which treats distortion as a post-processing operation (2D polynomial warping) after 3D points are projected into the image plane.[2] The distortion function in this model is given as

$$\boldsymbol{x_d} = \boldsymbol{x_u} + \boldsymbol{x_u}(k_1 r^2 + k_2 r^4 + \cdots) + \begin{bmatrix} p_1(r^2 + 2x^2) + 2p_2 xy \\ 2p_1 xy + p_2(r^2 + 2y^2) \end{bmatrix} (1 + p_3 r^2 + p_4 r^4 + \cdots)$$

where $\boldsymbol{x_d}$ is the 2D distorted image point; $\boldsymbol{x_u} = (x, y)$ is the 2D undistorted image point after perspective projection; $r = \|\boldsymbol{x_u}\|$ is the distance of the point from the distortion center;[3] $k_1,\ k_2,$ … are *radial distortion coefficients*; and $p_1, p_2,$ … are *tangential distortion coefficients*. We will adopt a simple version of this model in this assignment, only using the k_1 radial distortion term:

$$\boldsymbol{x_d} = \boldsymbol{x_u}(1 + k_1 r^2).$$

A depiction of this distortion model is shown in Fig. 3.

[1] Brown, D. C. "Close-range camera calibration." *Photogrammetric Engineering*, 37(8):855–866, 1971. [link]
[2] See also the class slides regarding the conventions for $\boldsymbol{x_u}$ and $\boldsymbol{x_d}$. Brown's formulation swaps them.
[3] Here, we'll assume the distortion center is the same as the camera's principal point, but this isn't necessarily true.

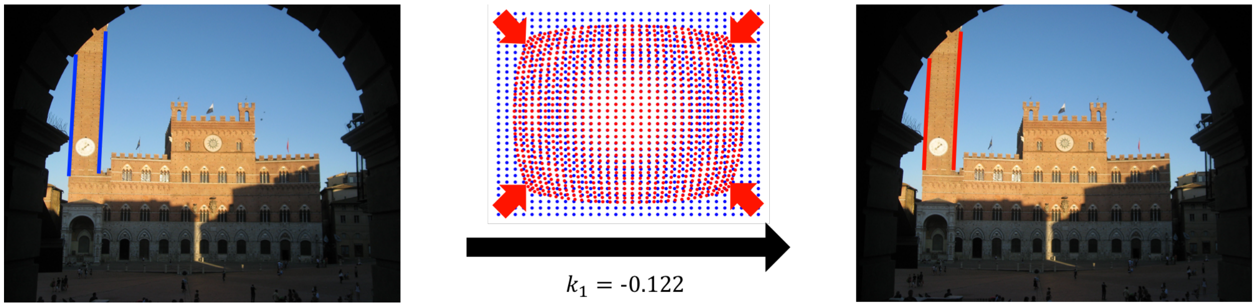

Figure 3. Barrel radial distortion. (Left) Without lens distortion, a 3D scene would be imaged with regular perspective projection. Straight 3D lines in the world project to straight lines in the image (shown as blue lines). (Middle) The Brown-Conrady model warps the 2D image space using a polynomial function. Here, we show a *barrel radial distortion* warping; blue points are moved to red points. (Right) With lens distortion, straight 3D lines project to curves in the 2D image (red curves; the curvature is somewhat subtle, here).

There are a number of higher-precision distortion models. In addition to tangential terms in the Brown-Conrady distortion model, other models (such as the full model used in OpenCV) might incorporate non-polynomial radial distortion terms. Non-parametric distortion models have also been proposed. Finally, one alternative distortion model that is sometimes used is the division model, which approximates the distortion function with

$$\boldsymbol{x_u} = \boldsymbol{x_d}(1 + k_1 r_d^2 + k_2 r_d^4 + \cdots)^{-1},$$

where $r_d = \|\boldsymbol{x_d}\|$.

Assignment

Your assignment is to write an optimization function to estimate a single radial distortion coefficient k_1 of an image with a known focal length and principal point. You can then visually verify this by undistorting the image using the estimated parameter (code already provided).

The basic idea is as follows: As shown in Fig. 3, straight lines in the world space are projected to straight lines in the image space under normal perspective projection, but they are distorted into curves by the lens. So, if we pick points on a curve in the distorted image (say, one of the red curves in Fig. 3), we can try undistorting the points using different values of k_1 and see which undistortion yields the closest thing to a straight line.

We will solve this problem using 3 manually selected points and non-linear optimization. For a given input image, you will first need to determine 2D pixel coordinates for three points in the image that fall along the same (straight) 3D world line.[4] Let's call these distorted coordinates $\boldsymbol{x_d^1}$, $\boldsymbol{x_d^2}$, and $\boldsymbol{x_d^3}$. These coordinates, along with the camera intrinsics, will be passed to an *optimizer* that will search for the best value of k_1 to explain the observed curve in the points.

To optimize k_1, we'll need to a define an *objective function*, in this case a function whose value we wish to minimize w.r.t. k_1. If we write the undistorted coordinates of the points as $\boldsymbol{x_u^1}$, $\boldsymbol{x_u^2}$, and $\boldsymbol{x_u^3}$, one possible objective function is to minimize the distance from the point $\boldsymbol{x_u^3}$ to the line formed between $\boldsymbol{x_u^1}$ and $\boldsymbol{x_u^2}$ – when this distance is zero, all three points lie on the same line.

[4] It is better if the points are nearer to the edge of the image and far apart – e.g., the endpoints and middle point of the left red curve in Fig. 3 would be a good choice for that image.

To put this more formally, our optimal value of k_1 is given as

$$\widehat{k_1} = \underset{k_1}{\arg\min}\ d(\boldsymbol{x_u^3}(k_1), \boldsymbol{l}(\boldsymbol{x_u^1}(k_1), \boldsymbol{x_u^2}(k_1)),$$

where $\boldsymbol{l}(\boldsymbol{x_u^1}, \boldsymbol{x_u^2})$ is the 2D line between $\boldsymbol{x_u^1}$ and $\boldsymbol{x_u^2}$, and $d(\cdot,\cdot)$ is the point-to-line distance. We'll optimize this function using SciPy's *minimize_scalar()* function (more below).

Concerning undistortion, we can rewrite the 1-parameter radial distortion as

$$\boldsymbol{x_u}(1 + k_1 r^2) - \boldsymbol{x_d} = 0.$$

Thus, we can use a root-finding method (e.g., Newton's method) to find the value of $\boldsymbol{x_u}$ for a provided value of $\boldsymbol{x_d}$. In summary, your optimizer function should take $\boldsymbol{x_d^1}$, $\boldsymbol{x_d^2}$, and $\boldsymbol{x_d^3}$ as input (plus camera matrix K) and perform the following steps:

1) Convert the 2D points from pixel coordinates into normalized camera coordinates.
2) Use SciPy's *minimize_scalar()* function to minimize the above objective function. The objective function should
    a. Compute $\boldsymbol{x_u^1}$, $\boldsymbol{x_u^2}$, and $\boldsymbol{x_u^3}$ for the provided value of k_1.
    b. Calculate and return the point-to-line distance from $\boldsymbol{x_u^3}$ to $\boldsymbol{l}(\boldsymbol{x_u^1}, \boldsymbol{x_u^2})$.
3) Return the estimated optimal value $\widehat{k_1}$.

Admittedly, the k_1 estimation strategy presented here is not too robust – it would be better to have subpixel precision on the selected points, and to optimize the value of k_1 using more points/curves. However, your implementation should still be able to undistort the image reasonably well.

Data and Code
We have provided two images of the Alamo for you to experiment on, one with pincushion distortion and one with barrel distortion. We've also provided you with a pre-computed K matrix for each image; K transforms points from normalized camera coordinates into pixel coordinates.

Concerning code, there are three files:

- *main.py*: This file is already set up to run the optimizer and subsequently undistort the input image. For usage, execute "*python main.py --help*".
- *undistort.py*: This file is already set up for undistorting a given image. In short, the *undistort_image()* function computes the point $\boldsymbol{x_d}$ for each point $\boldsymbol{x_u}$ in the undistorted output image pixel space. Each pixel in the output image is then assigned an RGB color using bilinear sampling of the input image at its corresponding point $\boldsymbol{x_d}$.
- *calculate_k1.py*: You will need to complete this file by filling in the *objective()* local function within the *calculate_k1()* main function. We have already written code for you to use the *objective()* function within SciPy's *minimize_scalar()* function. Check that you also understand what the SciPy function is doing (documentation here).

Submission
Submit a single PDF containing your undistorted images and the estimated value of $\widehat{k_1}$ for each. Also in your PDF, please include your name and a link to your code in your department Google Drive account. Be sure to share the folder with Jan-Michael Frahm, Marc Eder, and True Price.