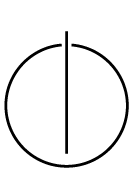

Multiple View Geometry: Exercise Sheet 4

Prof. Dr. Daniel Cremers, Julia Diebold, Robert Maier, TU Munich
http://vision.in.tum.de/teaching/ss2015/mvg2015

Exercise: June 2nd, 2015

Part I: Theory

The following exercises should be **solved at home**. You do not have to hand in your solutions, however, writing it down will help you present your answer during the tutorials.

Image Formation

1. 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.

2. Consider a 3D reconstruction of a single point $\mathbf{p} = (0\ 0\ 4)^\top$. The point is observed by two cameras given by the following projection matrices:

$$\mathbf{P}_1 = \begin{pmatrix} 1 & 0 & 0 & -3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \qquad \mathbf{P}_2 = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

   Compute the images $\tilde{\mathbf{p}}_\mathbf{1}$ and $\tilde{\mathbf{p}}_\mathbf{2}$ of the point $\mathbf{p}$ after projection for camera 1 and 2, respectively.

Radial Distortion

A general projection model for radially distorted cameras is generic projection followed by a non-linear transformation of the radius for each image point.
The calibrated projection function $\pi_1 \colon \mathbb{R}^3 \to \Omega_1 \subset \mathbb{R}^2$ projects a 3D point $\mathbf{p}$ in the camera coordinate system to pixel coordinates $\tilde{\mathbf{p}}$ and is given by

$$\tilde{\mathbf{p}} := \pi_1(\mathbf{p}) = \mathbf{K} \cdot \begin{pmatrix} f(||\pi(\mathbf{p})||) \cdot \pi(\mathbf{p}) \\ 1 \end{pmatrix}. \tag{1}$$

Here, π denotes the generic perspective projection $\pi((x, y, z)^T) := (\frac{x}{z}, \frac{y}{z})^T$, and $\mathbf{K} \in \mathbb{R}^{3\times3}$ is the intrinsic parameter matrix. The function $f \colon \mathbb{R}^+ \to \mathbb{R}^+$ determines the radial distortion factor ($r := ||\pi(\mathbf{p})|| =$ distance to principal point), and is typically approximated by some parametric function (e.g. a polynomial).

1. can this model be used for lenses with a field of view of more than $180°$?

2. what might be the advantage of using only even powers of r, i.e., $f(r) = 1 + a_1 r^2 + a_2 r^4$, as supposed to a more general fourth order polynomial $f(r) = 1 + a_1 r + a_2 r^2 + a_3 r^3 + a_4 r^4$?

Part II: Practical Exercises

This exercise is to be solved **during the tutorial**.

1. **Twist-coordinates**

   (a) Write a function which takes a vector $w \in \mathbb{R}^3$ as input and returns its corresponding element $R = e^{\hat{w}} \in SO(3) \subset \mathbb{R}^{3\times 3}$ from the Lie group. Hence, the function will be a concatenation of the hat operator $\hat{}: \mathbb{R}^3 \to so(3)$ and the exponential mapping.

   (b) Implement another function which performs the corresponding inverse transformation and test the two functions on some examples.

   (c) Implement similar functions which calculate the transformation for twists.
   I.e. from $\xi \in \mathbb{R}^6$ to $e^{\hat{\xi}} \in SE(3) \subset \mathbb{R}^{4x4}$ and the other way around.

   (d) How can you use Matlab's built-in functions `expm` and `logm` to achieve the same functionality (your solutions to (a)-(c) should *not* use these functions)?

2. **Image Formation**

   Consider the 3D model `model.off` from the second exercise sheet (contained in the package `ex2.zip`) and a camera centered at $C = (0, 0, -1)^\top$ with focal length $f = 1$.

   (a) Compute the perspective projection of the model using a homogeneous projection matrix. To this end, you need to transform the list of vertices returned by `openOFF` into homogeneous coordinates.

   (b) Consider a parallel projection where the projection rays are parallel to the z-axis. What is the corresponding projection matrix? Use this matrix to project the model onto the image plane.

3. **Radial Distortion**

   In this exercise you will compute a rectified image from a radially distorted image. Given an image $I_1 \colon \Omega_1 \to \mathbb{R}$ with calibrated projection function $\pi_1 \colon \mathbb{R}^3 \to \Omega_1$, this corresponds to computing a new, virtual image $I_{u_1} \colon \Omega_{u_1} \to \mathbb{R}$ with identical focal point and a (arbitrarily defined) pinhole projection function $\pi_{u_1} \colon \mathbb{R}^3 \to \Omega_{u_1}$.

   (a) Download `ex4.zip`, containing two images. The projection function π_1 of `img1.jpg` is calibrated according to Eq. (1), with

   $$\mathbf{K}_1 = \begin{pmatrix} 388.6 & 0 & 343.7 \\ 0 & 389.4 & 234.6 \\ 0 & 0 & 1 \end{pmatrix},$$

   $$f_1(r) = \frac{1}{0.926r} \operatorname{atan}\left(2r \tan\left(\frac{0.926}{2}\right)\right).$$

   Load the image using `I1 = imreadbw('img1.jpg');`, and display it using `imagesc(I1);`. Change the colormap to grayscale, using `colormap gray;`

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

(b) Compute a virtual, rectified image I_{u_1} of dimensions 1024×768, with a projection function according to a pinhole camera model and intrinsic parameters

$$\mathbf{K}_u := \begin{pmatrix} 250 & 0 & 512 \\ 0 & 250 & 384 \\ 0 & 0 & 1 \end{pmatrix}$$

The intensity $I_{u_1}(\tilde{\mathbf{p}})$ at $\tilde{\mathbf{p}} \in \Omega_{u_1}$ is computed by first un-projecting $\tilde{\mathbf{p}}$ (assuming a fixed depth of 1), projecting it into I_1, and interpolating the intensity at the projected position, that is

$$I_{u_1}(\tilde{\mathbf{p}}) = I_1(\pi_1(K_u^{-1} \begin{pmatrix} \tilde{\mathbf{p}} \\ 1 \end{pmatrix}))$$

Use `interp2` for bilinear interpolation (for efficiency first accumulate all projected point positions in temporary arrays, and then call `interp2` once). Note that the coordinates of the top-left pixel are (0,0).

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

(c) Repeat (1) and (2) for `img2.jpg`. For this image, the projection function is given by

$$\mathbf{K}_2 = \begin{pmatrix} 279.7 & 0 & 347.3 \\ 0 & 279.7 & 235.0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$f_2(r) = 1 - 0.3407r + 0.057r^2 - 0.0046r^3 + 0.00014r^4$$

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

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

(e) Optional: 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.

Info:
The first distortion model for `img1.jpg` is called the FOV or ATAN model, and is used e.g. in the open-source implementation of PTAM (Parallel Tracking and Mapping). Its primary advantage is that it is invertible in closed form. A polynomial approximation of f (of degree 4, as used for `img2.jpg`) is often more general, but cannot be inverted in closed form; making it unsuited for some applications. Which model is best generally depends on the specific lens and intended application.

Neither formulation allows for a field of view of more than $180°$. For cameras with a field of view of more than $180°$, a different projection (e.g. stereographic or spherical projection) has to be used.