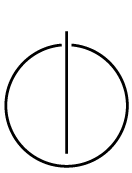

Multiple View Geometry: Exercise Sheet 9

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

Exercise: July 7th, 2015

Part I: Exam Preparation

With regard to the upcoming exam, we can discuss topics covered in the lecture or previous exercises. It might be beneficial to send us your questions to mvg-ss15@vision.in.tum.de before the exercise session.

Part II: Practical Exercises

In this exercise you will continue with the implementation of direct image alignment on SE(3) (Exercise 8). You can either use the provided solution on the website, or use your own. We recommend that you finish Exercise 1 to 4 from Sheet 8.

1. Implement Huber weighting, in order to make your solution robust to outliers. Use GIMP to add some "outliers" to the images, and test your implementation. You will have to implement iteratively re-weighted least-squares (see e.g. Wikipedia). The main idea is to use the Huber norm of the residuals instead of the squared residual:

$$E(\xi) = \sum_{\mathbf{p}_i \in \Omega_{\text{ref}}} \| \underbrace{I_{\text{ref}}(\mathbf{p}_i) - I(\omega(\mathbf{p}_i, D_{\text{ref}}(\mathbf{p}_i), \xi))}_{r_i(\xi)} \|_\delta \tag{1}$$

with

$$\|r\|_\delta := \begin{cases} \frac{r^2}{2\delta} & \text{if } |r| \leq \delta \\ |r| - \frac{\delta}{2} & \text{otherwise.} \end{cases} \tag{2}$$

To minimize this error function using Gauss-Newton, in each iteration it is re-formulated to a **weighted sum of squares**:

$$E(\xi) = \sum_{\mathbf{p}_i \in \Omega_{\text{ref}}} w_i(r_i(\xi)) \cdot r_i^2(\xi) \tag{3}$$

with

$$w(r_i(\xi)) := \begin{cases} 1 & \text{if } |r_i(\xi)| \leq \delta \\ \frac{\delta}{|r_i(\xi)|} & \text{otherwise.} \end{cases} \tag{4}$$

The weighted update is then computed using

$$\delta_\epsilon = -(J_{\mathbf{r}}^T W J_{\mathbf{r}})^{-1} J_{\mathbf{r}}^T W \mathbf{r} \tag{5}$$

where W is a diagonal matrix containing the $w_i(\xi)$; it has to be re-computed every iteration. Use $\delta = 4$ for the Huber norm.

2. Adapt your solution to use **gradient descent** instead of Gauss-Newton to minimize the error function. What is a good stepsize?

3. Adapt your solution to use the **Levenberg-Marquardt** Algorithm for minimization. Test your implementation using the corrupted test images `rgb/*_broken.png` from the solution of exercise 8. Compare results and convergence behavior with Gauss-Newton and gradient descent.