# Multiple View Geometry: Exercise Sheet 7

Prof. Dr. Daniel Cremers, Julia Diebold, Jakob Engel, TU Munich
http://vision.in.tum.de/teaching/ss2014/mvg2014

Exercise: June 2th, 2014

## Part II: Practical Exercises

In this exercise you will implement direct image alignment as Gauss-Newton minimization on SE(3). Download the package `mvg_exerciseSheet_07.zip` provided on the website. It contains a code-framework, test-images and the corresponding camera calibration.

1. Implement a function `[Id,Dd,Kd] = downscale(I,D,K)` which halves the image resolution of the image $I$, the depth map $D$ and adjusts the corresponding Camera matrix $K$ (see slides). For the intensity image, downscaling is performed by averaging the intensity, that is

$$I_d(x,y) := 0.25 \sum_{x',y' \in O(x,y)} I(x',y') \tag{1}$$

   where $O(x,y) = \{(2x,2y),(2x+1,2y),(2x,2y+1),(2x+1,2y+1)\}$.

   For the depth map, downscaling is performed by averaging the *inverse depth* of all valid pixels (invalid depth values are set to zero), that is

$$D_d(x,y) := \left( \left( \sum_{x',y' \in O_d(x,y)} D(x',y')^{-1} \right) \Big/ |O_d(x,y)| \right)^{-1} \tag{2}$$

   where $O_d(x,y) := \{(x',y') \in O(x,y) \colon D(x',y') \neq 0\}$.

2. Implement a 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 (`r` should be a $n \times 6$ vector, where $n$ is the number of valid (with depth and not out of bounds). Visualize the residual as image for $\xi = \mathbf{0}$. *Hint: work on a coarse version of the image (e.g.* $160 \times 120$*) to make it run faster.*

3. Implement a function `J = deriveNumeric(I1, D1, I2, xi, K)` that **numerically** derives $\mathbf{r}(\xi)$. `J` should be a $n \times 6$ matrix) pixels in the image.

4. Implement Gauss Newton minimization for the photometric error $E(\xi) = ||\mathbf{r}(\xi)||_2^2$ as derived in the slides. Use only one pyramid level ($160 \times 120$) in the beginning, and then add the remaining levels. You should get $\xi \approx (-0.002, 0.006, 0.037, -0.029, -0.018, -0.001)^T$

5. Implement a function `J = deriveAnalytic(I1, D1, I2, xi, K)` which **analytically** derives $\mathbf{r}(\xi)$ (see slides). Use it instead of the numeric derivatives in the minimization from the previous task. It should give you a significant speed-up.

6. *Bonus: Add Huber weights.*