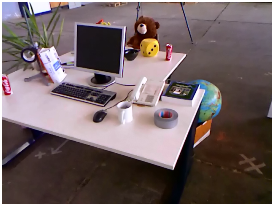
Dense Visual Odometry

Bianca Tost, Maximilian Staab

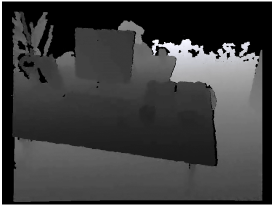
March 30, 2015

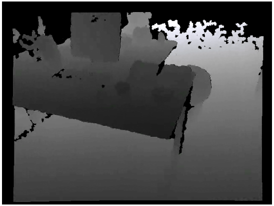
Table of Contents

RGB-Sequence



Depth-Sequence





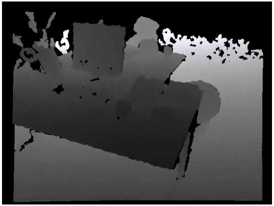
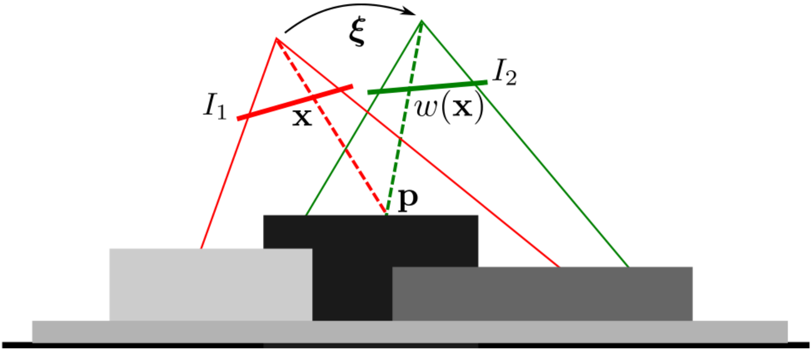
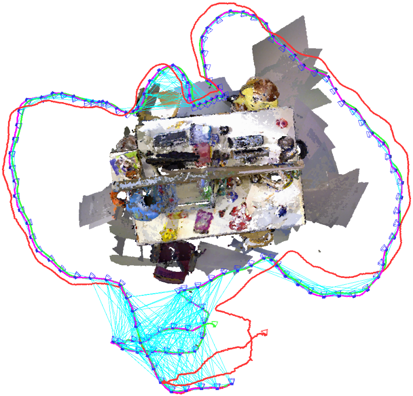


Photo Consistency Assumption

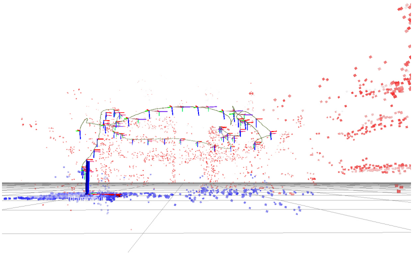


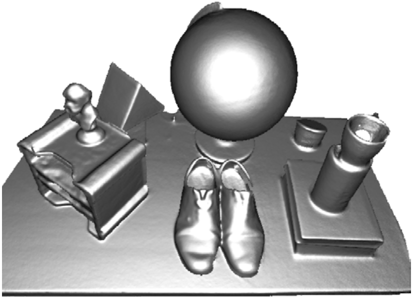
$$I_1(x) = I_2(\tau(\xi, x))$$

Odometry

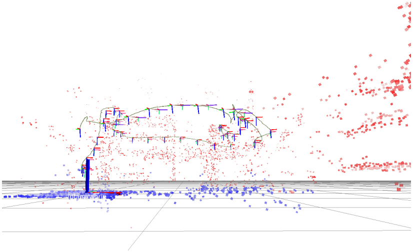


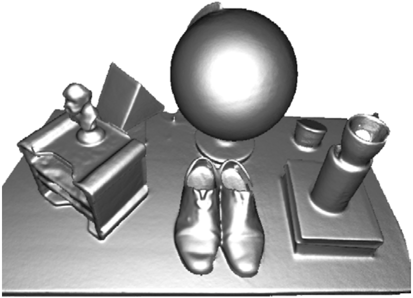
Feature Based vs. Dense Mapping





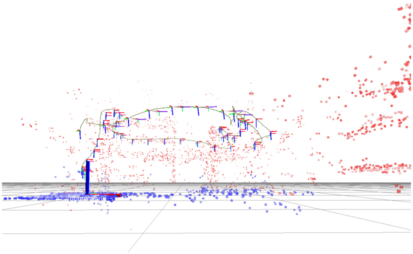
Feature Based vs. Dense Mapping

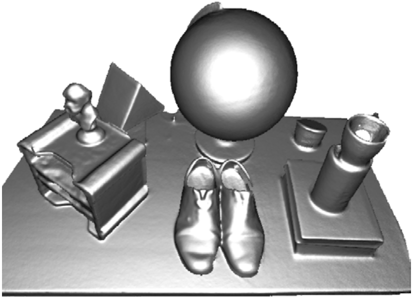




- more accuracy

Feature Based vs. Dense Mapping





- more accuracy
- the map itself is useful

Visual

Visual sensors e. g.:

- RGB camera
- IR depth sensor
- Laser scanner

Hardware

Kinect sensor

- Structured, light based depth sensor (IR-rays)
- framerate 30Hz
- 640 $\times$ 320 $\times$ 11 bit

Geometric Error - Photometric Error

depth sensor data

$\downarrow$

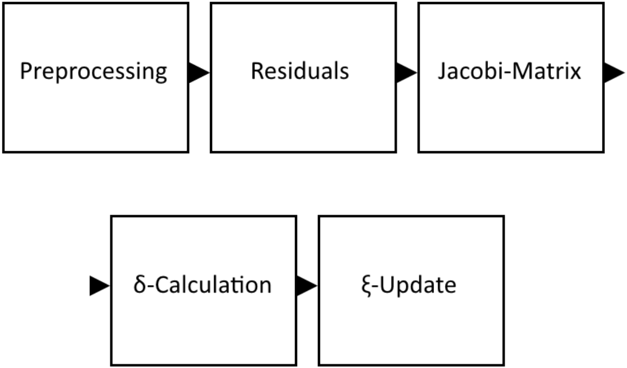
minimize geometric error

intensity sensor data
(RGB camera)

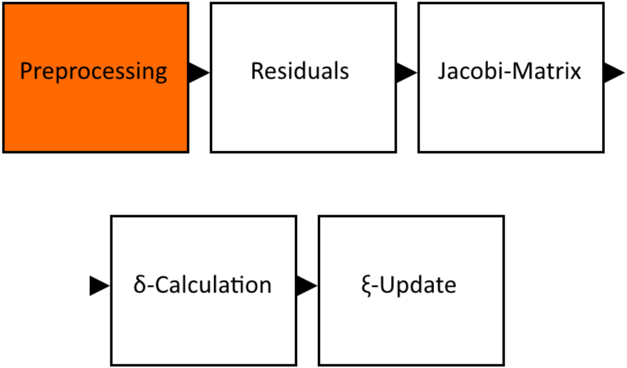
$\downarrow$

minimize photometric error

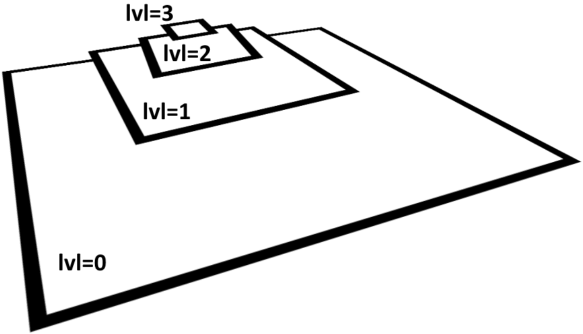
Algorithm


Preprocessing
Residuals
Jacobi-Matrix
δ-Calculation
ξ-Update

Algorithm

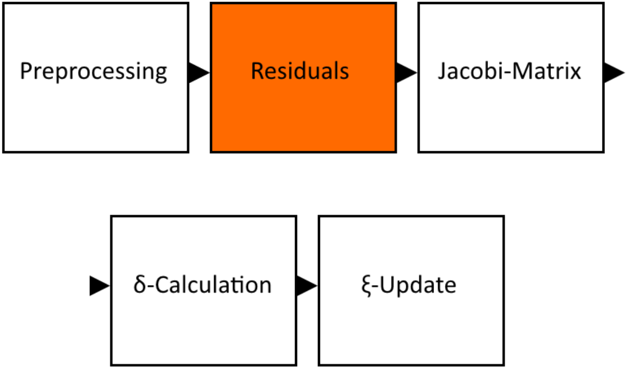

Preprocessing
Residuals
Jacobi-Matrix
δ-Calculation
ξ-Update

Preprocessing

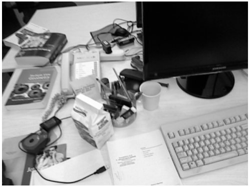


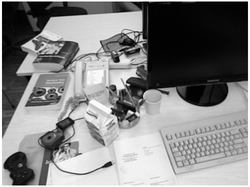
- RGB $\rightarrow$Intensity-Image
- Depth-Image
- Intrinsic Calibration K
- Inverse Calibration IK

Algorithm

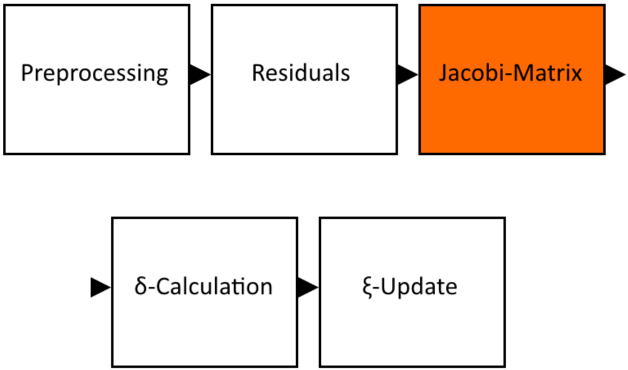

Preprocessing
Residuals
Jacobi-Matrix
δ-Calculation
ξ-Update

Residual





Algorithm


Preprocessing
Residuals
Jacobi-Matrix
δ-Calculation
ξ-Update

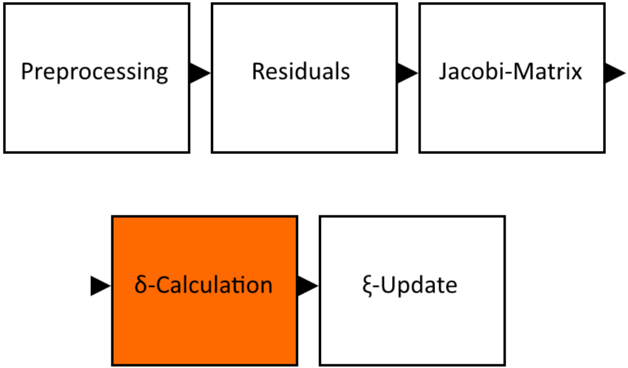
Jacobi

$$\begin{pmatrix} 1 & 0 & -\frac{x'}{z'} & -\frac{x'y'}{z'} & (z' + \frac{x'^2}{z'}) & -y' \\ 0 & 1 & -\frac{y'}{z'} & -(z' + \frac{y'^2}{z'}) & \frac{x'y'}{z'} & x' \end{pmatrix}$$

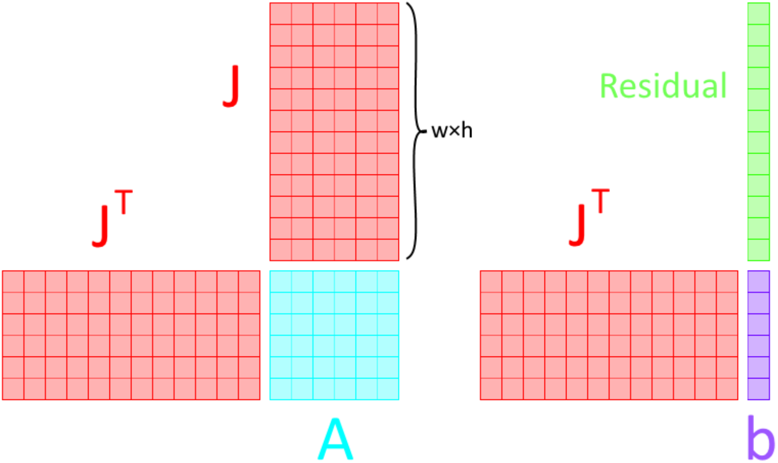
$$\begin{pmatrix} \frac{\nabla I_x f_x}{z'} & \frac{\nabla I_y f_y}{z'} \end{pmatrix}$$

1 Row of n×6 Jacobi Matrix

Algorithm

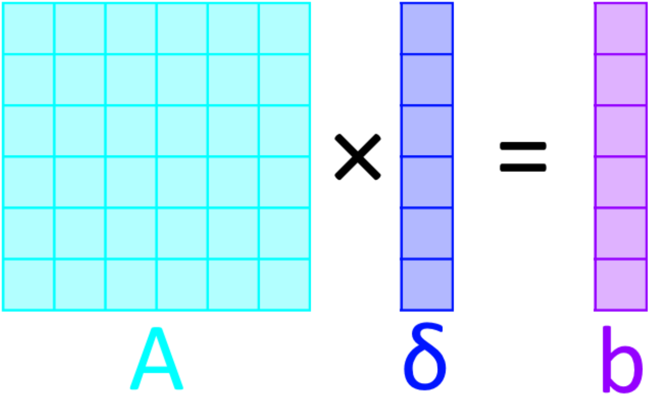

Preprocessing
Residuals
Jacobi-Matrix
δ-Calculation
ξ-Update

A and b

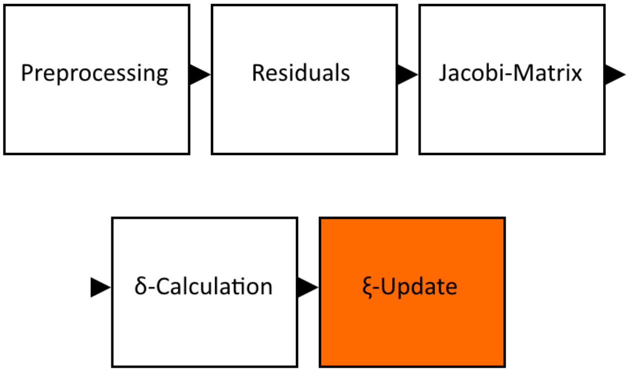


Delta - Solve Linear System

$$\delta_\xi = -(J_{\mathbf{r}}^T J_{\mathbf{r}})^{-1} J_{\mathbf{r}}^T \mathbf{r}_0$$



Algorithm


Preprocessing
Residuals
Jacobi-Matrix
δ-Calculation
ξ-Update

Xi-Update

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

(solved using the **Gauss-Newton** algorithm
using left-multiplicative increments on SE(3):
$\xi_1 \circ \xi_2 := \log(\exp(\widehat{\xi_1}) \cdot \exp(\widehat{\xi_2}))^\vee \neq \xi_1 + \xi_2$
$\neq \xi_2 \circ \xi_1$)

```
xi = Sophus::SE3f::log(Sophus::SE3f::exp(delta)*Sophus::SE3f::exp(xi));
```

Iterate (until convergence)

Shared Memory

Parallel reduction as a part of matrix-matrix- and matrix-vector-multiplication for A and b

Constant Memory

Storage of intrinsic calibration matrix K and its inverse
(Stored once, accesses 8 times per iteration)

Saving Computation Steps

Compute only current frame's pyramids and store it as reference for next step

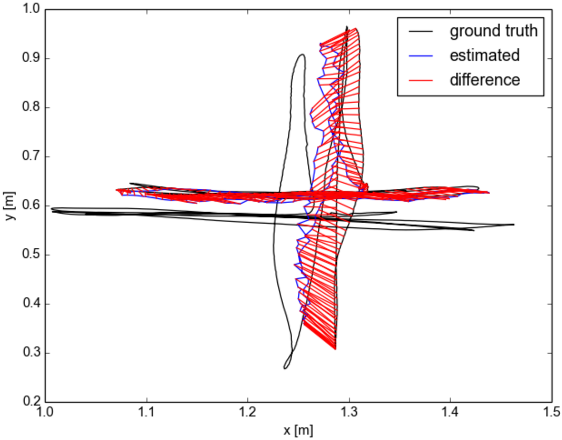
Saving Computation Steps

A is symmetric, so only 21 unique elements of 36 have to be computed

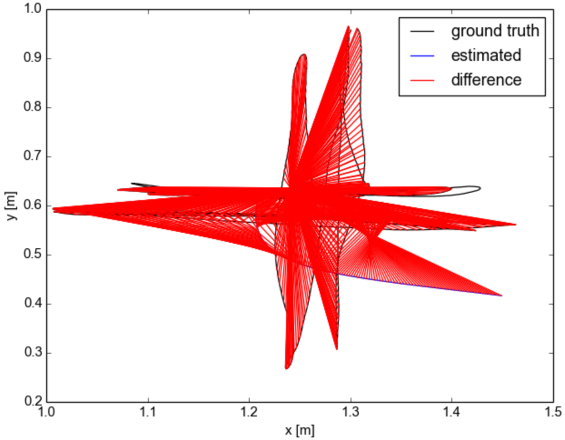
Early Break

Break out of hard coded 20 iterations if delta is below threshold

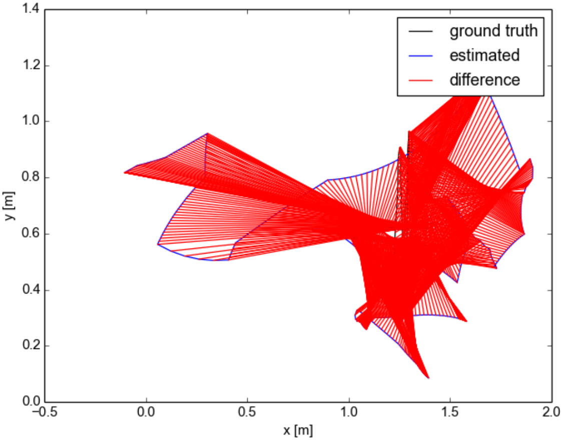
Dataset 'fr1/xyz'



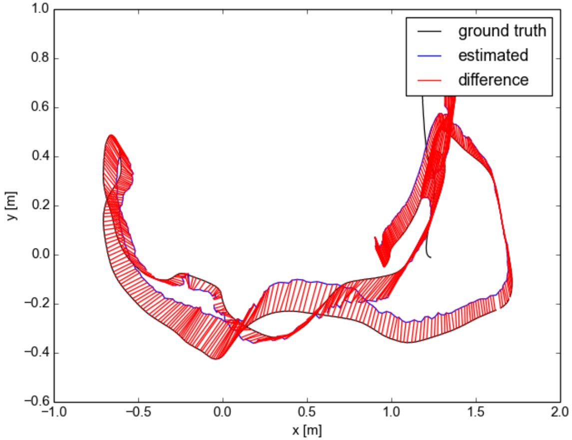
Dataset 'fr1/xyz' - threshold too high



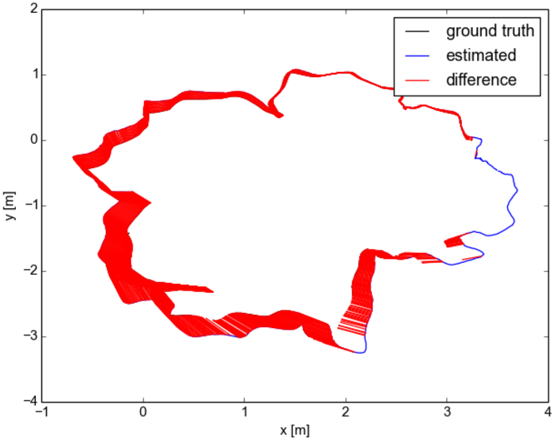
Dataset 'fr1/xyz' - too few pyramid levels



Dataset 'fr1_desk'



Dataset 'fr2_desk'



Computation Time

Alignment only (without frame loading and pyramid computations)

- naive approach: 60ms
- early break: saved 43ms
- constant K and iK: saved 1-2ms if many iterations

Further ideas

- R, t (=xi) in constant memory
- cuBLAS for multiplications

Live-Demo

Questions?

Thank you for your attention!