



Multiple View Geometry: Exercise Sheet 8

Prof. Dr. Daniel Cremers, Christiane Sommer, Rui Wang
Computer Vision Group, TU Munich

<http://vision.in.tum.de/teaching/ss2017/mvg2017>

Exercise: July 6th, 2017

Part I: Theory

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

Download the ICRA 2013 paper *Robust Odometry Estimation for RGB-D Cameras* by Kerl, Sturm and Cremers from the *Publications* sections on our webpage.¹ Read the paper and focus in particular on *III. Direct Motion Estimation*.

1. Image Warping

- Look at the warping function $\tau(\xi, \mathbf{x})$ in Eq. (9). What do $\tau(\xi, \mathbf{x})$ and $r_i(\xi)$ look like at $\xi = \mathbf{0}$?
- Prove that the derivative of $r_i(\xi)$ w.r.t. ξ at $\xi = \mathbf{0}$ is

$$\left. \frac{\partial r_i(\xi)}{\partial \xi} \right|_{\xi=\mathbf{0}} = \frac{1}{z} \begin{pmatrix} I_x f_x & I_y f_y \end{pmatrix} \begin{pmatrix} 1 & 0 & -\frac{x}{z} & -\frac{xy}{z} & z + \frac{x^2}{z} & -y \\ 0 & 1 & -\frac{y}{z} & -z - \frac{y^2}{z} & \frac{xy}{z} & x \end{pmatrix} \Bigg|_{(x,y,z)^\top = \pi^{-1}(\mathbf{x}_i, Z_1(\mathbf{x}_i))}$$

To this end, apply the chain rule multiple times and use the following identity:

$$\left. \frac{\partial T(g(\xi), \mathbf{p})}{\partial \xi} \right|_{\xi=\mathbf{0}} = (\text{Id}_3 \quad -[\mathbf{p}]_\times) \in \mathbb{R}^{3 \times 6}.$$

2. Image Pyramids

In order to handle large translational and rotational motions, a coarse-to-fine scheme is applied in the paper. To go from one level l to $l + 1$, the images $I^{(l)}$ (intensity) and $D^{(l)}$ (depth) are downsampled by averaging over intensities or valid depth values, respectively:

$$I^{(l+1)}(n, m) := \frac{1}{4} \cdot \sum_{n', m' \in O(n, m)} I^{(l)}(n', m')$$
$$O(n, m) = \{(2n, 2m), (2n + 1, 2m), (2n, 2m + 1), (2n + 1, 2m + 1)\}$$
$$D^{(l+1)}(n, m) := \frac{1}{|O_d(n, m)|} \cdot \sum_{n', m' \in O_d(n, m)} D^{(l)}(n', m')$$
$$O_d(n, m) = \{(n', m') \in O(n, m) : D(n', m') \neq 0\}$$

How does the camera matrix K change from level l to $l + 1$? Write down $f_x^{(l+1)}$, $f_y^{(l+1)}$, $c_x^{(l+1)}$ and $c_y^{(l+1)}$ in terms of $f_x^{(l)}$, $f_y^{(l)}$, $c_x^{(l)}$ and $c_y^{(l)}$.

¹<http://vision.in.tum.de/publications>

3. Optimization for Normally Distributed $p(r_i)$

- (a) Confirm that a normally distributed $p(r_i)$ with a uniform prior on the camera motion leads to normal least squares minimization. To this end, insert

$$p(r_i|\xi) = p(r_i) = A \exp\left(-\frac{r_i^2}{\sigma^2}\right)$$

into Eq. (15) (use $p(\xi) = \text{const}$ there) and show that

$$\xi_{\text{MAP}} = \arg \min_{\xi} \sum_i r_i(\xi)^2 .$$

- (b) Explicitly show that the weights

$$w(r_i) = \frac{1}{r_i} \frac{\partial \log p(r_i)}{\partial r_i}$$

are constant for normally distributed $p(r_i)$.

- (c) Show that in the case of normally distributed $p(r_i)$ the update step $\Delta\xi$ can be computed as

$$\Delta\xi = - \left(J^\top J \right)^{-1} J^\top \mathbf{r}(\mathbf{0}) .$$

Part II: Practical Exercises

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

In this exercise you will implement direct image alignment as Gauss-Newton minimization on $SE(3)$. Download the package `ex8.zip` provided on the website. It contains a code framework, test images and the corresponding camera calibration.

1. Implement the function `[Id, Dd, Kd] = downscale(I, D, K, level)` which (recursively) halves the image resolution of the image I , the depth map D and adjusts the corresponding camera matrix K per pyramid level l . For an input frame of dimensions 640×480 ($l = 1$), level 2 corresponds to 320×240 pixels, level 3 corresponds to 160×120 pixels and so on. Use the equations and results obtained in the theory part.
2. Complete the function `r = calcErr(I1, D1, I2, xi, K)` that takes the images and their (assumed) relative pose, and calculates the per-pixel residual $\mathbf{r}(\xi)$ as defined in the slides. \mathbf{r} should be a $n \times 1$ vector, with $n = w \times h$, the number of pixels. Visualize the residual as image for $\xi = \mathbf{0}$.

Hint: perform tests on a coarse version of the image (e.g. 160×120) to make it run faster.

3. Implement the function `[J, r] = deriveNumeric(I1, D1, I2, xi, K)` that differentiates $\mathbf{r}(\xi)$ **numerically** w.r.t. ξ : for each pixel \mathbf{x}_i compute

$$\frac{\partial r_i(\xi)}{\partial \xi} = \left(\frac{r_i((\epsilon \mathbf{e}_1) \circ \xi) - r_i(\xi)}{\epsilon}, \dots, \frac{r_i((\epsilon \mathbf{e}_6) \circ \xi) - r_i(\xi)}{\epsilon} \right)$$

where ϵ is a small value (for Matlab $\epsilon = 10^{-6}$), \mathbf{e}_j is the j 'th unit vector and the operator \circ is defined by

$$\xi_1 \circ \xi_2 := \log(\exp(\xi_1) \cdot \exp(\xi_2)) .$$

\mathbf{J} should be a $n \times 6$ matrix. The per-pixel residuals $\mathbf{r}(\xi)$ are returned as \mathbf{r} .

4. Implement Gauss-Newton minimization for the photometric error

$$E(\xi) = \sum_i r_i(\xi)^2 = \|\mathbf{r}(\xi)\|_2^2$$

according to the theory part. To this end, complete the script `Ex8_Script` in ll. 70 and ll. 75. For an update $\Delta\xi$, compute the updated motion as $\xi_{\text{new}} = \Delta\xi \circ \xi_{\text{old}}$. Use only one pyramid level $l = 3$ (160×120) in the beginning, and then add the others.

5. Implement a function `J = deriveAnalytic(I1, D1, I2, xi, K)` that differentiates $\mathbf{r}(\xi)$ **analytically** w.r.t. ξ . Use the result of the theory part, Exercise 1 (b). The use of this analytical gradient instead of the numeric derivatives in the minimization should result in a significant speed-up.
6. Run your implementation on the provided images using the script `Ex8_Script`.