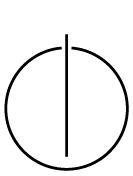

Multiple View Geometry: Exercise Sheet 4

Prof. Dr. Daniel Cremers, Christiane Sommer, Rui Wang
Computer Vision Group, TU Munich
http://vision.in.tum.de/teaching/ss2017/mvg2017

Exercise: June 1st, 2017

Part I: Theory

This part of the exercises should be **solved at home**.

1. **Image Formation**

   We are looking at the formation of an image in camera coordinates $\mathbf{X} = (X\ Y\ Z\ 1)^\top$. In the lecture, you learned the following relation of homogeneous pixel coordinates $\mathbf{x}'$ and $\mathbf{X}$:

   $$\lambda \mathbf{x}' = K \Pi_0 \mathbf{X} \tag{1}$$

   with the intrinsic camera matrix K. To clearly differentiate between camera coordinates and pixel coordinates, call the pixel coordinates n and m: $\mathbf{x}' = (n\ m\ 1)^\top$. Furthermore, let the non-homogeneous camera coordinates be $\tilde{\mathbf{X}} := \Pi_0 \mathbf{X} = (X\ Y\ Z)^\top$. (1) is then equivalent to

   $$\lambda \begin{pmatrix} n \\ m \\ 1 \end{pmatrix} = K \tilde{\mathbf{X}} \,. \tag{2}$$

   Let $s_x = s_y = 1$ and $s_\theta = 0$ in the intrinsic camera matrix.

   (a) Compute λ and show that (2) is equivalent to

   $$n = \frac{fX}{Z} + o_x \,, \quad m = \frac{fY}{Z} + o_y \,. \tag{3}$$

   (b) A classic ambiguity of the perspective projection is that one cannot tell an object from another object that is exactly *twice as big but twice as far*. Explain why this is true.

   (c) For a camera with $f = 540$, $o_x = 320$ and $o_y = 240$, compute the pixel coordinates n and m of a point $\tilde{\mathbf{X}} = (60\ 100\ 180)^\top$. Explain with the help of (b) why the units of $\tilde{\mathbf{X}}$ are not needed for this task. Will the projected point be in the image if it has dimensions 640×480?

   We define the generic projection π of $\tilde{\mathbf{X}}$ to 2D coordinates as follows:

   $$\pi(\tilde{\mathbf{X}}) := \begin{pmatrix} X/Z \\ Y/Z \end{pmatrix}$$

   (d) Using the generic projection π, show that (1), (2) and (3) are equivalent to

   $$\begin{pmatrix} n \\ m \\ 1 \end{pmatrix} = K \begin{pmatrix} \pi(\tilde{\mathbf{X}}) \\ 1 \end{pmatrix} \,.$$

2. **Radial Distortion**

A general image formation model for radially distorted cameras is generic projection followed by a non-linear transformation of the radius for each image point. The distorted coordinates of a generically projected point $\tilde{\mathbf{X}}$ are given by

$$\pi_d(\tilde{\mathbf{X}}) = g\left(\|\pi(\tilde{\mathbf{X}})\|\right) \cdot \pi(\tilde{\mathbf{X}}) \in \mathbb{R}^2 \,. \tag{4}$$

$g : \mathbb{R}^+ \to \mathbb{R}^+$ is the function that radially distorts the coordinates of $\pi(\tilde{\mathbf{X}})$. It is typically approximated by some parametric function. The pixel coordinates of the distorted camera are

$$\begin{pmatrix} n_d \\ m_d \\ 1 \end{pmatrix} = K \begin{pmatrix} \pi_d(\tilde{\mathbf{X}}) \\ 1 \end{pmatrix} = K \begin{pmatrix} g\left(\|\pi(\tilde{\mathbf{X}})\|\right) \cdot \pi(\tilde{\mathbf{X}}) \\ 1 \end{pmatrix} .$$

(a) Can this model be used for lenses with a field of view of more than $180°$?

The so-called FOV or ATAN model, first suggested by Devernay and Faugeras in 2001, and used e.g. in the open source implementation of PTAM (Parallel Tracking and Mapping), is given by

$$g_{\text{ATAN}}(r) = \frac{1}{\omega r} \arctan\left(2r \tan\left(\frac{\omega}{2}\right)\right) \,. \tag{5}$$

(b) Derive a closed form solution for f in the undistortion formula

$$\pi(\tilde{\mathbf{X}}) = f\left(\|\pi_d(\tilde{\mathbf{X}})\|\right) \cdot \pi_d(\tilde{\mathbf{X}}) \tag{6}$$

using (4) and $g(r) = g_{\text{ATAN}}(r)$.
Hint: compute the norm of both sides of the equation in (4) and (6).

Note (additional information):
Both formulations of a distortion model, (4) and (6), can be found in the literature. (6) is the one that was presented in the lecture for a polynomial f. In order to switch between the two formulations, it is beneficial if $g(r)r$ and $f(r_d)r_d$ are invertible in closed form. For that reason, the ATAN model is very popular. Another popular choice for radial distortion functions are polynomials.

3. **Image Rectification**

Many computer vision algorithms require a perfect perspective projection. For example, line detection algorithms need straight lines that are not curved due to distortion. Thus, image rectification is very important. Let I_d be a distorted image taken by a camera with intrinsic parameter matrix K_d.

Write down expressions for the pixel coordinates n_d, m_d of a point $\tilde{\mathbf{X}} \in \mathbb{R}^3$ in the distorted image and for the pixel coordinates n, m of the same point projected by a perfect pinhole camera with intrinsic camera matrix K.

Part II: Practical Exercises

These exercises are to be solved **during the tutorial**.

1. **Image Formation**

   Consider the 3D model `model.off` from the last exercise sheet (contained in the package `ex3.zip`). You can use `openOFF` from `ex3.zip` to load the model into your workspace.

   (a) Translate the model vertices by the vector $\mathbf{T} = (-0.5 \ -0.5 \ 1)^\top$ using homogeneous coordinates and a 3×4 rigid body transformation matrix.

   (b) Compute the perspective projection of the new model imaged by a camera with intrinsic parameters $f = 540$, $o_x = 320$, $o_y = 240$ and $s_x = s_y = 1$, $s_\theta = 0$. Visualize the projection and scale the axes to `[0 640 0 480]` to see if the whole model fits into the image.

   A projection model that is widely used e.g. in microscopy is the parallel projection model. In this model, the generic perspective projection π is replaced by a parallel projection orthogonal to the z-axis

   $$\pi_{\text{parallel}}(\tilde{\mathbf{X}}) = \begin{pmatrix} X \\ Y \end{pmatrix} .$$

   (c) Compute the parallel projection of the translated model using the same camera intrinsics as in (b), but the parallel projection π_{parallel}. Again, visualize the projection and scale the axis to `[0 640 0 480]` to see if the whole model fits into the image.

2. **Radial Distortion and Image Rectification**

   In this exercise you will compute a rectified image from a radially distorted image. Given a distorted image I_d with projection function π_d as defined in (4) and camera intrinsics K_d, this amounts to computing a virtual, undistorted image I_{new} with generic projection function π and a given camera matrix K_{new}.

   (a) Download `ex4.zip` and extract its content. `img1.jpg` is the distorted image, with camera intrinsics

   $$K_{d1} = \begin{pmatrix} 388.6 & 0 & 343.7 \\ 0 & 389.4 & 234.6 \\ 0 & 0 & 1 \end{pmatrix},$$

   and a radial distortion $g = g_{\text{ATAN}}$ given by the ATAN model in (5) with $\omega = 0.92646$. Load the image using `Id1 = imreadbw('img1.jpg')`, and display it. Change the colormap to grayscale.

   Note how straight lines in the scene appear as *curved* lines in the image.

   (b) Compute a virtual, rectified image I_{new} of dimensions 1024×768, with a projection function according to a pinhole camera model and intrinsic parameters

   $$K_{\text{new}} = \begin{pmatrix} 250 & 0 & 512 \\ 0 & 250 & 384 \\ 0 & 0 & 1 \end{pmatrix}$$

   The intensity $I_{\text{new}}(n, m)$ at pixels n, m is computed by

   - un-projecting $(n \ m \ 1)^\top$ to obtain $\pi(\tilde{\mathbf{X}})$ for each projected point,
   - computing the according pixel coordinates n_d, m_d in the distorted image (i.e. re-projecting), and
   - interpolating the intensity I_d at the projected position (n_d, m_d).

Use the result of Ex. 3 in the theory part for these three steps. For bilinear interpolation, Matlab has the function `interp2`. For efficiency first accumulate all projected point positions in arrays, and then call `interp2` once. Note that the coordinates of the top-left pixel are (0,0).

Straight lines in the scene should now appear as *straight* lines in the image.

(c) Repeat (a) and (b) for `img2.jpg`. For this image, the camera intrinsics are given by

$$K_{d2} = \begin{pmatrix} 279.7 & 0 & 347.3 \\ 0 & 279.7 & 235.0 \\ 0 & 0 & 1 \end{pmatrix},$$

and the radial distortion function is modeled by the polynomial

$$g_{\text{pol}}(r) = 1 - 0.3407r + 0.057r^2 - 0.0046r^3 + 0.00014r^4 \,.$$

This image has been taken with a 180° fisheye lens, resulting in large distortions around the border of the image.

The following tasks are optional and only to be solved if you have time over.

(d) Optimize your code to run in less than 1s.
Hint: get rid of all loops, using point-wise Matlab expressions instead.

(e) Now that your code is fast, play around with the resolution and intrinsic parameters of the virtual image. What are their effects? Try to find intrinsic parameters such that the whole virtual image is defined (no black borders), while retaining as much of the image as possible.

(f) Compute a virtual image I' from `img2.jpg`, which looks like it were taken by a virtual camera at the same position and orientation as `img2.jpg`, but with the lens (= projection function) from `img1.jpg`. For this, proceed as follows:

- un-project (n, m) to obtain $g_{\text{ATAN}}\left(\|\pi(\tilde{\mathbf{X}})\|\right)\pi(\tilde{\mathbf{X}})$ for each projected point,
- use the result of Ex. 2(b) in the theory part to calculate $\pi(\tilde{\mathbf{X}})$,
- distort $\pi(\tilde{\mathbf{X}})$ using $g_{\text{pol}}(r)$,
- re-project to obtain (n_d, m_d) in `img2`, and
- interpolate the intensity of `img2` at the projected position.

Similar methods can be used to compute virtual views of available 360° images, e.g. to interactively view a panorama picture.