TTK4255 Robotic Vision, Spring 2020

Exercise 2: Camera Calibration and Image Undistortion

Authored by Simen Haugo (simen.haugo@ntnu.no)

Due: See Blackboard

This exercise is a mix of programming problems and theory questions. You are free to use any programming language, but solutions will be provided in Python and Matlab, and we may not be able to help you effectively if you use a different language. We therefore recommend that you use either Python or Matlab. You should also do exercise 0 to get familiar with language features and libraries that are useful for the exercises.

Instructions

This exercise is **approved / not approved**. To get your exercise approved, submit your answers to the questions as a **PDF** to Blackboard. You **don't** have to submit your code. Feel free to use LaTeX, Word, handwritten scans, or any other tool to write your answers.

Getting help

If you have questions about the exercise, you can:

- Post a question on Piazza (you can post anonymously if you want)
- Ask for help during the tutorial sessions (see Piazza for time and place)

Relevant resources

Below are the relevant slides and chapters from the course syllabus for this exercise (the course material is accessible on Piazza under Resources).

- Lecture 2
- Matlab Single Camera Calibrator App: You won't be using this application in this exercise, but the page linked to is a good overview of a typical calibration process and important terminology. Read and expand all the bullet points on the page.

$\boxed{1}$ Image undistortion

The pinhole camera model neglects many aspects of real cameras, such as lenses which can cause significant deviation from a rectilinear projection. [1] Thus, straight lines in the world may not remain straight in the image. If the distortion is not too big, it can be compensated for by an appropriate coordinate transformation. Transforming an image with lens distortion into one that satisfies a rectilinear projection is called undistortion, which you will implement in this exercise.

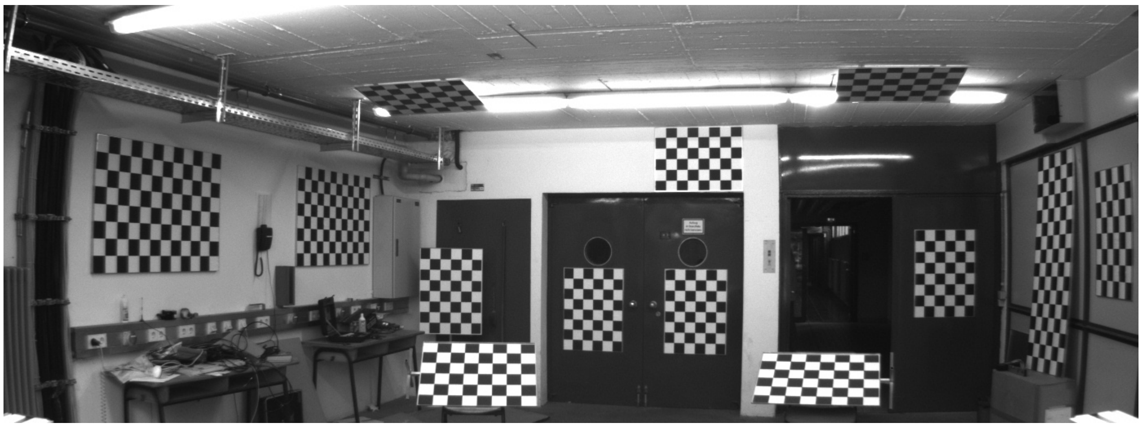


Figure 1: Example image from the KITTI dataset. Lens distortion is particularly noticeable on the sides of the image and on the ceiling tiles.

Recall the pinhole camera model from the previous exercise:

$$u = c_x + f_x X/Z \tag{1}$$
$$v = c_y + f_y Y/Z \tag{2}$$

A common extension that takes small lens distortion into consideration is the polynomial distortion model. [2] For convenience define $x = X/Z$ and $y = Y/Z$. The pixel coordinates of a point (X, Y, Z) under this extended model is given by:

$$u = c_x + f_x(x + \delta_x) \tag{3}$$
$$v = c_y + f_y(y + \delta_y) \tag{4}$$

where the distortion terms (δ_x, δ_y) consist of two types of distortion:

- Radial: points on a circle around the optical axis are distorted equally.
- Decentering: points are distorted in both radial and tangential directions.

Radial distortion is a polynomial in the radial distance $r = \sqrt{x^2 + y^2}$ while decentering distortion involves cross-terms as well:

$$\delta_x = (k_1r^2 + k_2r^4 + k_3r^6 + \cdots)x + 2p_1xy + p_2(r^2 + 2x^2) \tag{5}$$
$$\delta_y = \underbrace{(k_1r^2 + k_2r^4 + k_3r^6 + \cdots)y}_{\text{Radial}} + \underbrace{p_1(r^2 + 2y^2) + 2p_2xy}_{\text{Tangential}} \tag{6}$$

The distortion coefficients $k_1, k_2, ..., p_1, p_2$ are determined by camera calibration along with the standard pinhole model parameters c_x, c_y, f_x, f_y.

(a) One method is based on iterating over each pixel in the desired undistorted image (*destination*), and calculating which pixel from the original distorted image (*source*) should be used. Let $\mathcal{I}_{\text{dst}}, \mathcal{I}_{\text{src}}$ be the undistorted and original image, respectively. The undistortion method is as follows:

1. Assume an ideal rectilinear projection applies for the undistorted image. For each pixel $(u_{\text{dst}}, v_{\text{dst}})$ to be colored in the destination image, compute (x, y) by inverting equations (1)-(2).

2. Using equations (3)-(4) and (x, y) computed previously, compute the source image pixel coordinates $(u_{\text{src}}, v_{\text{src}})$ under the distorted pinhole model.

3. Assign $\mathcal{I}_{\text{dst}}(u_{\text{dst}}, v_{\text{dst}}) \leftarrow \mathcal{I}_{\text{src}}(u_{\text{src}}, v_{\text{src}})$.

Write a program that implements the above method and undistort the example image `kitti.jpg`. Include the undistorted image in your submission. The camera parameters and distortion coefficients are provided in `kitti_parameters.txt`.

Hint: The source pixel coordinates will generally not be integers. The standard way to handle this is bilinear interpolation, but for simplicity you can simply round the coordinates to the nearest integer.

Hint: If you're using Python or Matlab, you may need to specify that the destination image datatype should be uint8, unless you convert the input to floating point first.

(b) This method generates the image that would have been seen by a camera with the same intrinsic parameters, but zero distortion. Explain how to modify the method to generate an image seen by a camera with *different* intrinsic parameters.

Solution: The inversion in step 1 should be modified to use the desired intrinsic parameters. The projection in step 2 should use the original (true) intrinsic parameters. In fact, as long as (x, y) can be computed, an arbitrary camera model can be used in step 1. For example, it's possible to simulate a fisheye lens from the image taken by a rectilinear camera.

(c) It's also possible to generate an image seen by a camera at the same position, but different orientation. Not all pixels will receive a value, but otherwise the image will be correct. Using equations (1)-(4), show why this is possible.

Solution: Consider a pixel (u, v) in the destination image, which is the projection of some 3D point $\mathbf{X}^{\text{dst}} = (X, Y, Z)$ in the rotated camera's coordinate frame. If we knew $\mathbf{X}^{\text{dst}}$, we could recover the corresponding pixel in the source image by first rotating into the original camera's coordinate frame:

$$\mathbf{X}^{\text{src}} = \mathbf{R}^{\text{src}}_{\text{dst}} \mathbf{X}^{\text{dst}}$$

and then applying equations (3)-(4). Unfortunately, we don't know $\mathbf{X}^{\text{dst}}$ because depth is lost in the projection. But, since we know the camera intrinsics, we can obtain (x, y) from (u, v):

$$x := \frac{X}{Z} = \frac{u - c_x}{f_x} \quad \text{and} \quad y := \frac{Y}{Z} = \frac{v - c_y}{f_y}$$

and thereby establish that $\mathbf{X}^{\text{dst}}$ lies along a known vector

$$\mathbf{X}^{\text{dst}} = Z \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = Z\boldsymbol{x}^{\text{dst}}$$

for some unknown depth Z. We then have

$$\mathbf{X}^{\text{src}} = \mathbf{R}^{\text{src}}_{\text{dst}}\mathbf{X}^{\text{dst}} = Z\mathbf{R}^{\text{src}}_{\text{dst}}\boldsymbol{x}^{\text{dst}}$$

The right-hand side is all known or computable quantities, except for Z. To obtain the corresponding source image pixel we apply equations (3)-(4). However, since this involves a division, the unknown depth Z cancels out, and we are therefore able to compute the source image pixel without knowledge of the depth.

To support this, step (b) can be modified to use $\mathbf{R}^{\text{src}}_{\text{dst}}\boldsymbol{x}^{\text{dst}}$ rather than the unrotated (x, y) when computing the projection. Note that you need to divide by the third component.

(d) Explain why it's not possible to generate the image seen by a camera at a different position, in general.

Solution: This follows from a small modification of the previous answer. Now, the transformation includes a translation. Thus,

$$\mathbf{X}^{\text{src}} = \mathbf{R}^{\text{src}}_{\text{dst}}\mathbf{X}^{\text{dst}} + \mathbf{t}$$

We can rewrite this as above by pulling out the unknown depth:

$$\mathbf{X}^{\text{src}} = Z\mathbf{R}^{\text{src}}_{\text{dst}}\boldsymbol{x}^{\text{dst}} + \mathbf{t}$$

Again the right-hand side is all known or computable quantities, except for Z. Unfortunately, Z doesn't cancel out during projection in this case, due to the added term $\mathbf{t}$, and we therefore can't compute the corresponding source image pixel. The exception to this is when both cameras are observing a planar scene, in which case the mapping is defined by a homography.

$\boxed{2}$ Camera calibration

Calibration is the problem of estimating camera model parameters. For instance in the previous task, these are the focal length, principal point and distortion coefficients. Most calibration software today is based on taking images of an object with known geometry, called a calibration rig/target. As it can be impractical to position the rig accurately with respect to the camera coordinate system, the transformation between the rig's coordinate system and the camera is typically also estimated.

(a) What are the *intrinsics* and *extrinsics* for the camera model and calibration setup in task 1 (Fig. 1)? How many extrinsic parameters are there in total?

Solution: The extrinsics are the camera's position and orientation relative to each calibration target. The intrinsics are the focal length, principal point and distortion coefficients. Since one relative pose has six degrees of freedom, there are in total $6n$ extrinsic parameters, where n is the number of calibration targets.

(b) Calibration can be done using any object that has a set of distinct points with known position relative to the object. Using a *planar* object is most common. What are the primary advantages of a planar calibration target?

Solution: Planar targets are much easier to manufacture than 3D targets.

(c) A planar target can be printed out and attached to a flat surface. It can also be displayed on a screen, such as on your phone or laptop. Can you think of some pros/cons (advantages and challenges) to these two approaches?

Solution:
Printed:

+ Can be printed matte to avoid specular highlights
+ More visible outdoors
+ Easier to deploy, e.g. underwater in a casing
+ Maybe cheaper to print a large target than using a monitor?
- Takes more time to prepare
- Ensuring flatness can be difficult or expensive

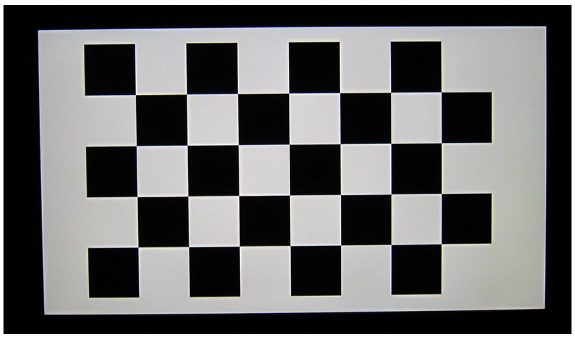
Screen:

+ Easy to prepare
+ Easy to ensure flatness
- Requires a power supply
- Reduced visibility as viewing angle increases
- Less visible outdoors
- A small screen can be difficult to work with. Taking images close-up will require a focus setting that may not match the actual focus setting you will

use during runtime, and thus yield inaccurate calibration. Taking images far away means the target occupies a small portion of the image, which makes it harder to detect the pattern accurately.

(d) An image yields a set of correspondences between known points in the rig and their observed pixel coordinates. As these are related by the projection equations (3)-(4), we obtain two equations for each correspondence. Assume that there is no lens distortion. Based purely on the number of equations and degrees of freedom, how many correspondences are needed at minimum to estimate the intrinsics and extrinsics from a single image of a calibration rig?

Solution: The intrinsics f_x, f_y, c_x, c_y make up four degrees of freedom. The extrinsics (rotation and translation) have six degrees of freedom. That is in total ten degrees of freedom, which requires at least ten equations, and thus at least five point correspondences. Note that if the pinhole model were to include a skew parameter (as in some textbooks), we would need at least six points.



(e) The above image of a checkerboard yields 28 correspondences, yet it can't be used alone to uniquely determine all parameters. Give an example of two different configurations (intrinsics and extrinsics) that are both possible from this image. (You don't need to specify any values here, a sketch or rough argument is sufficient).

Solution: As discussed in *A Flexible New Technique for Camera Calibration* by Zhengyou Zhang, for a planar target we need at least two images from different viewpoints in order to have a unique solution for the camera intrinsics. (Side note: If we have a 3D target, meaning the model points occupy a volume in space, a single viewpoint is enough.)

This can be seen intuitively in this case of a frontoparallel view, which causes ambiguity between the focal length and the distance to the target along the optical axis. Additionally, there are ambiguities between the principal point (c_x, c_y) and the xy translation of the camera. Infinitely many configurations that balance these parameters will yield the same image of the checkerboard points.

(f) Reprojection error is often reported as an indication of calibration quality, but it can't always be trusted. Give an example where the reprojection error can be zero in each image, yet the calibration is probably bad.

Solution: One example is the above image, or in general whenever the viewpoints don't sufficiently constrain the calibration parameters. For example, if most of the images are taken from primarily one viewpoint, or if the images are biased such that the reprojected points don't evenly distribute in the image, the resulting estimate may be biased as well despite having a low reprojection error. In those cases looking at the distribution of reprojection errors across each image can be useful.

Notes

[1]Although lens distortion is often discussed in textbooks as an imperfection that needs to be corrected, it should be emphasized that distortion can be an intentional part of the lens design, and that many computer vision algorithms are perfectly applicable with non-rectilinear images. For instance, distortion is utilized in fisheye lenses to map a large field of view, often near or greater than 180 degrees, onto a planar imaging sensor while preserving local features over the image, such as the angle or shape of objects. It can be impractical, and possibly unecessary, to undistort such an image into an equivalent rectilinear projection.

[2]This model (also known as the radial-tangential or Brown-Conrady model) can be considered a standard, and is supported by many computer vision libraries and tools, including OpenCV, Kalibr and Matlab's Computer Vision Toolbox.