# Robust Odometry Estimation for RGB-D Cameras

Oskar Carlbaum
Guillermo Gonzalez de Garibay

Technische Universität München
Department of Informatics
Computer Vision Group

April 13, 2016

# Outline

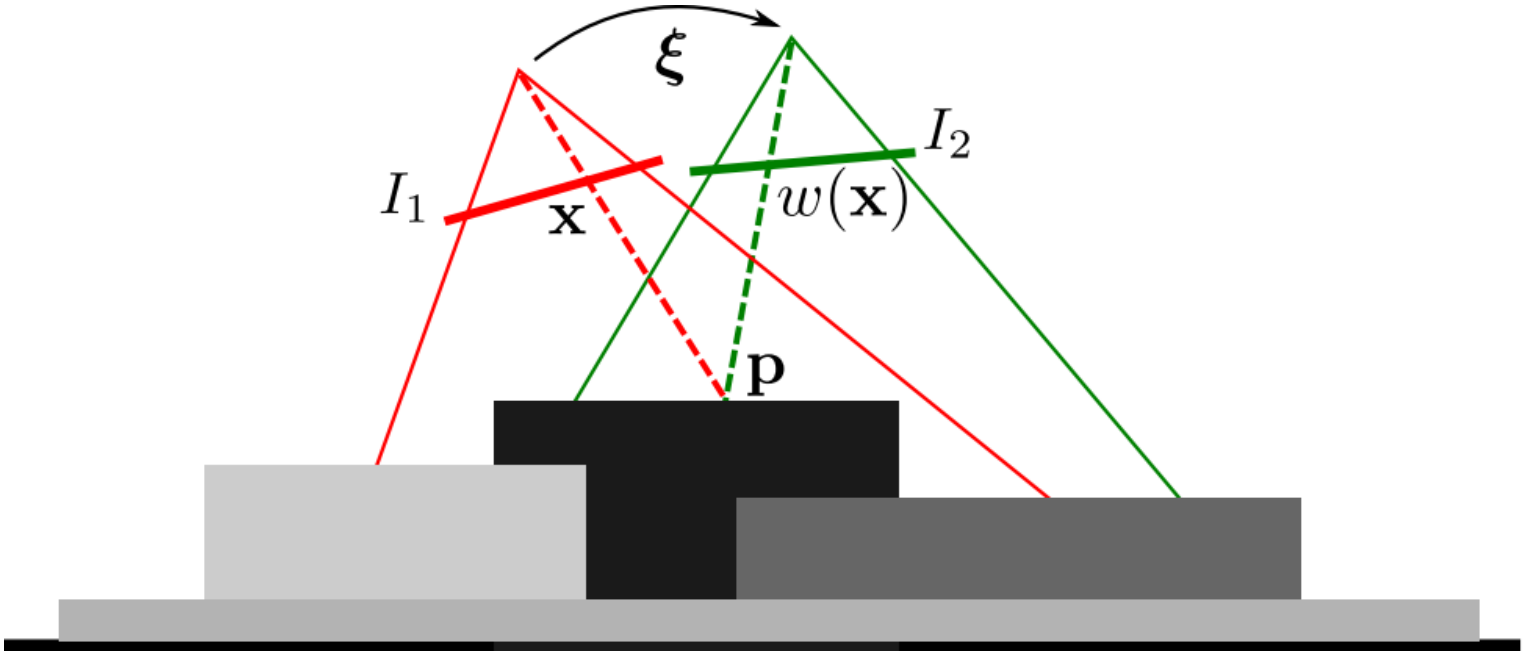
# Outline

# Introduction

Visual Odometry Estimation

- Estimating camera movements from image sequence
- Uses: Robots, Quadcopters, Mars rover
  - Challenges: Requires high frame rates, robustness to outliers
- Sparse vs Dense
  - Sparse: Feature based detectors for tracking (common)
  - Dense: All image information (more robust)
    - RGB-D images

# RGB-D Frames



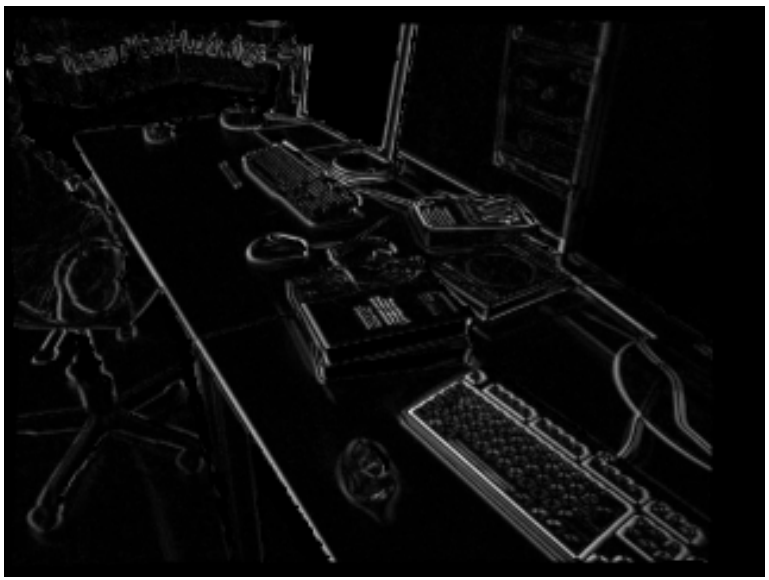

[Kerl et al., 2013]

- Calculate $\xi_i^{i+1}$ for consecutive frames
- Depth information from first frame only
- Align gray images
- Minimize photometric error $\rightarrow$ See next slide

# Photometric error

- $E(\xi) = \sum_i w(r_i)(r_i(\xi))^2$ is to be minimized for consecutive frames
- $r_i(\xi) := I_2(\tau(\xi, \mathbf{x}_i)) - I_1(\mathbf{x}_i)$ (intensity difference after transformation)
  - $I_2$ **not** evaluated at grid points $\rightarrow$ `__texture__` interpolation

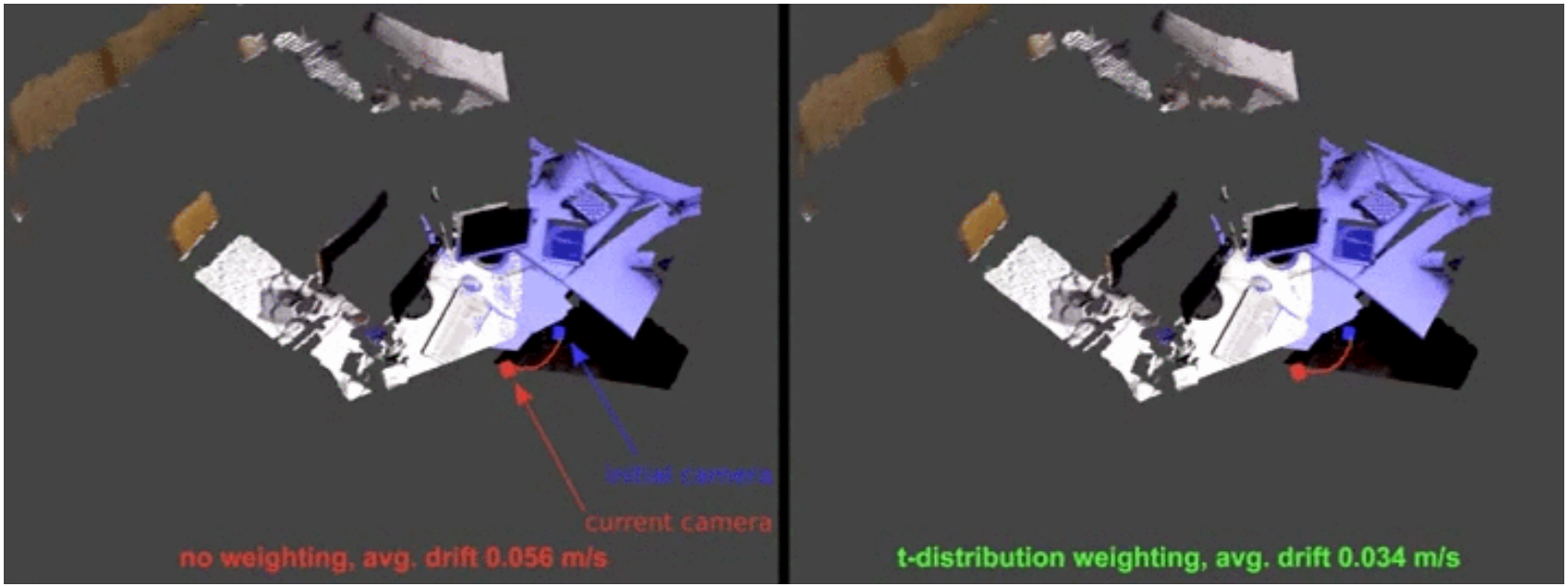
- By linearizing the residuals:
  $\mathbf{r}_{lin}(\xi) = \mathbf{r}(\mathbf{0}) + J\Delta\xi$
- We get to this linear system:
  $J^T W J \Delta\xi = -J^T W \mathbf{r}(\mathbf{0})$



# Twist coordinates

- $\xi$ is just a **minimal** and **smooth** representation of a Rigid Body Motion
- Rigid Body Motion $\equiv$ Rotation + Translation (6DoF)
- Many possible parametrizations ($\geq$ 6 parameters)
  - Most have mathematical singularities
  - $\Longrightarrow$ Use twist coordinates: $\xi = (v_1, v_2, v_3, w_1, w_2, w_3)$
- $\xi \longleftrightarrow (R, T)$ have closed-form expressions

# Reconstruction




[Kerl et al., 2013]

- Accumulate $\xi_i^{i+1}$ to get $\xi_0^t$
- Translation is represented above

# Outline

# Related work

- Paper:
  - Robust Odometry Estimation for RGB-D Cameras [Kerl et al., 2013]
- Previous implementations:
  - MATLAB: Multiple View Geometry lecture, Exercise Sheet 8 [2015]
  - C++/CUDA: This same course in SS14, WS14/15, SS15

# Outline

# Our mission

Goals

- Ludicrous speed
- Code readability
- Fewest libraries
- Robustness
- Comparable results

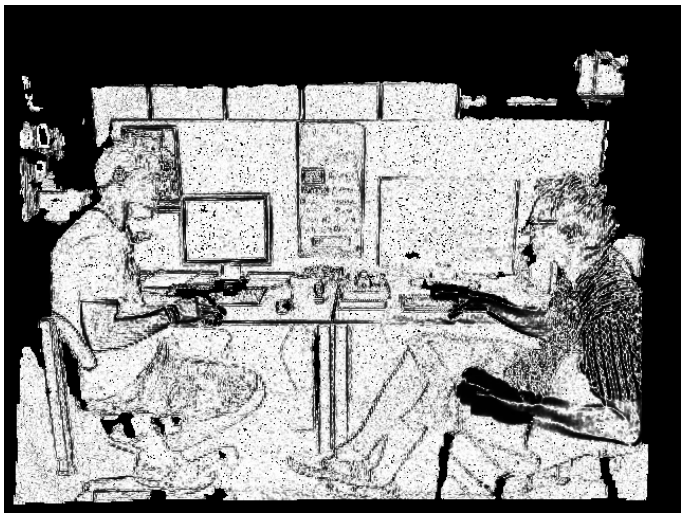
Test material

- TUM Benchmark Datasets
  - RGB-images
  - Depth-images
  - Intrinsic Camera Parameters

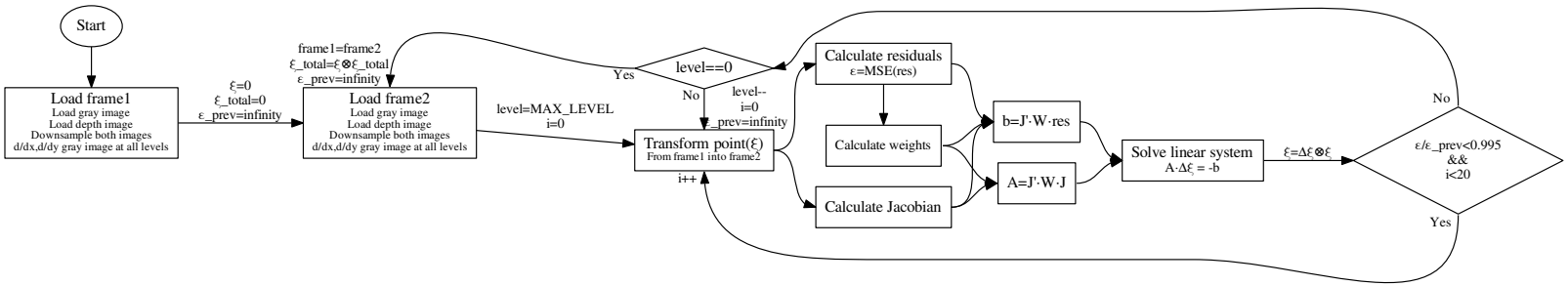
# Robustness

Robustness $\rightarrow$ Weighting

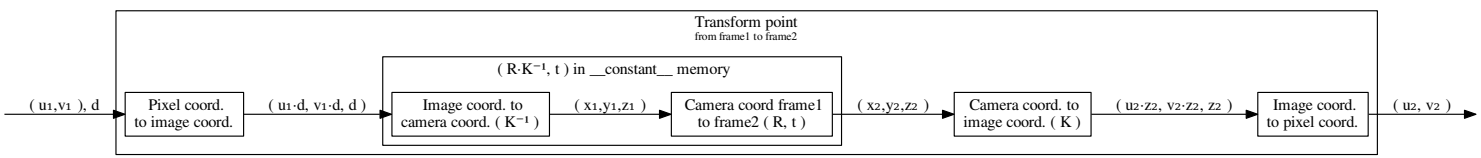
- Movement, blur, reflections, intensity changes
- Student-t distribution $\rightarrow$ Exclude unreliable data



# Flow chart




- Gauss-Newton algorithm
- Gray image and derivatives into `__texture__` memory
  - Interpolation for residuals and Jacobian




- The transformation $(R \cdot K^{-1}, t)$ is loaded into `__constant__` memory
  - $\sim 10^6$ reads

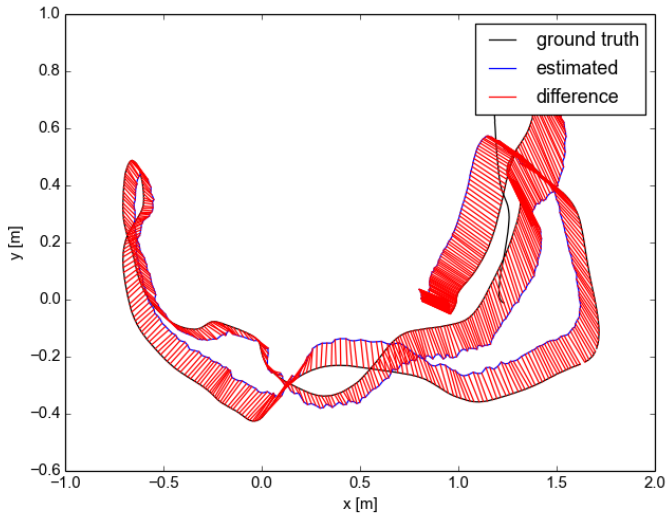
# CUDA

Operations using CUDA:

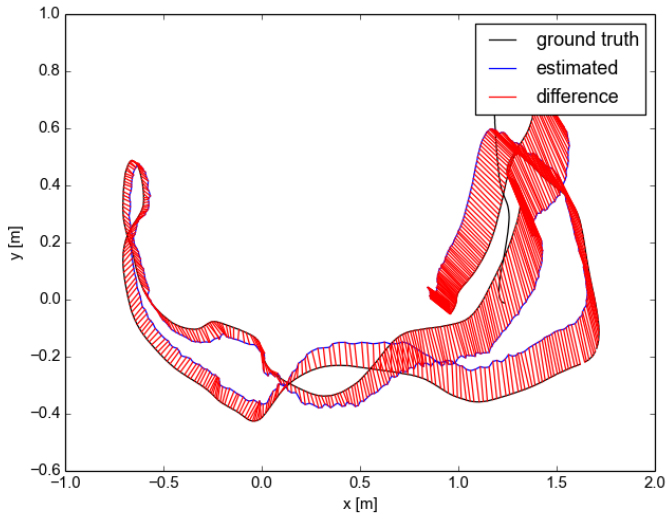
- Image downscaling
- Intensity image derivatives
- Transform points
- Residuals
- Error $\rightarrow$ cuBLAS (reduction)
- Weights $\rightarrow$ cuBLAS (reduction)
- Jacobian
- $J^T W J,\ J^T W \mathbf{r} \rightarrow$ cuBLAS (matrix mult.)

# Outline

# Comparisons - ATE, Desk1



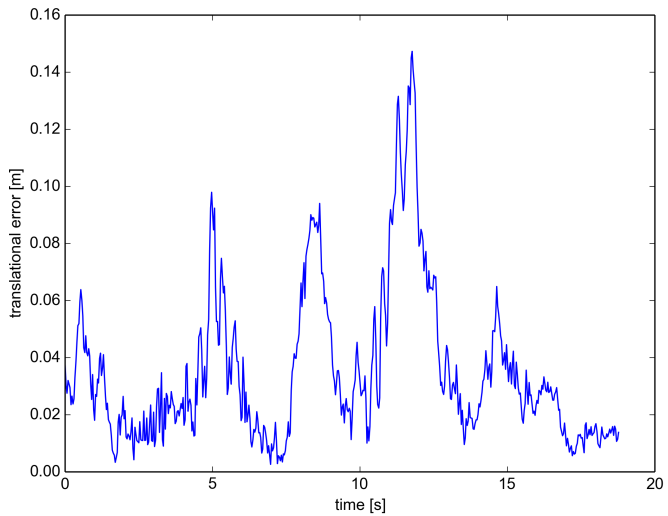

No weights



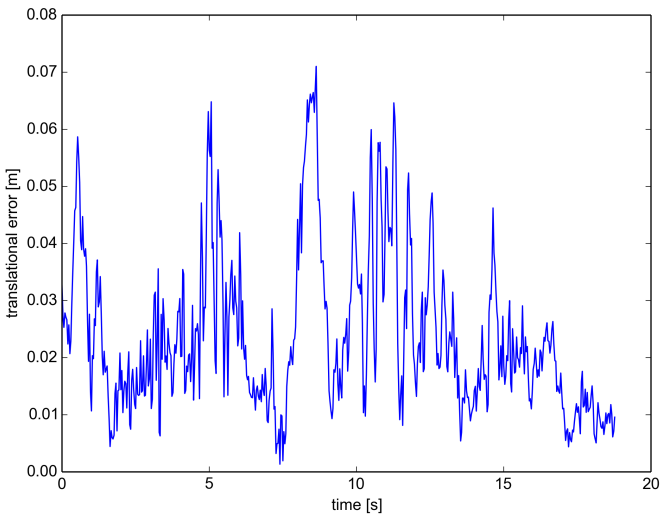

T-dist. weights

Absolute Trajectory Error

# Comparisons - RPE, Desk1




No weights




T-dist. weights

Relative Pose Error

# Results

- Accuracy, TUM-benchmark toolset

| Method | fr1/desk | fr1/desk2 |
|---|---|---|
| Paper: no weights | 0.0551 | 0.1003 |
| Paper: t-dist. weights | 0.0458 | 0.0708 |
| Our: no weights | 0.0449 | 0.0501 |
| Our: t-dist. weights | 0.0278 | 0.0433 |

Note: we take the translational_error.rmse value from the online_evaluation for relative pose error for pose pairs with a distance of 1 second. We are not completely sure if it is the same value as used in the paper.

- Speed - runtime
  - 'Homemade' method
    - $\sim 53.3$ ms / frame (no weights)
    - $\sim 93.4$ ms / frame (t-Dist. weights)
  - cuBLAS
    - $\sim 26.2$ ms / frame (no weights)
    - $\sim 44.7$ ms / frame (t-Dist. weights)

# Outline

# Conclusions

- Works as expected
- Libraries:
  - cuBLAS $\rightarrow$ easy, fast
  - Eigen $\rightarrow$ easy CPU matrix handling
  - Sophus (Lie algebra) $\rightarrow$ replace by 2 functions
- Gaussian for downscaling $\rightarrow$ not noticeable
- $J^T W J$ just at the highest resolution level, takes about half of the execution time, even with cuBLAS

# Improvements

- Motion Prior $\rightarrow$ Velocity Gaussian distribution
  - Constant velocity assumption
  - Inertial sensors
  - ...
- SLAM
  - Robustness against moving objects
- Do not recalculate Jacobian every iteration (compositional image alignment?)

# Outline

## THANKS!

# Questions?

# Bibliography I

[Kerl et al., 2013] Kerl, C., Sturm, J., and Cremers, D. (2013).
Robust odometry estimation for rgb-d cameras.
In *Int. Conf. on Robotics and Automation*.