Computer Vision II: Multiple View Geometry

Exercise 8: Direct Image Alignment

Robert Maier, Rui Wang
mvg-ss16@vision.in.tum.de

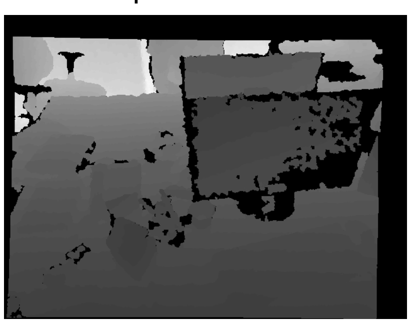
June 30, 2016

Direct Image Alignment

- = “Direct Tracking” / “Dense Tracking” / “Dense Visual Odometry”
- = “Lucas-Kanade Tracking on SE(3)”

ref. image

ref. depth



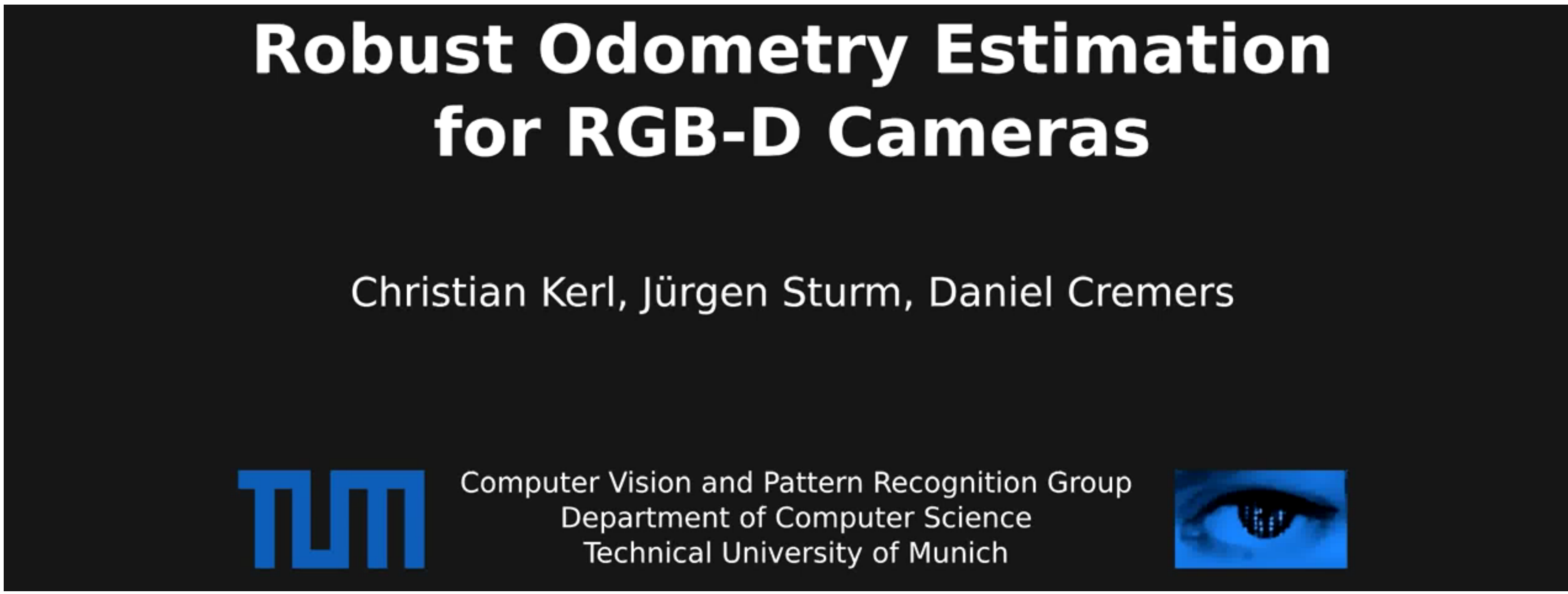
\+

new image

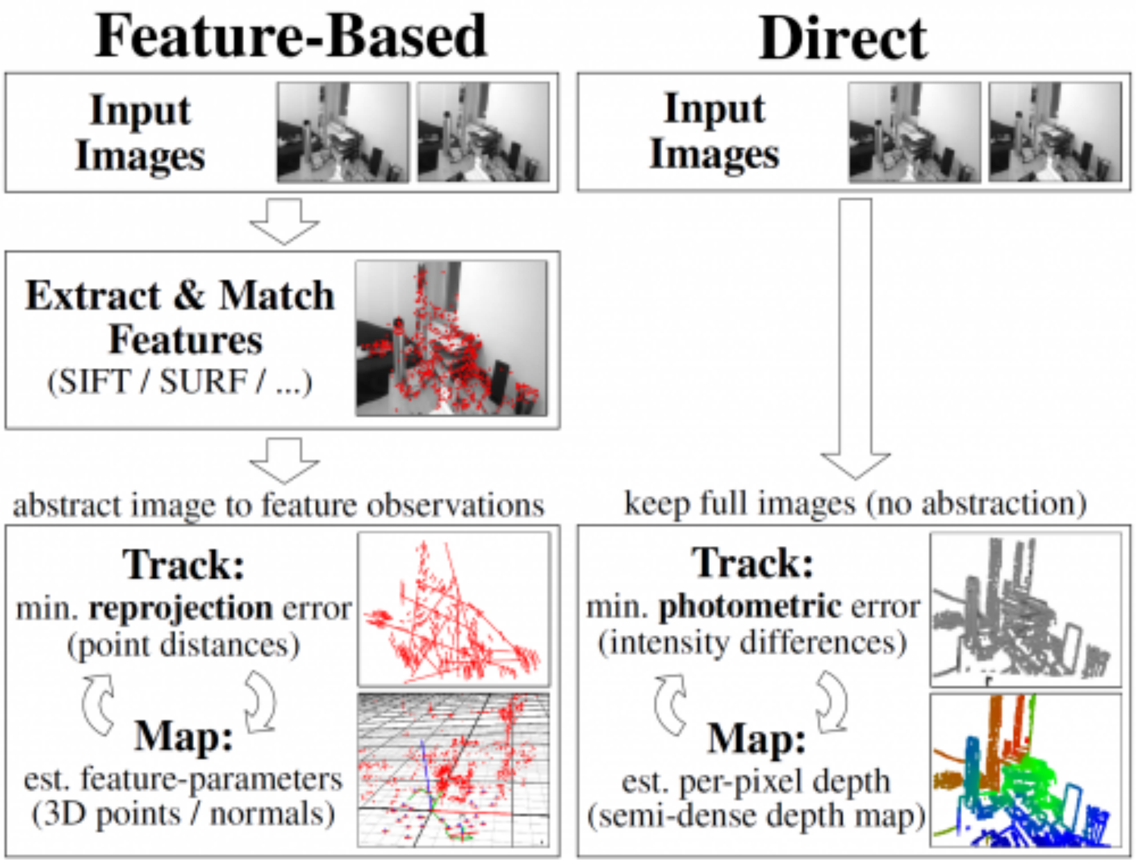
->

Camera pose ξ

Direct Image Alignment

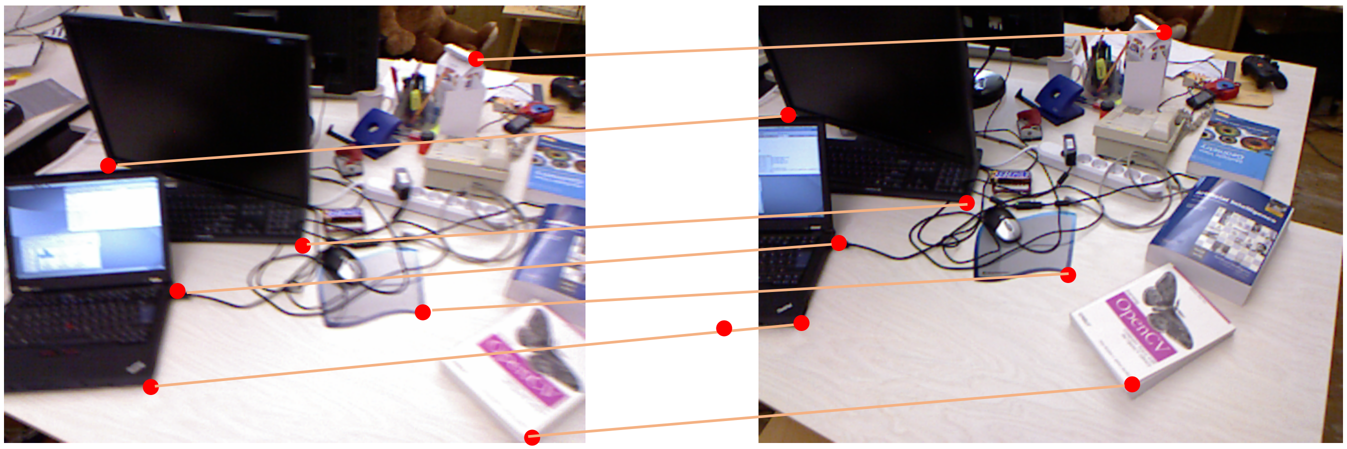


Keypoints, Direct, Sparse, Dense

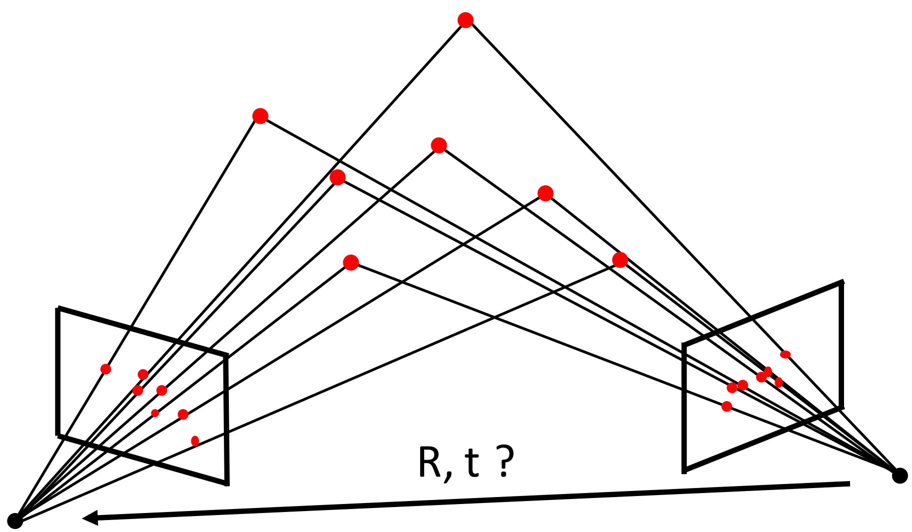


- Sparse: use a small set of selected pixels (keypoints)
- Dense: use all (valid) pixels

Sparse Keypoint-based Visual Odometry

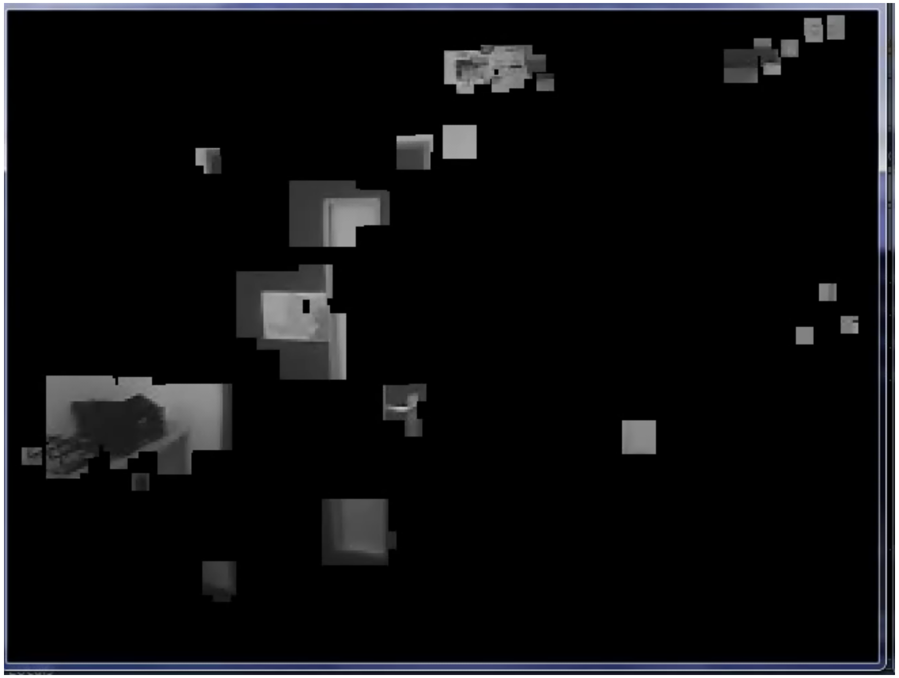


Extract and match keypoints

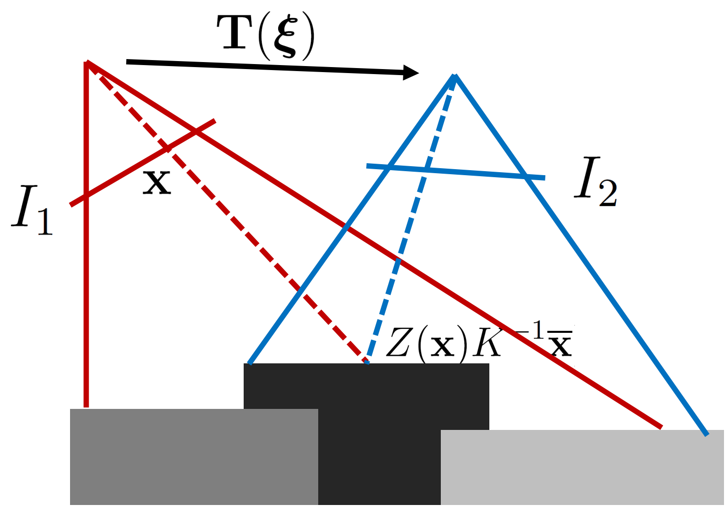


Determine relative camera pose (R, t) from keypoint matches

Problem with Keypoint-based Methods



Dense Direct Image Alignment



- Known pixel depth -> "simulate" RGB-D image from different view point
- Ideally: warped image = image taken from that pose:

$$I_1(\mathbf{x}) = I_2(\pi(\mathbf{T}(\boldsymbol{\xi})Z(\mathbf{x})K^{-1}\overline{\mathbf{x}}))$$

- RGB-D: depth available -> find camera motion!

- Motion representation using the SE(3) Lie algebra
- Non-linear least squares optimization

Direct minimization of photometric error

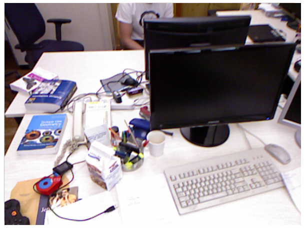
$$E(\xi) = \sum_{\mathbf{p}_i \in \Omega_{\text{ref}}} \left(I_{\text{ref}}(\mathbf{p}_i) - I(\omega(\mathbf{p}_i, D_{\text{ref}}(\mathbf{p}_i), \xi)) \right)^2$$

camera pose

sum over valid ref. pixel

ref. image

new image



ref. depth

$$\omega(\mathbf{p}_i, d, \xi) := \pi\left(K\left(R_\xi K^{-1} \begin{pmatrix} dp_{i,x} \\ dp_{i,y} \\ d \end{pmatrix} + \mathbf{t}_\xi\right)\right)$$

$$\pi(x, y, z) := \begin{pmatrix} x/z \\ y/z \end{pmatrix}$$

$$\begin{pmatrix} R_\xi & \mathbf{t}_\xi \\ \mathbf{0} & 1 \end{pmatrix} := \exp(\widehat{\xi})$$

$\omega(\mathbf{p}_i, d, \xi)$ “warps” a pixel from ref. image to new image

Gauss-Newton

$$E(\xi) = \sum_{\mathbf{p}_i \in \Omega_{\text{ref}}} \left(I_{\text{ref}}(\mathbf{p}_i) - I(\omega(\mathbf{p}_i, D_{\text{ref}}(\mathbf{p}_i), \xi)) \right)^2$$

- Solved using the **Gauss-Newton** algorithm using left-multiplicative increments on SE(3):

$$\xi_1 \circ \xi_2 := \log(\exp(\widehat{\xi_1}) \cdot \exp(\widehat{\xi_2}))^\vee \quad \neq \xi_1 + \xi_2$$
$$\neq \xi_2 \circ \xi_1$$

- **Intuition:** iteratively solve for $\nabla E(\xi) = 0$ by approximating $\nabla E(\xi)$ linearly (i.e. by approximating $E(\xi)$ quadratically)

- Using **coarse-to-fine** pyramid approach

Gauss-Newton

$$E(\xi) = \sum_{\mathbf{p}_i \in \Omega_{\text{ref}}} \underbrace{\left(I_{\text{ref}}(\mathbf{p}_i) - I(\omega(\mathbf{p}_i, D_{\text{ref}}(\mathbf{p}_i), \xi))\right)^2}_{=: \, r_i^2(\xi)}$$

1. „Linearize" **r** on Manifold around current pose $\xi^{(n)}$:

$$\mathbf{r}(\xi) \approx \underbrace{\mathbf{r}(\xi^{(k)})}_{\mathbf{r}_0 \in R^n} + \underbrace{\left.\frac{\partial \mathbf{r}(\epsilon \circ \xi^{(k)})}{\partial \epsilon}\right|_{\epsilon=0}}_{J_\mathbf{r} \in R^{n \times 6}} \cdot \underbrace{(\xi \circ (\xi^{(k)})^{-1})}_{\delta_\xi}$$

2. Solve for $\nabla E(\xi) = 0$

$$E(\xi) = ||\mathbf{r}_0 + J_\mathbf{r} \cdot \delta_\xi||_2^2 = \mathbf{r}_0^T \mathbf{r}_0 + 2\delta_\xi^T J_\mathbf{r}^T \mathbf{r}_0 + \delta_\xi^T J_\mathbf{r}^T J_\mathbf{r} \delta_\xi$$

$$\nabla E(\xi) = 2 J_\mathbf{r}^T \mathbf{r}_0 + 2 J_\mathbf{r}^T J_\mathbf{r} \delta_\xi = 0 \quad \Rightarrow \quad \delta_\xi = -(J_\mathbf{r}^T J_\mathbf{r})^{-1} J_\mathbf{r}^T \mathbf{r}_0$$

3. Apply $\xi^{(k+1)} = \delta_\xi \circ \xi^{(k)}$
4. Iterate (until convergence)

Gauss-Newton

$$E(\xi) = \sum_{\mathbf{p}_i \in \Omega_{\text{ref}}} \underbrace{\left(I_{\text{ref}}(\mathbf{p}_i) - I(\omega(\mathbf{p}_i, D_{\text{ref}}(\mathbf{p}_i), \xi))\right)^2}_{=: \, r_i^2(\xi)}$$

- Jacobian J_r: partial derivatives

Requires gradient of residual:

$$\left.\frac{\partial r_i(\epsilon \circ \xi^{(k)})}{\partial \epsilon}\right|_{\epsilon=0} = -\frac{1}{z'} \begin{pmatrix} \nabla I_x f_x & \nabla I_y f_y \end{pmatrix} \begin{pmatrix} 1 & 0 & -\frac{x'}{z'} & -\frac{x'y'}{z'} & (z' + \frac{x'^2}{z'}) & -y' \\ 0 & 1 & -\frac{y'}{z'} & -(z' + \frac{y'^2}{z'}) & \frac{x'y'}{z'} & x' \end{pmatrix}$$

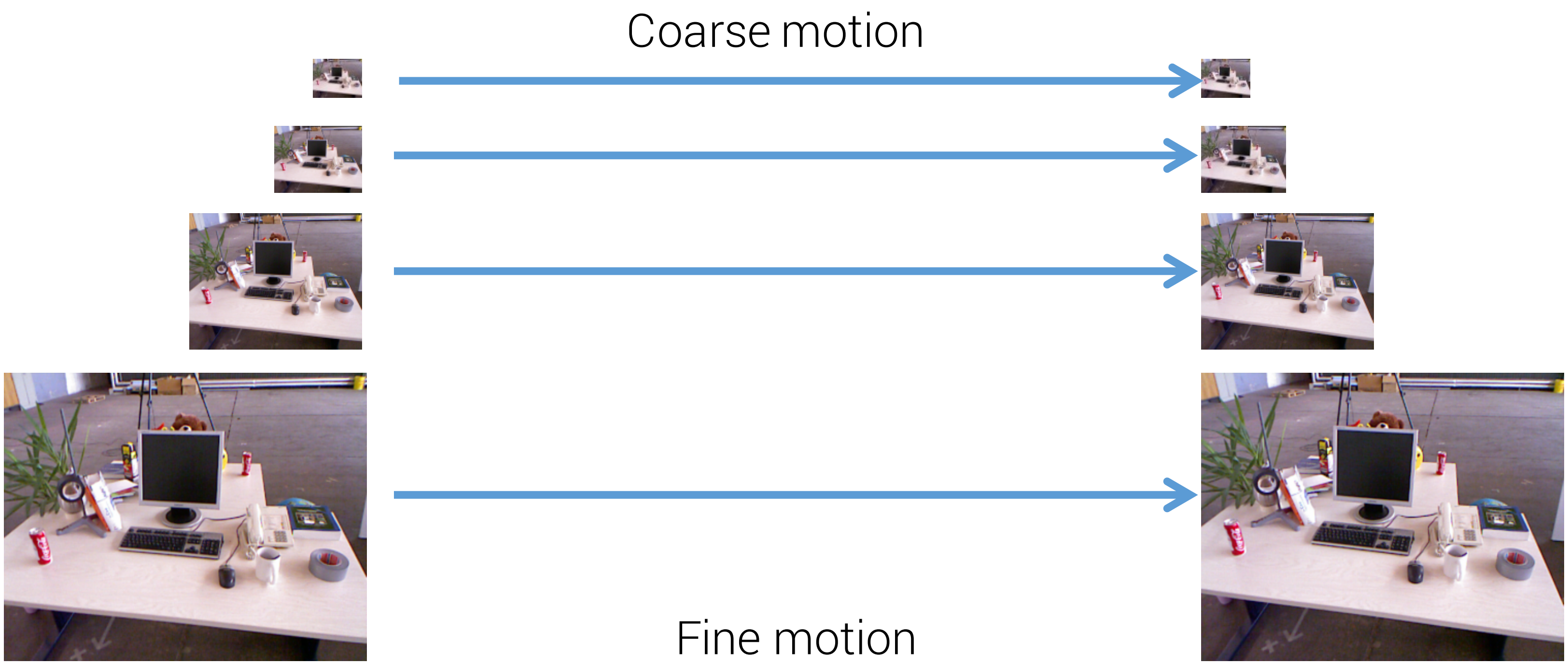
= 1x6 row of $J_{\mathbf{r}}$

with

- $\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} := R_{\xi^{(k)}} K^{-1} \begin{pmatrix} dp_{i,x} \\ dp_{i,y} \\ d \end{pmatrix} + \mathbf{t}_{\xi^{(k)}}$ = warped point (before projection)
- f_x, f_y, K = intrinsic camera calibration
- $\nabla I_x, \nabla I_y$ = image gradients

Coarse-to-Fine

- Adapt size of the neighborhood from coarse to fine



Coarse-to-Fine

- Minimize for down-scaled image (e.g. factor 8, 4, 2, 1) and use result as initialization for next finer level
- Elegant formulation: Downscale image and adjust K accordingly
  - Downscale by factor of 2 (e.g. 640x480 -> 320x240)
  - Camera matrix

$$K = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \quad \text{->} \quad K_{\frac{1}{2}} = \begin{pmatrix} \frac{f_x}{2} & 0 & \frac{c_x+0.5}{2} - 0.5 \\ 0 & \frac{f_y}{2} & \frac{c_y+0.5}{2} - 0.5 \\ 0 & 0 & 1 \end{pmatrix}$$

(assuming discrete pixel (x,y) contains continuous value at (x,y))

Final Algorithm

$\xi^{(0)} = \mathbf{0}$

k = 0

for *level* = 3 ... 1

compute down-scaled images & depthmaps (factor = 2^{level})

compute down-scaled K (factor = 2^{level})

for *i* = 1..20

compute Jacobian $J_{\mathbf{r}} \in R^{n \times 6}$

compute update δ_ξ

apply update $\xi^{(k+1)} = \delta_\xi \circ \xi^{(k)}$

k++; maybe break early if δ_ξ too small or if residual increased

done

done

\+ robust weights (e.g. Huber), see *iteratively reweighted least squares*

\+ *Levenberg-Marquad (LM)* Algorithm