VISUM Summer School

RGB-D Cameras

Dr. Thomas Whelan

Imperial College London

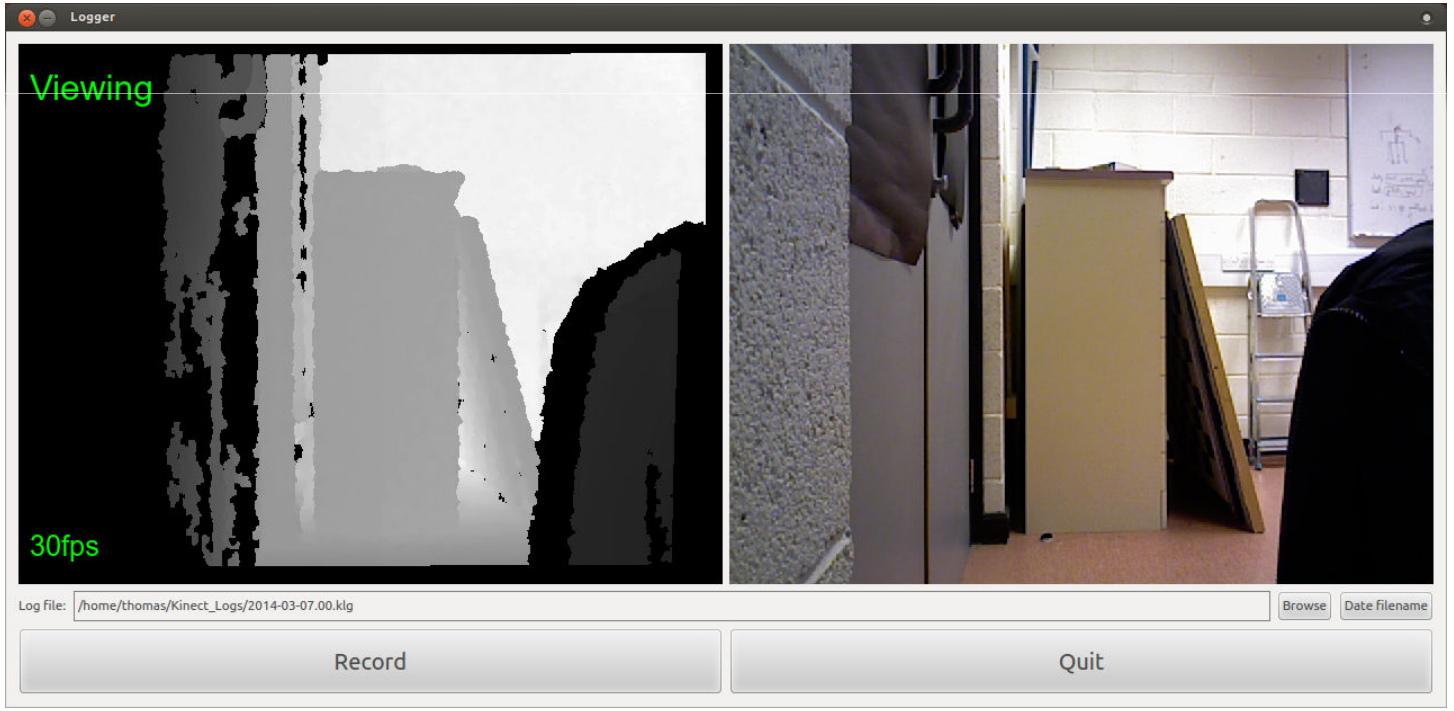
dyson

Outline: First Session

- Hands-on with RGB-D cameras
- RGB-D SLAM
- RGB-D Benchmark

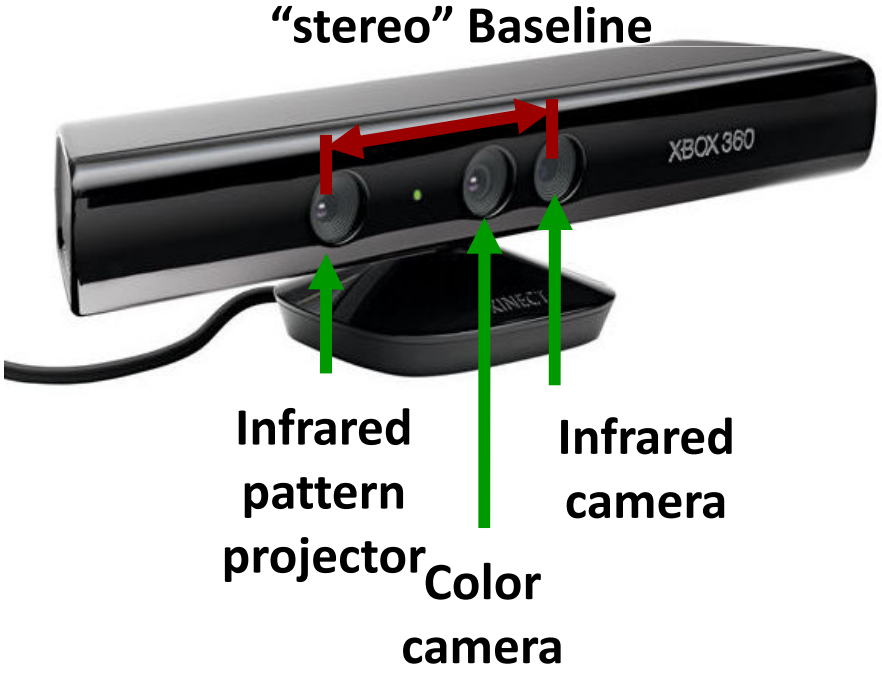
What are RGB-D Sensors?

- Provide color (RGB) and depth (D) images
- Low-cost sensors such as Kinect pushed research and enabled novel approaches



Sensor Principle of Kinect

- Kinect projects a diffraction pattern (speckles) in near-infrared light
- CMOS IR camera observes the scene

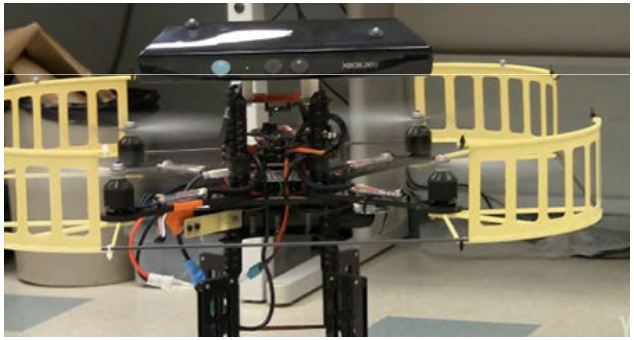


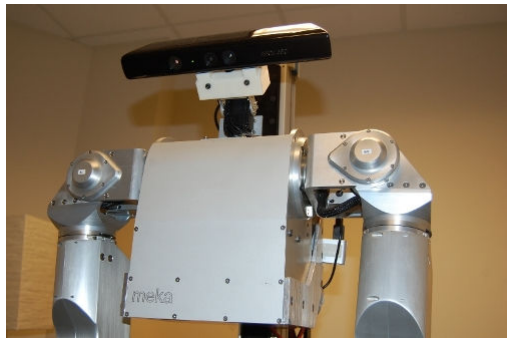
History

- 2005: Developed by PrimeSense (Israel)
- 2006: Offer to Nintendo and Microsoft, both companies declined
- 2007: Alex Kidman becomes new incubation director at Microsoft, decides to explore PrimeSense device. Johnny Lee assembles a team to investigate technology and develop game concepts
- 2008: The group around Prof. Andrew Blake and Jamie Shotton (Microsoft Research) develops pose recognition
- 2009: The group around Prof. Dieter Fox (Intel Labs / Univ. of Washington) works on RGB-D mapping and RGB-D object recognition
- Nov 4, 2010: Official market launch
- Nov 10, 2010: First open-source driver available
- 2011: First programming competitions (ROS 3D, PrimeSense), First workshops (RSS, Euron)
- 2012: First special Issues (JVCI, T-SMC)

Impact of the Kinect Sensor

- Sold >18M units, >8M in first 60 days (Guinness: “fastest selling consumer electronics device). Has become a “standard” sensor in robotics

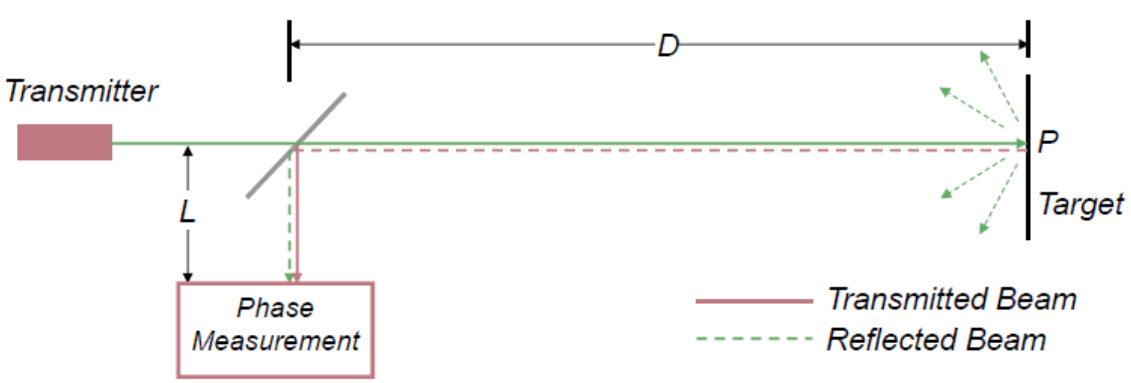



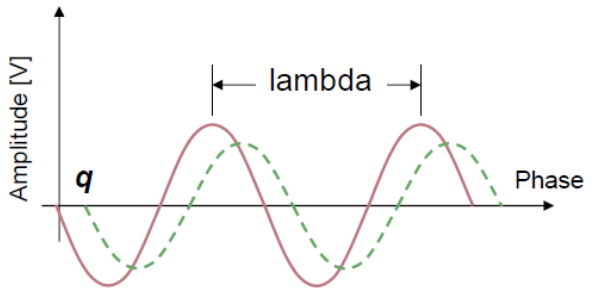
RGB-D Sensor Principles

- 3D laser scanner (Velodyne) + color camera
- Stripe laser + color camera
- Stereo camera (Bumblebee, Videre)
- Projected Texture Stereo (as used in PR2)
- Structured light (Primesense/Kinect)
- Time-of-flight cameras

Laser Scanner

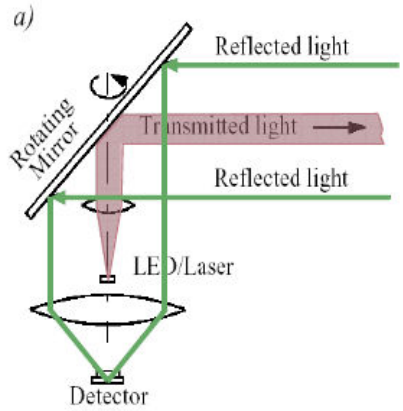
- Classical sensor in robotics
- **Pro:** High precision, wide field of view, safety approved for collision detection
- **Con:** Relatively expensive + heavy



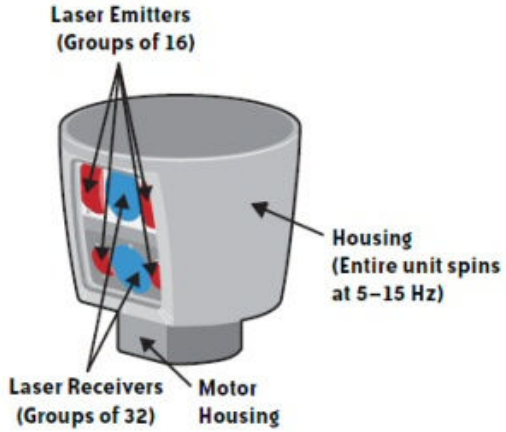


Laser Scanner

- 2D scanners

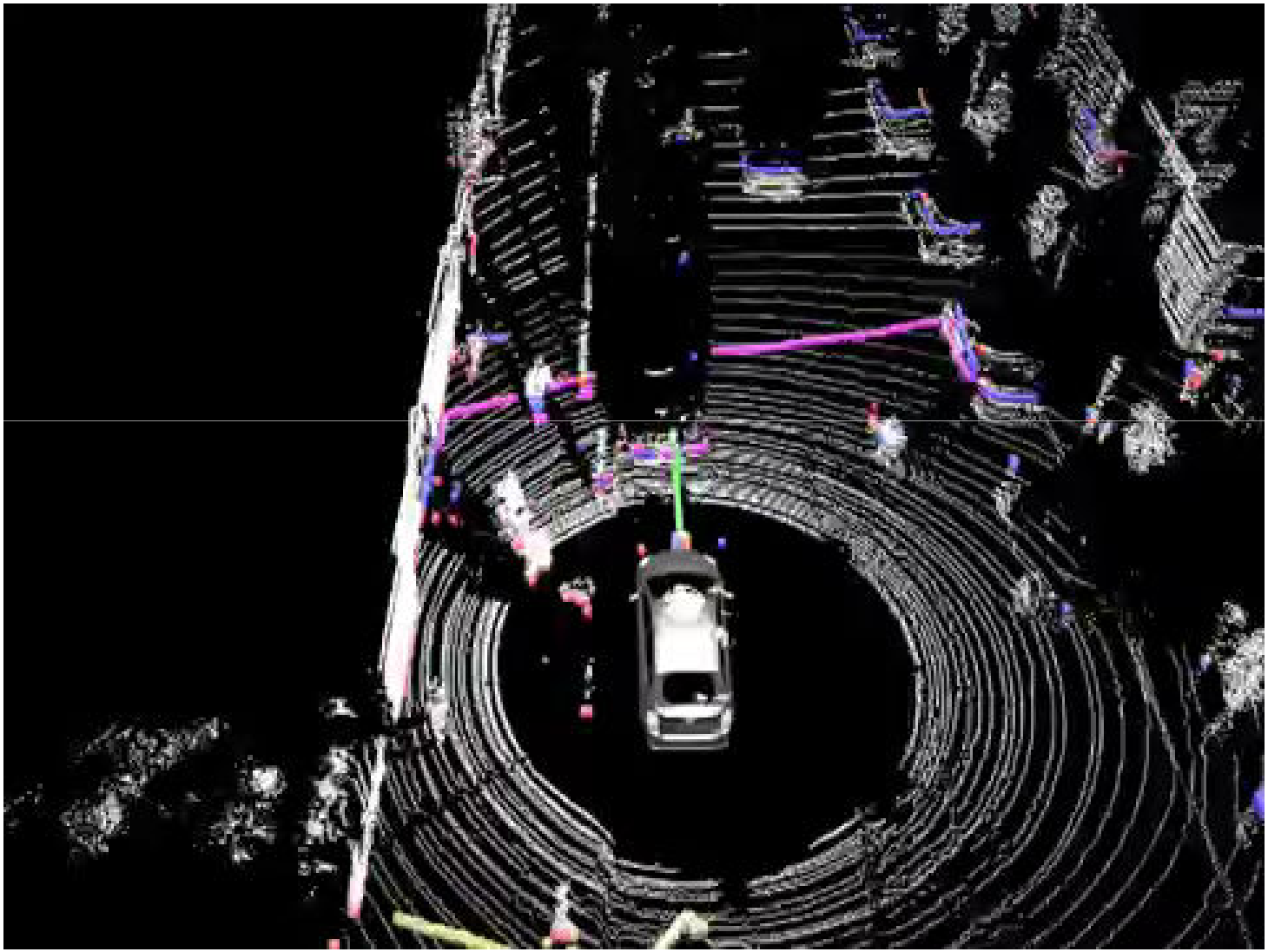


- 3D scanners



Dr. Thomas Whelan, Dyson Robotics Lab, Imperial College London & Dr. Jürgen Sturm, Computer Vision Group, TUM

Velodyne Laser Scanner



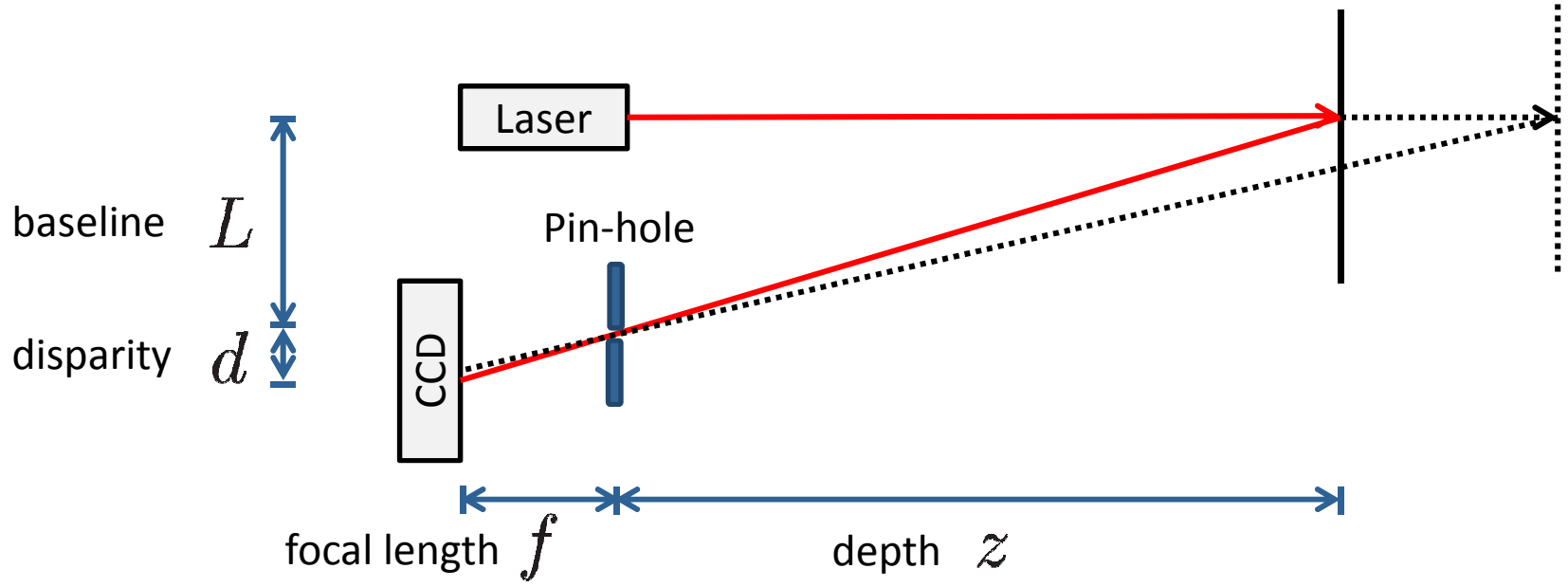
Laser Triangulation

Idea:

- Well-defined light pattern (e.g., point or line) projected on scene
- Observed by a line/matrix camera or a position-sensitive device (PSD)
- Simple triangulation to compute distance

Laser Triangulation

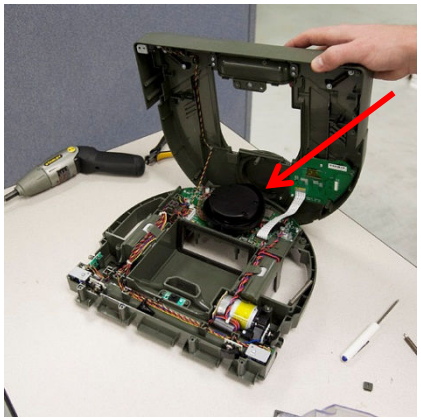
- Function principle

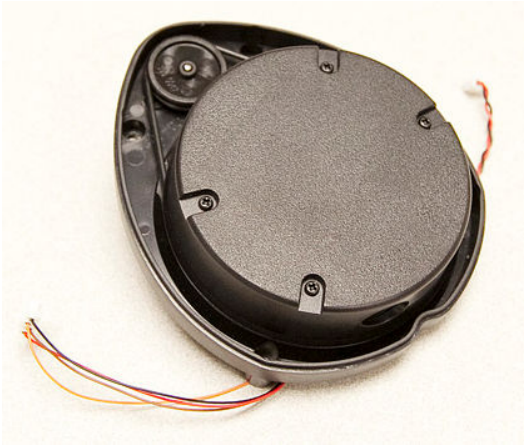


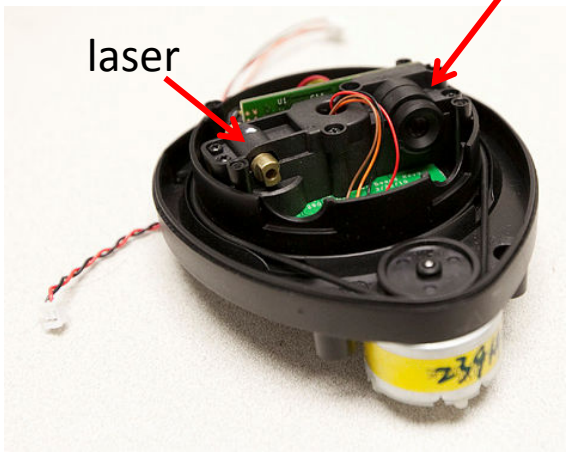
- Depth triangulation $z = f\frac{L}{d}$
  (note: same for stereo disparities)

Example: Neato XV-11

- K. Konolige, “A low-cost laser distance sensor”, ICRA 2008
- Specs: 360deg, 10Hz, 30 USD

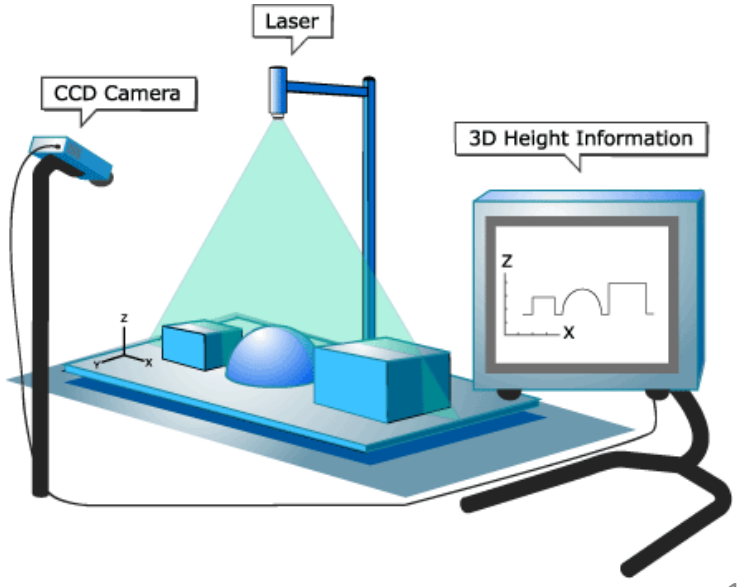


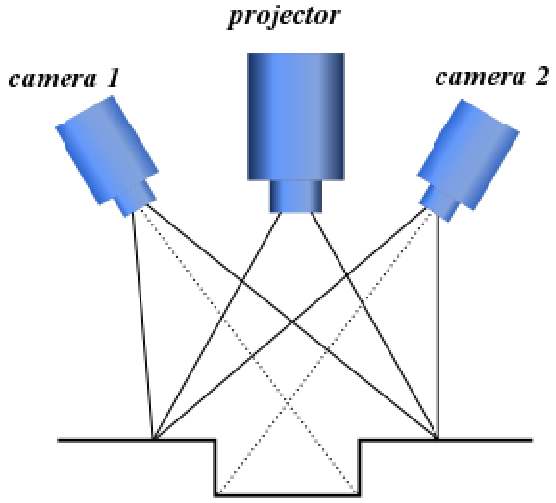




Laser Triangulation

- Stripe laser + 2D camera
- Often used on conveyer belts (volume sensing)
- Large baseline gives better depth resolution but more occlusions → use two cameras

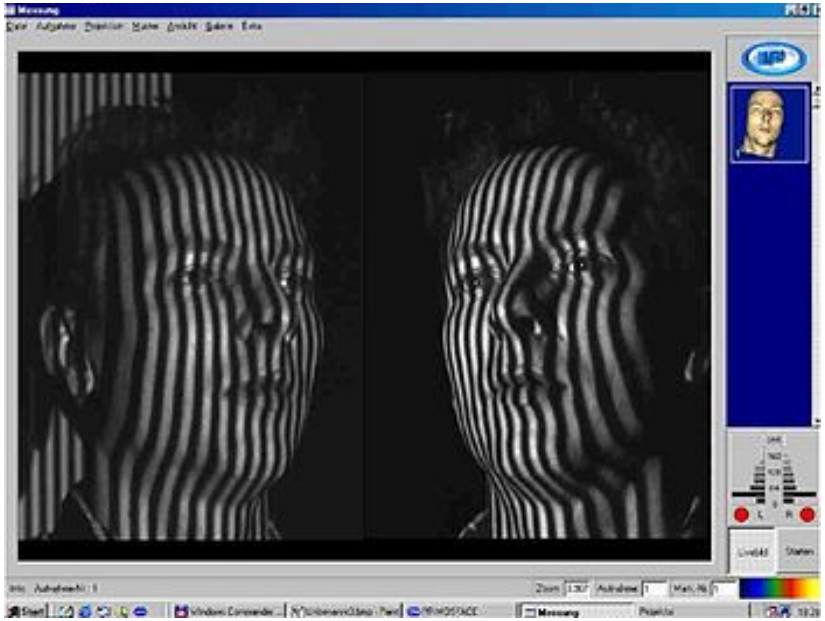




Dr. Thomas Whelan, Dyson Robotics Lab, Imperial College London & Dr. Jürgen Sturm, Computer Vision Group, TUM

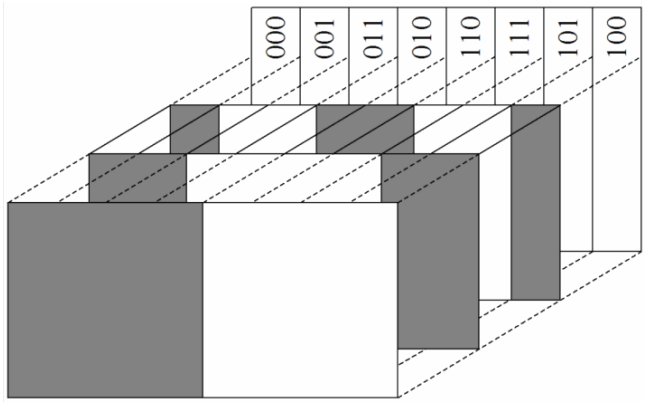
Structured Light

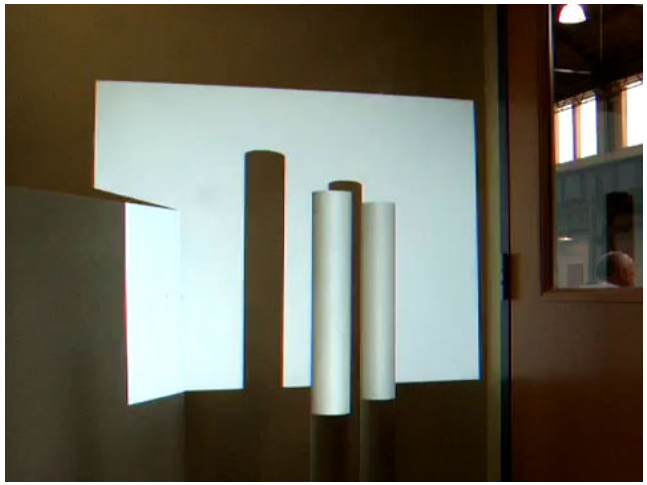
- Multiple stripes / 2D pattern
- Data association more difficult



Structured Light

- Multiple stripes / 2D pattern
- Data association more difficult
- Coding schemes
  - Temporal: Coded light

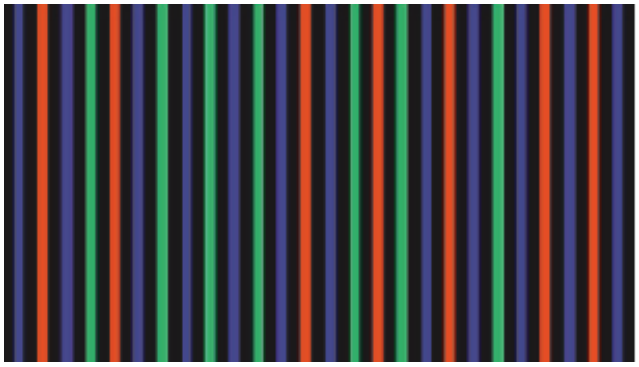




Dr. Thomas Whelan, Dyson Robotics Lab, Imperial College London & Dr. Jürgen Sturm, Computer Vision Group, TUM

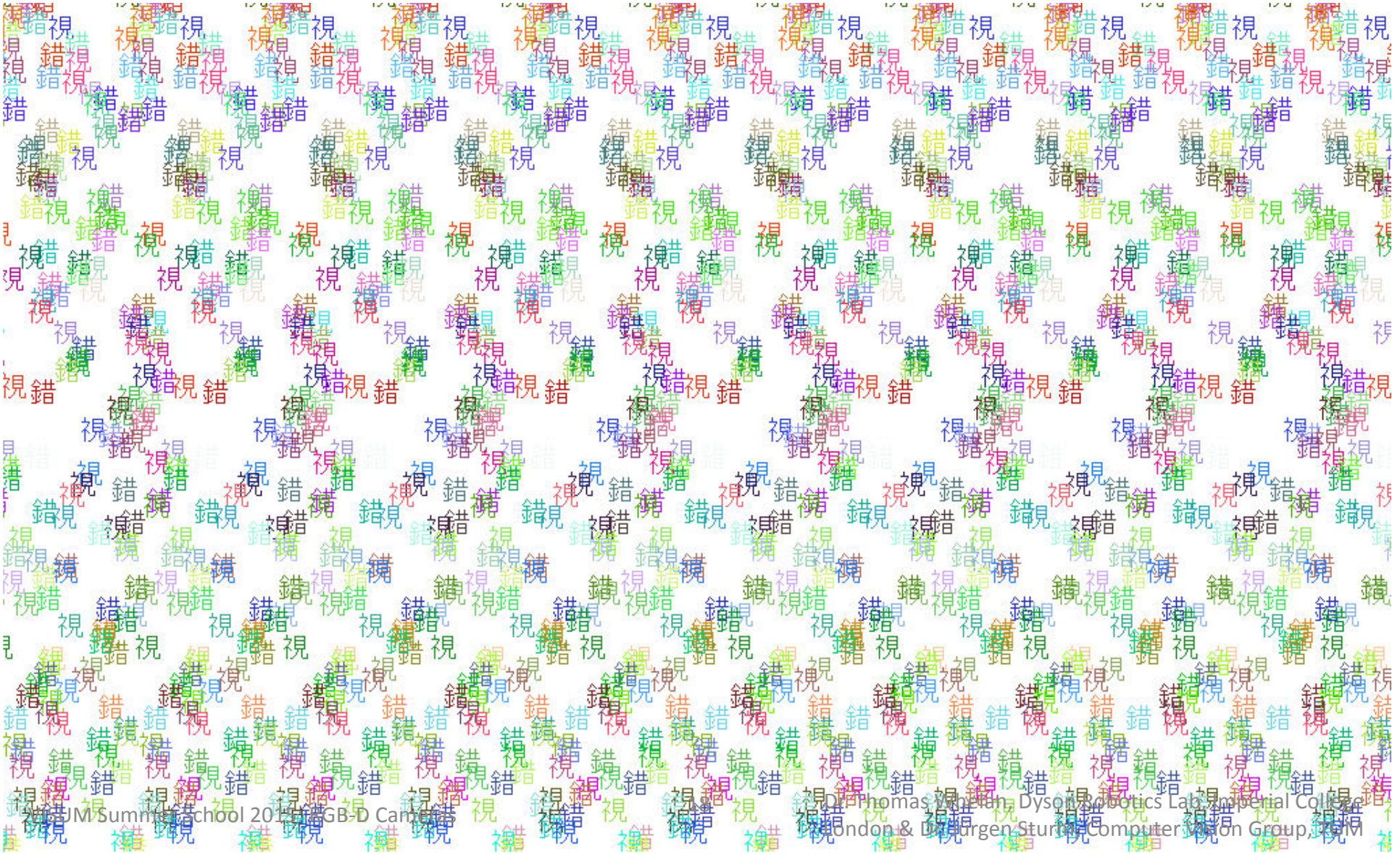
Structured Light

- Multiple stripes / 2D pattern
- Data association more difficult
- Coding schemes
  - Temporal: Coded light
  - Wavelength: Color
  - Spatial: Pattern (e.g., diffraction patterns)



Dr. Thomas Whelan, Dyson Robotics Lab, Imperial College London & Dr. Jürgen Sturm, Computer Vision Group, TUM

Human Stereo Vision



Dr. Thomas Whelan, Dyson Robotics Lab, Imperial College London & Dr. Jürgen Sturm, Computer Vision Group, TUM

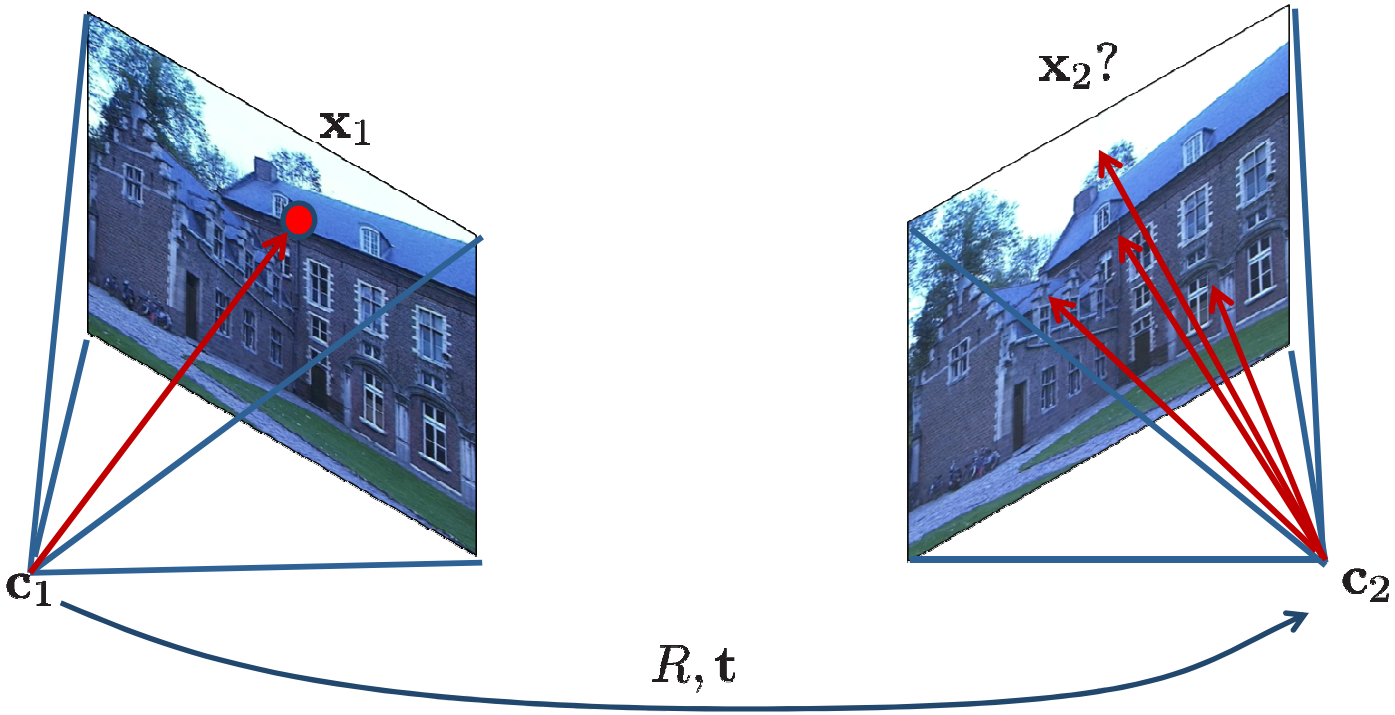
Stereo Cameras

- PointGrey Bumblebee

- Videre STOC camera

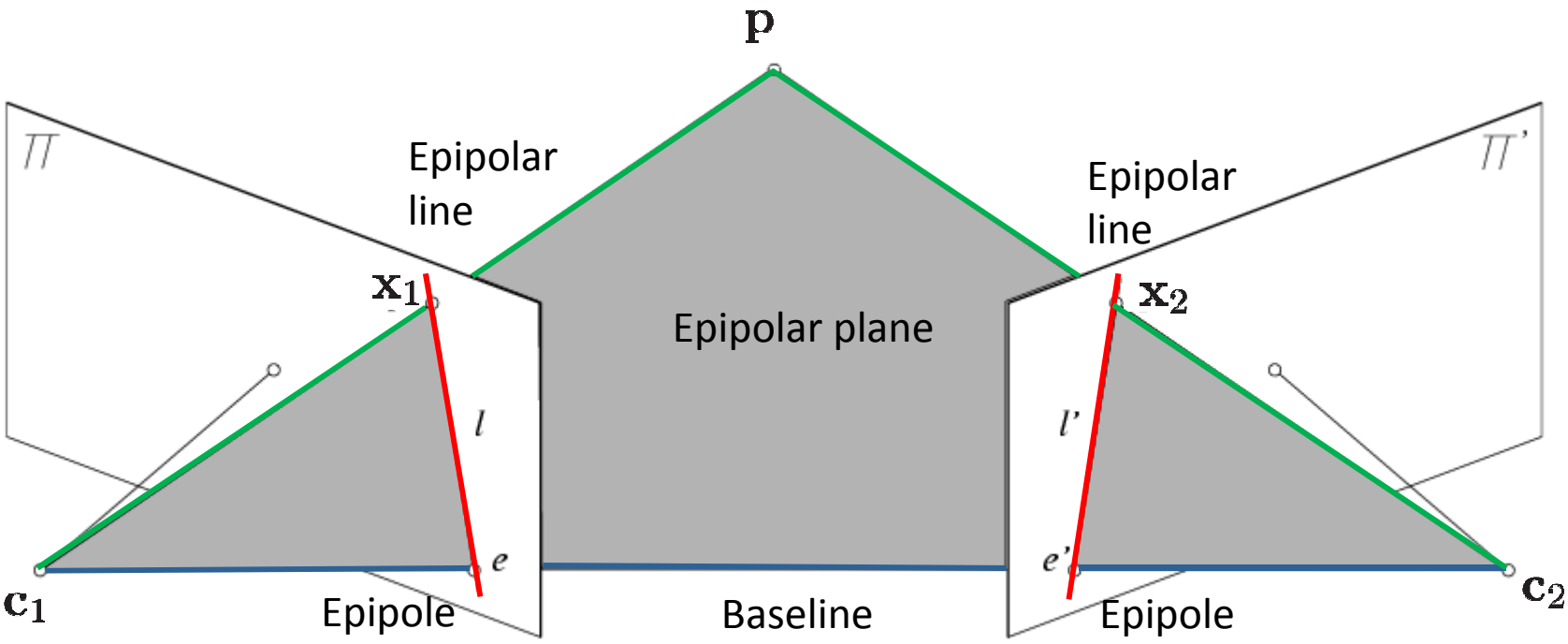
Stereo Correspondence Constraints

- Given a point in the left image, where can the corresponding point be in the right image?



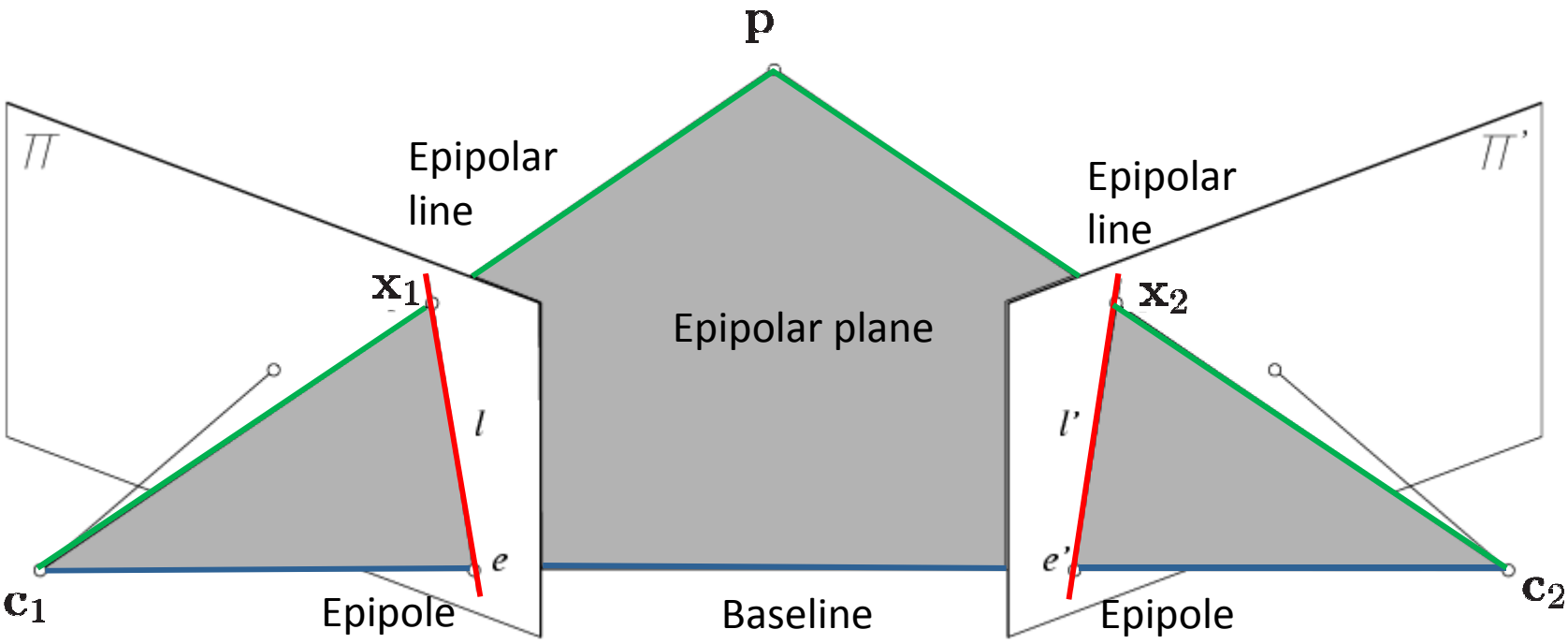
Epipolar Geometry

- A point in one image "generates" a line in another image (called the **epipolar line**)
- Epipolar constraint $\hat{\mathbf{x}}_2^\top E \hat{\mathbf{x}}_1 = 0$



Epipolar Plane

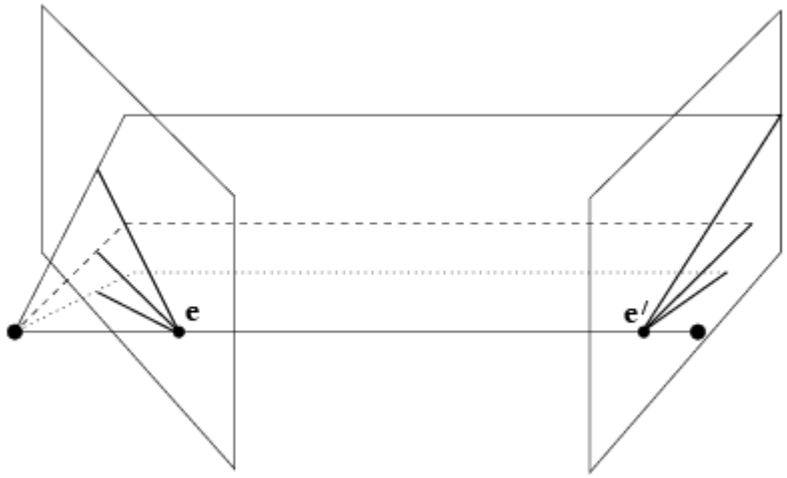
- All epipolar lines intersect at the epipoles
- An epipolar plane intersects the left and right image planes in epipolar lines

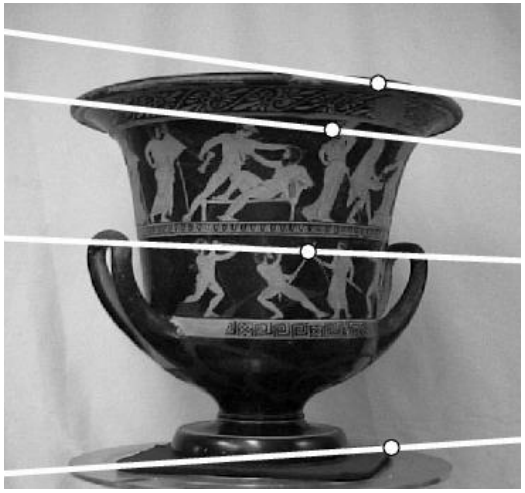


Epipolar Constraint

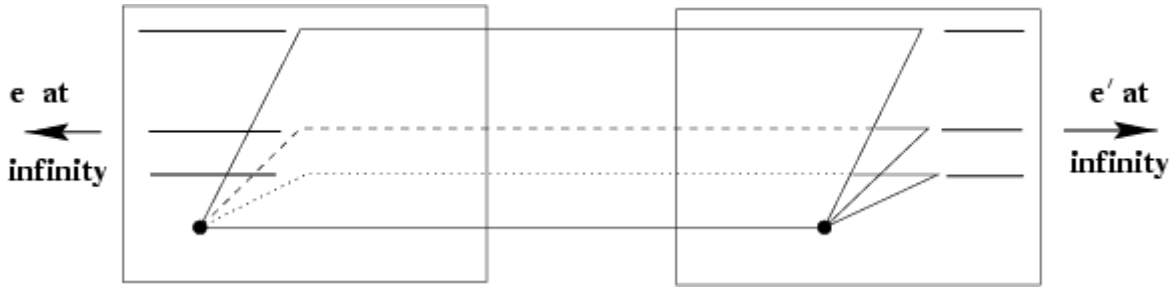
- This is useful because it reduces the correspondence problem to a 1D search along an epipolar line

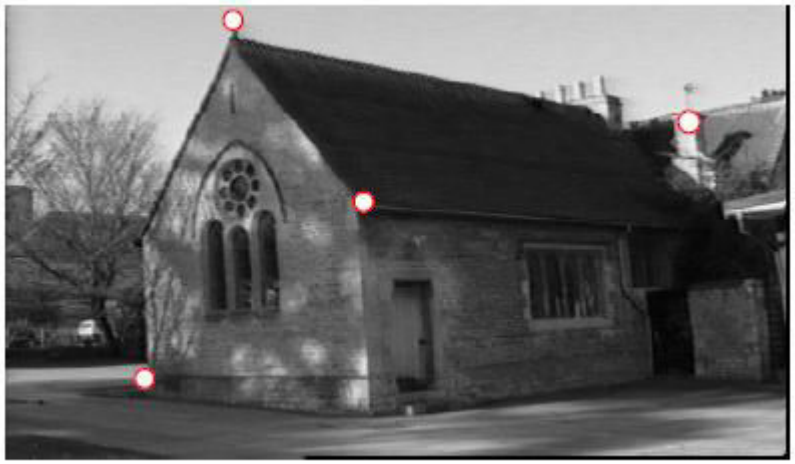
Example: Converging Cameras

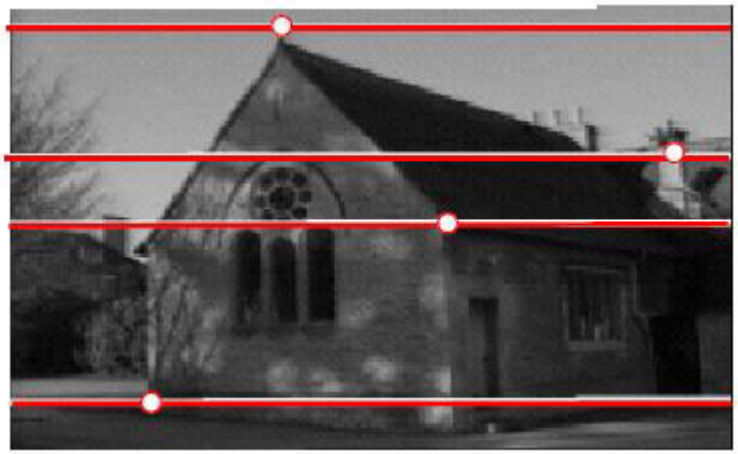




Example: Parallel Cameras





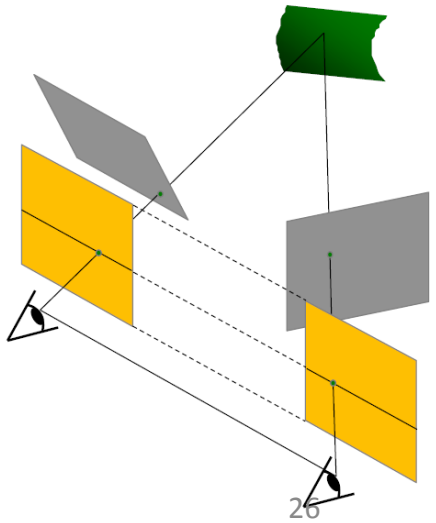


Rectification

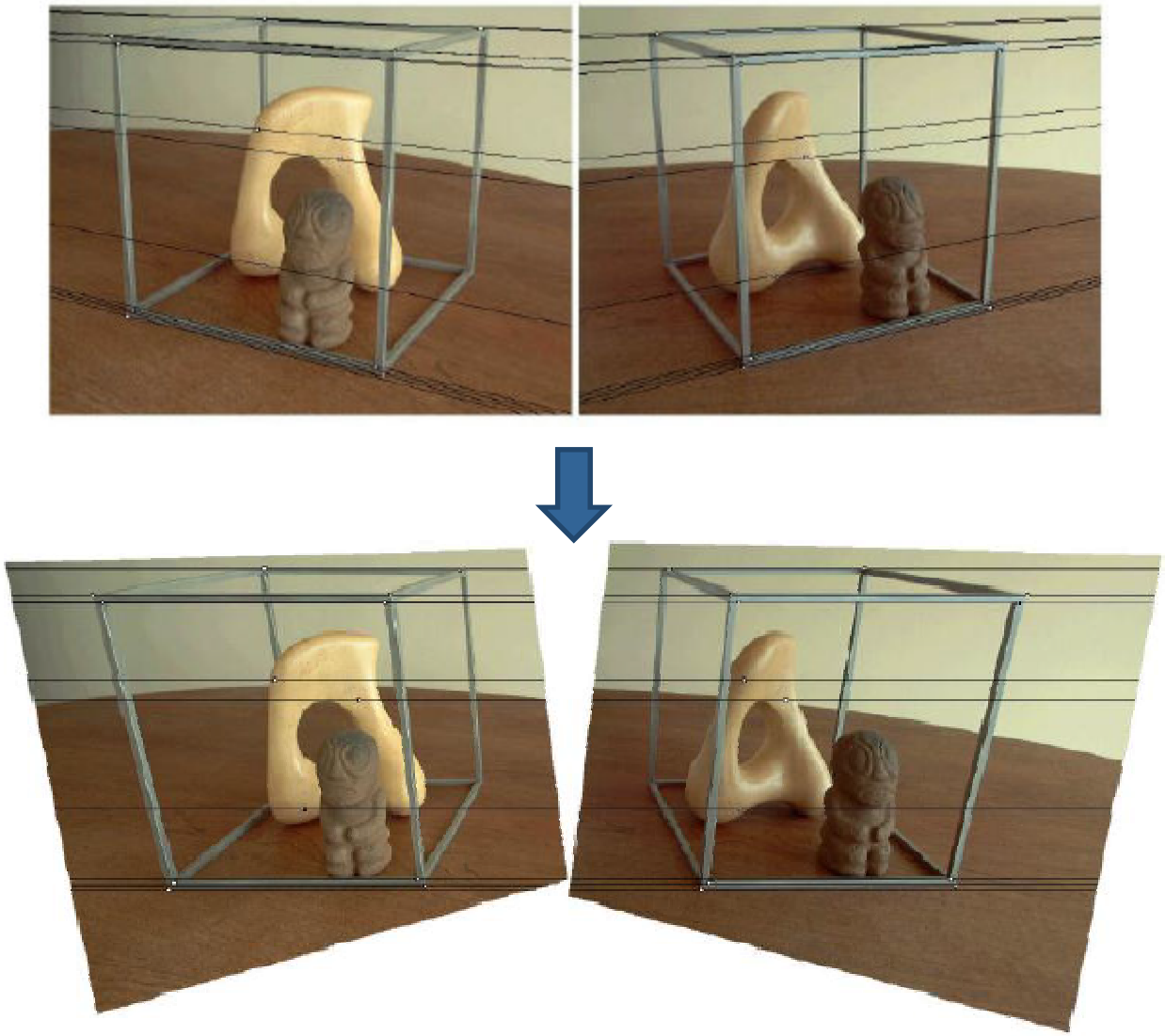
- In practice, it is convenient if the image scanlines (rows) are the epipolar lines

→ Reproject image planes onto a common plane parallel to the baseline (two 3x3 homographies)

- Afterwards pixel motion is horizontal

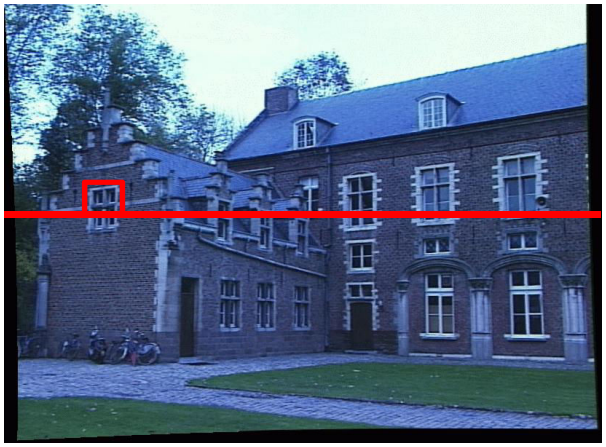


Example: Rectification

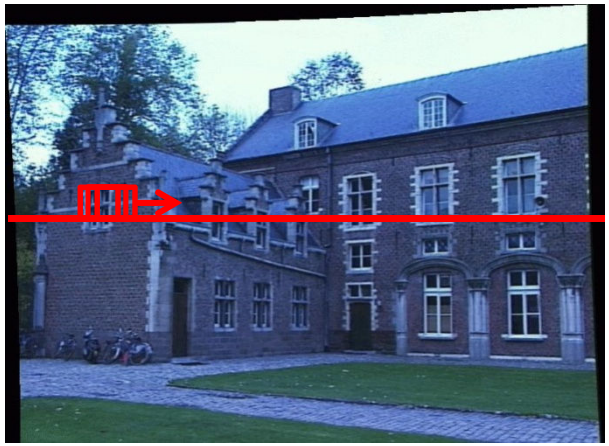


Basic Stereo Algorithm

- For each pixel in the left image
  - Compare with every pixel on the same epipolar line in the right image
  - Pick pixel with minimum matching cost (noisy)
  - Better: match small blocks/patches (SSD, SAD, NCC)



left image



right image

Block Matching Algorithm

Input: Two images and camera calibrations

Output: Disparity (or depth) image

Algorithm:

1. Geometry correction (undistortion and rectification)
2. Matching cost computation along search window
3. Extrema extraction (at sub-pixel accuracy)
4. Post-filtering (clean up noise)

Example

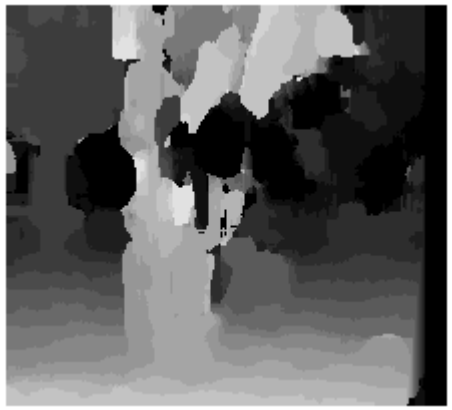
- Input

- Output

What is the Influence of the Block Size?

- Common choices are 5x5 .. 11x11
- Smaller neighborhood: more details
- Larger neighborhood: less noise
- Suppress pixels with low confidence (e.g., check ratio best match vs. 2nd best match)

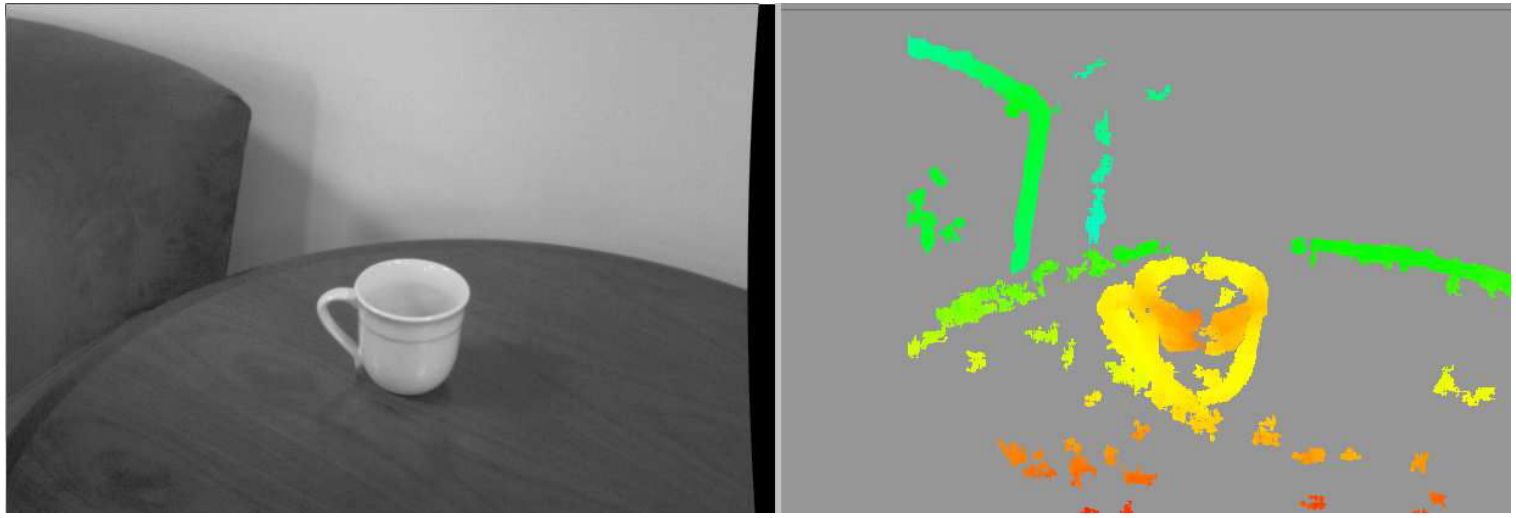
3x3



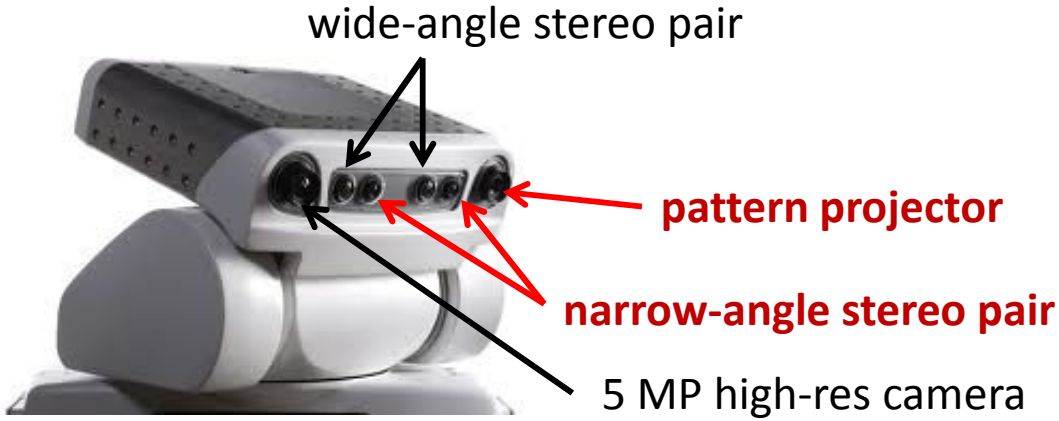
20x20

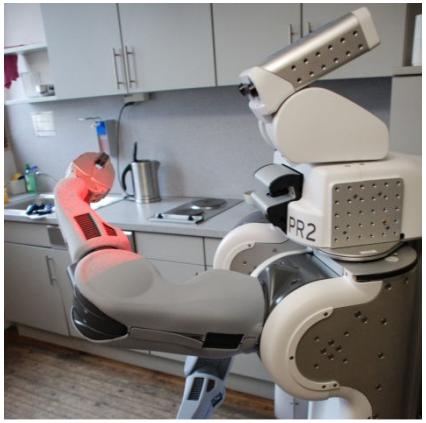
Problems with Stereo

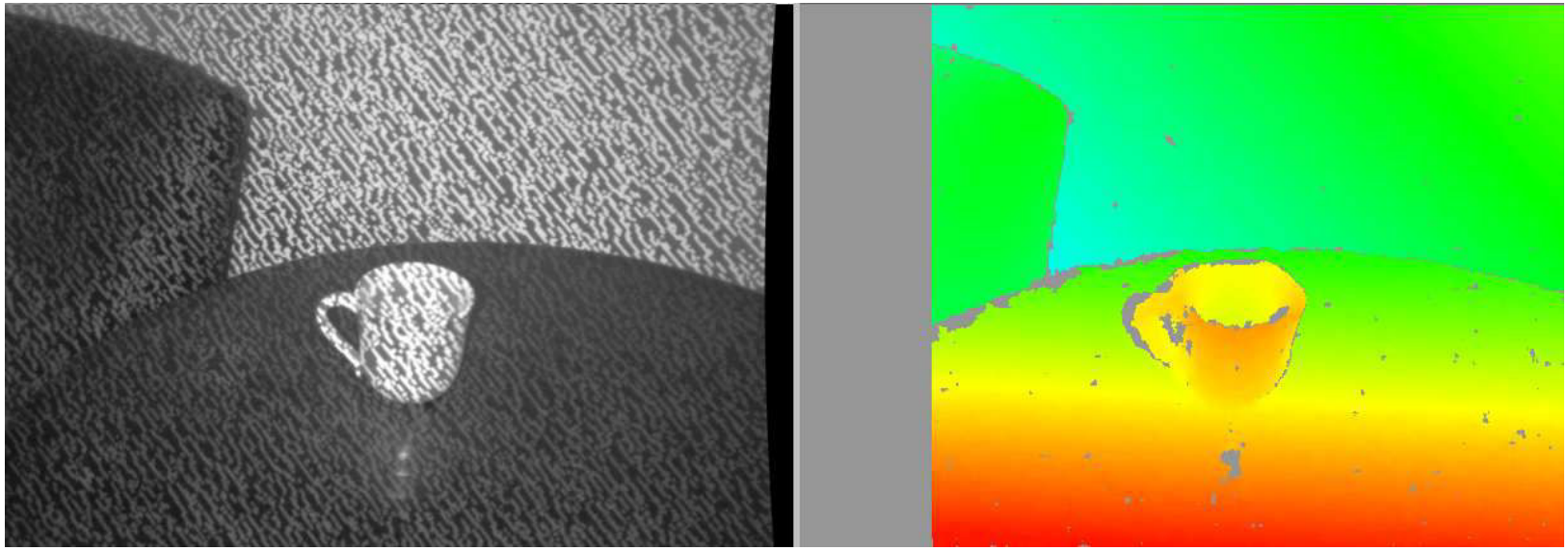
- Block matching typically fails in regions with low texture
  - Global optimization/regularization
  - Additional texture projection



Example: PR2 Robot with Projected Texture Stereo







Dr. Thomas Whelan, Dyson Robotics Lab, Imperial College London & Dr. Jürgen Sturm, Computer Vision Group, TUM

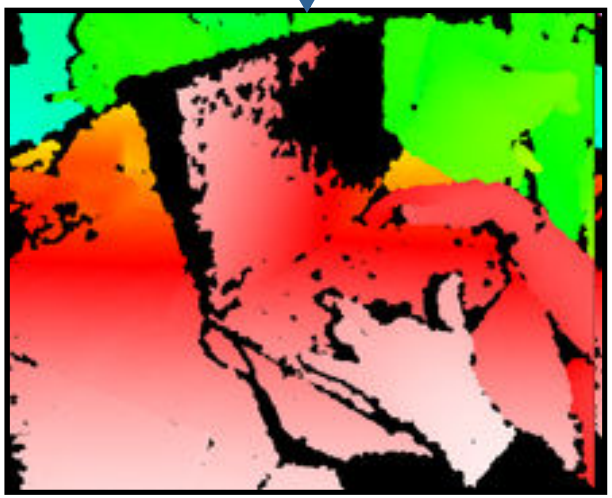
Sensor Principle of Kinect

Infrared pattern
(known)

Standard block matcher (9x9)

Disparity image

Infrared image
(with distorted pattern)



Depth image
(color encodes distance from camera)

Sensor Principle of Kinect

- Pattern is memorized at a known depth
- For each pixel in the IR image
  - Extract 9x9 template from memorized pattern
  - Correlate with current IR image over 64 pixels and search for the maximum
  - Interpolate maximum to obtain sub-pixel accuracy (1/8 pixel)
  - Calculate depth by triangulation

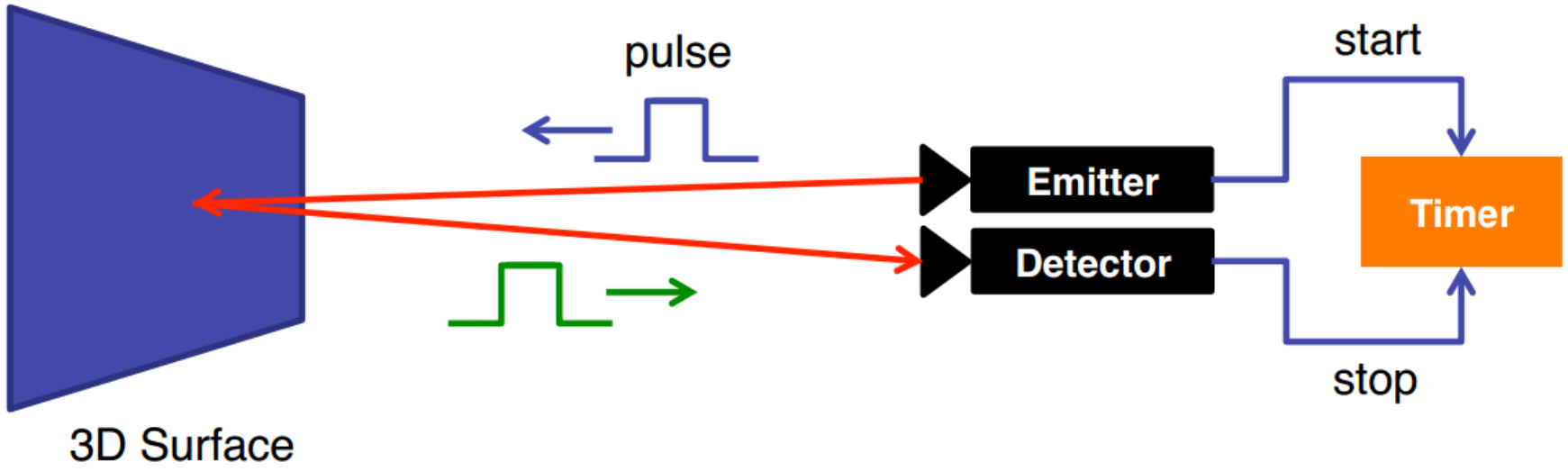
Technical Specs

- Infrared camera has 640x480 @ 30 Hz
  - Depth correlation runs on FPGA
  - 11-bit depth image
  - 0.8m – 5m range
  - Depth sensing does not work in direct sunlight (why?)
- RGB camera has 640x480 @ 30 Hz
- Cost: 90 Euro

Live Demo

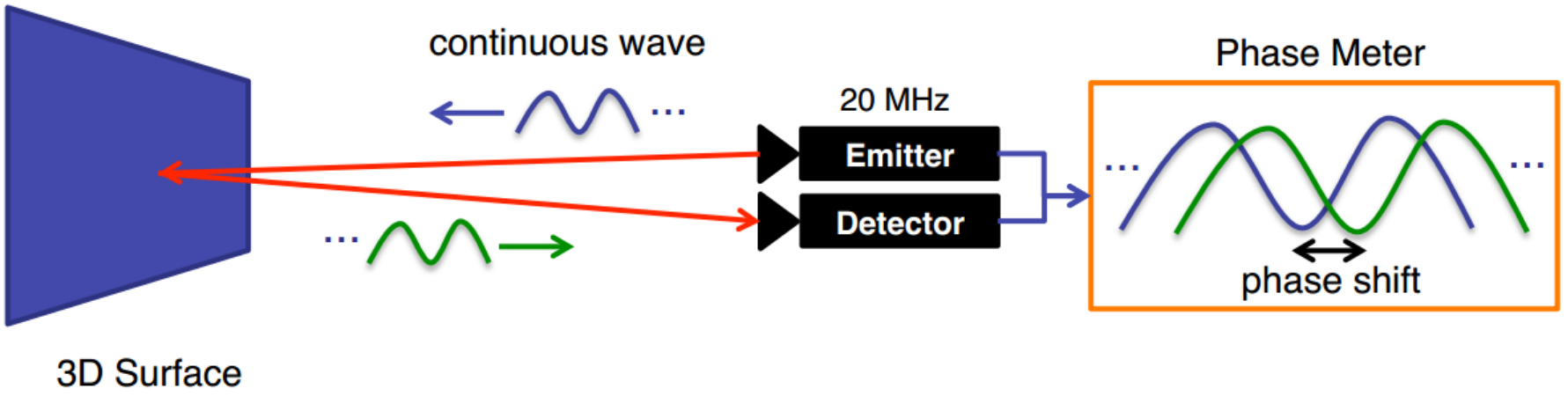
Time-of-Flight Cameras

- Direct time-of-flight measurement
  - Emit short light pulse (flash)
  - Every pixel counts time until signal is detected



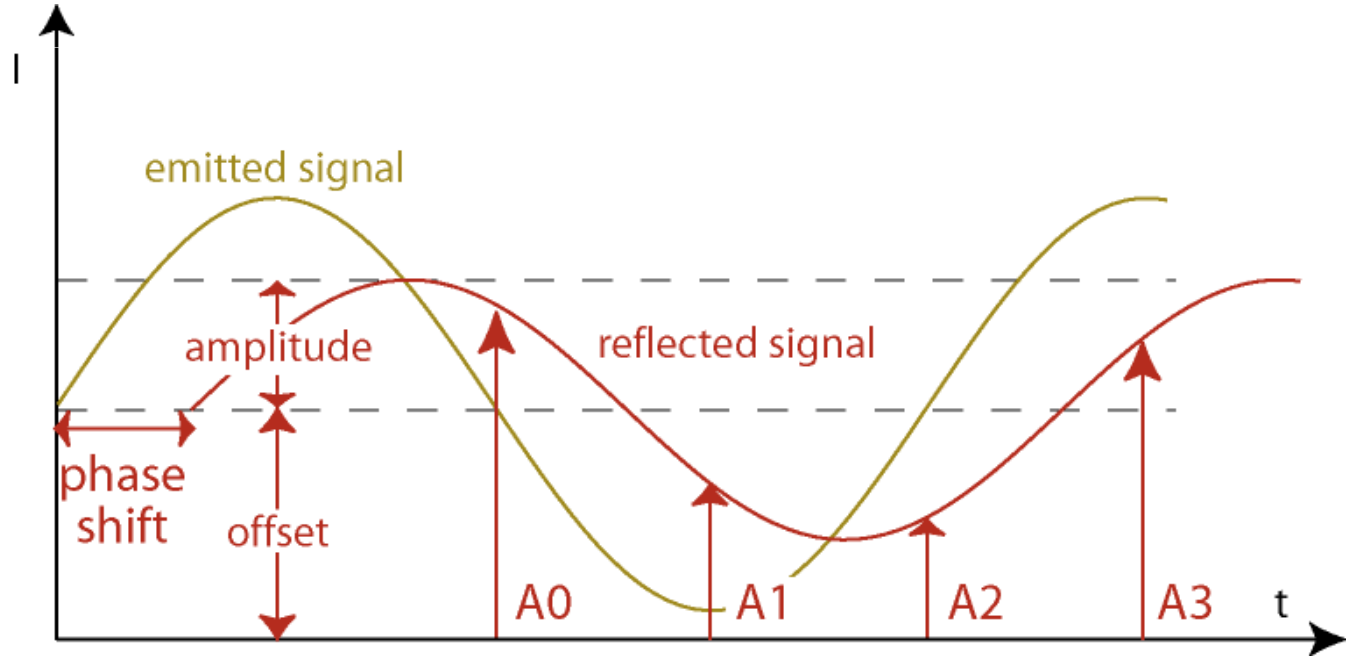
Time-of-Flight Cameras

- Indirect measurement (phase shift)
  - Emit modulated light (e.g., at 30 MHz → 10m wave length)
  - Every pixel measures the phase shift



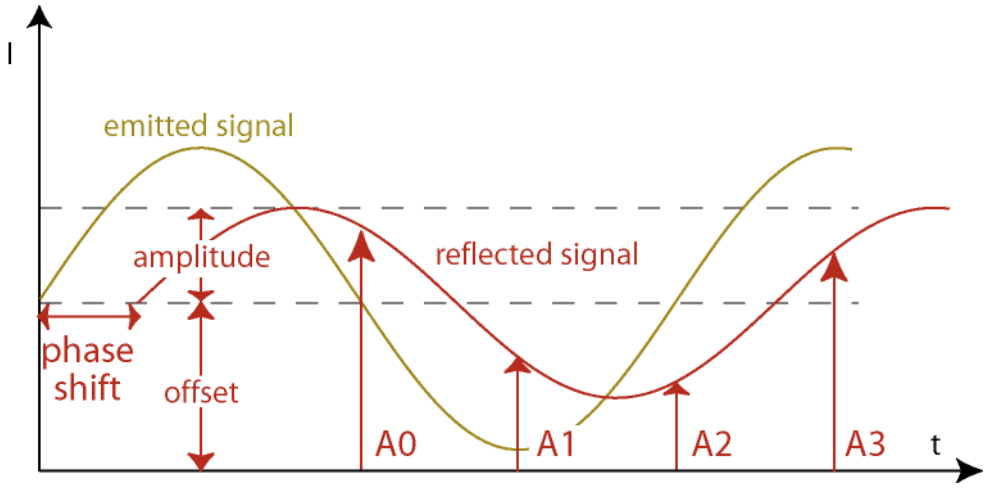
Decoding the Phase

- Take four intensity measurements at 90° angle
- Integrate over several waves to reduce noise
- Decode amplitude, offset, phase shift



Decoding the Phase

- Amplitude (=quality) $A = \frac{1}{2}\sqrt{(A_3 - A_1)^2 + (A_2 - A_0)^2}$
- Offset (=intensity) $B = A_0 + A_1 + A_2 + A_3$
- Phase shift (=distance) $\phi = \arctan \frac{A_3 - A_1}{A_2 - A_0}$



Commercial Time-Of-Flight Sensors

- 2008: Mesa SwissRanger, 176x144, $9,000
- 2009: PMDTec, 204x204, $12,000
- 2012: Intel Creative Camera, 320x240, $150
- 2013: Xbox One Kinect, 512x424, 13-bit depth

Wrap-Up: RGB-D Sensor Principles

- Different sensor principles
- Dense depth images at video frame rates
- Very cheap (since recently)

- Now that we have the data, what can we do with it?

Outline: First Session

- Hands-on with RGB-D cameras ✓
- RGB-D SLAM
- RGB-D Benchmark

Outline: Second Session

- Pose graph optimization
- Dense tracking and 3D reconstruction
  - Motion estimation
  - Signed distance functions
  - Dense visual odometry
  - Large scale reconstruction and SLAM
  - Applications

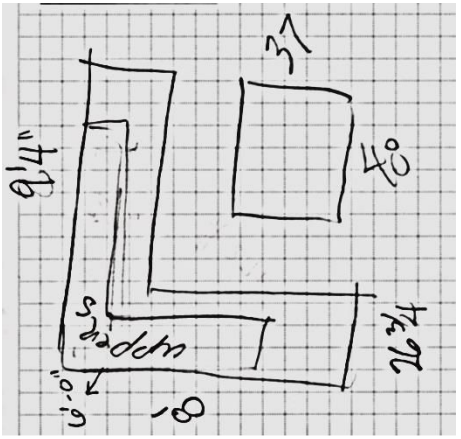
Outline: Afternoon Session

- Get bootable USB stick + RGB-D camera
- Play around
  - Inspect color, infrared, depth images
  - Inspect 3D point clouds
  - Write your own program (given example code)
    - Simple operations on RGB-D data
    - A full SLAM system

RGB-D SLAM

[Engelhard et al., 2011; Endres et al. 2012]

- Goal:
  Acquire 3D models of (indoor) scenes



- Applications:
  - Architecture
  - Gaming
  - Archaeology

Approach

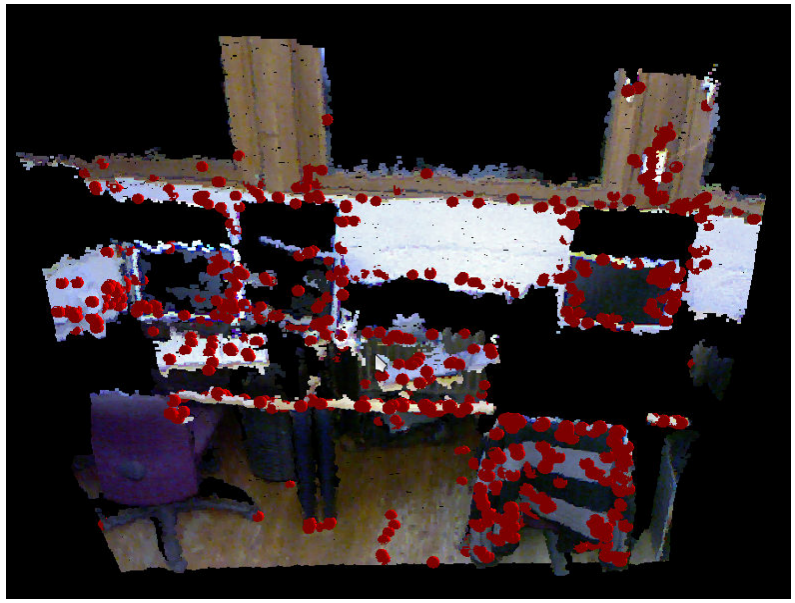
[Engelhard et al., 2011; Endres et al., 2012]

- Step 1: Extract 2D features (SIFT)

Approach

[Engelhard et al., 2011; Endres et al., 2012]

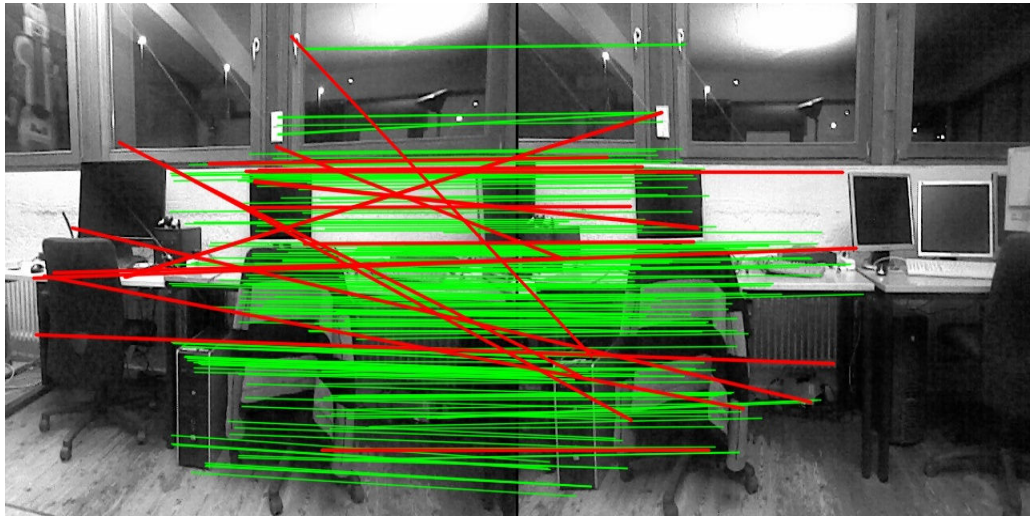
- Step 1: Extract 2D features (SIFT)
- Step 2: Associate features with 3D points



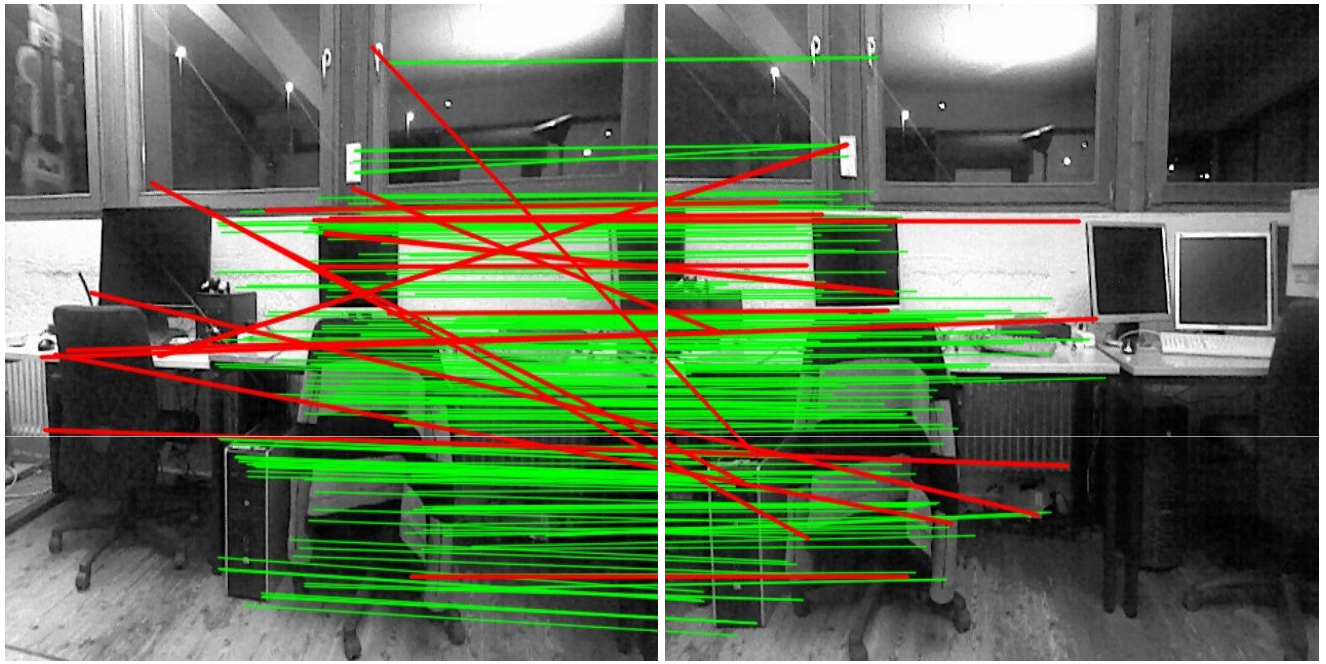
Approach

[Engelhard et al., 2011; Endres et al., 2012]

- Step 1: Extract 2D features (SIFT)
- Step 2: Associate features with 3D points
- Step 3: Find corresponding points (RANSAC)
- Step 4: Estimate camera motion (SVD)



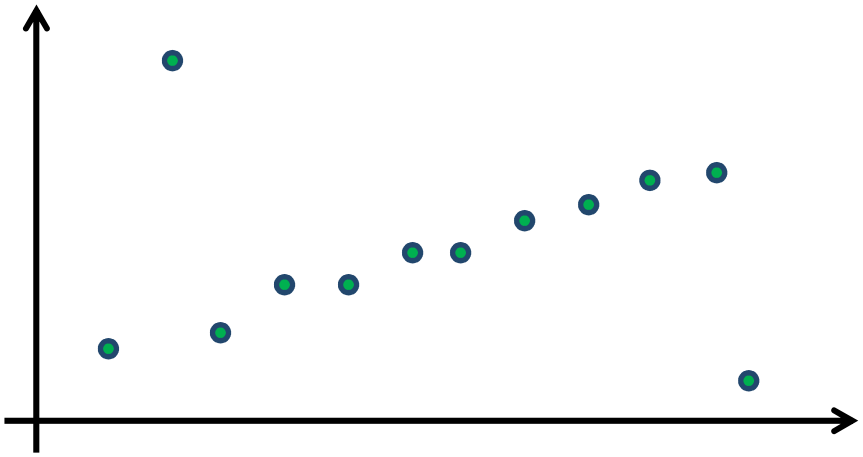
How To Deal With Outliers?



Problem: No matter how good the feature descriptor/matcher is, there is always a chance for bad point correspondences (=outliers)

Robust Estimation

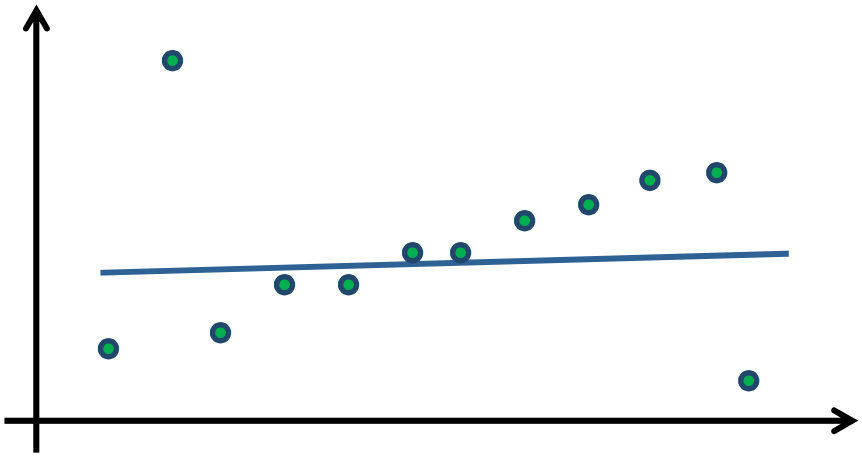
Example: Fit a line to 2D data containing outliers



- Input data is a mixture of
  - Inliers (perturbed by Gaussian noise)
  - Outliers (unknown distribution)
- Let's fit a line using least squares…

Robust Estimation

Example: Fit a line to 2D data containing outliers



- Input data is a mixture of
  - Inliers (perturbed by Gaussian noise)
  - Outliers (unknown distribution)
- Least squares fit gives poor results!

RANdom SAmple Consensus (RANSAC)

[Fischler and Bolles, 1981]

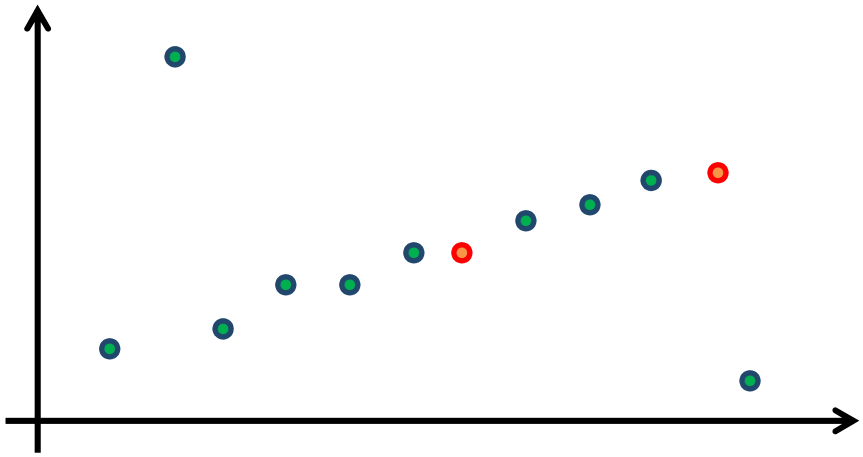
Goal: Robustly fit a model to a data set S which contains outliers

Algorithm:

1. Randomly select a (minimal) subset
2. Instantiate the model from it
3. Using this model, classify the all data points as inliers or outliers
4. Repeat 1-3 for N iterations
5. Select the largest inlier set, and re-estimate the model from all points in this set

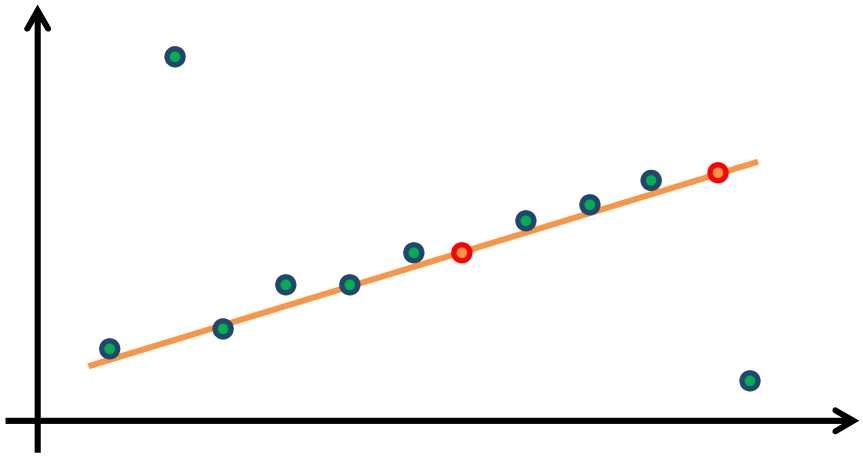
Example

- Step 1: Sample a random subset



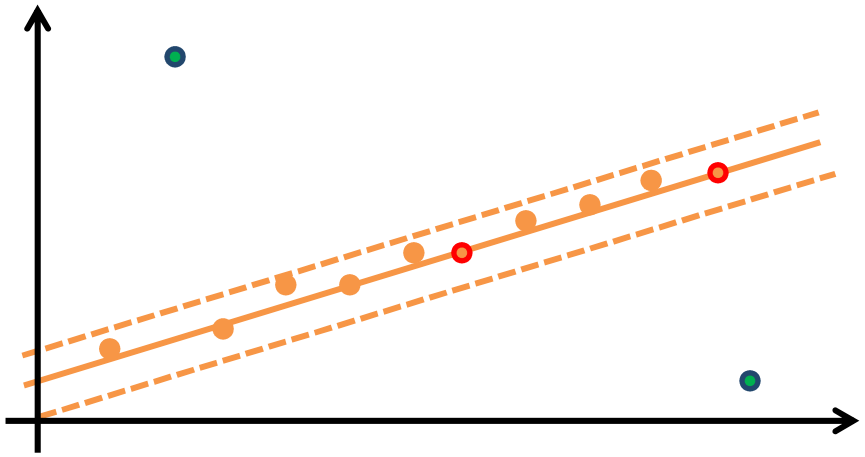
Example

- Step 2: Fit a model to this subset



Example

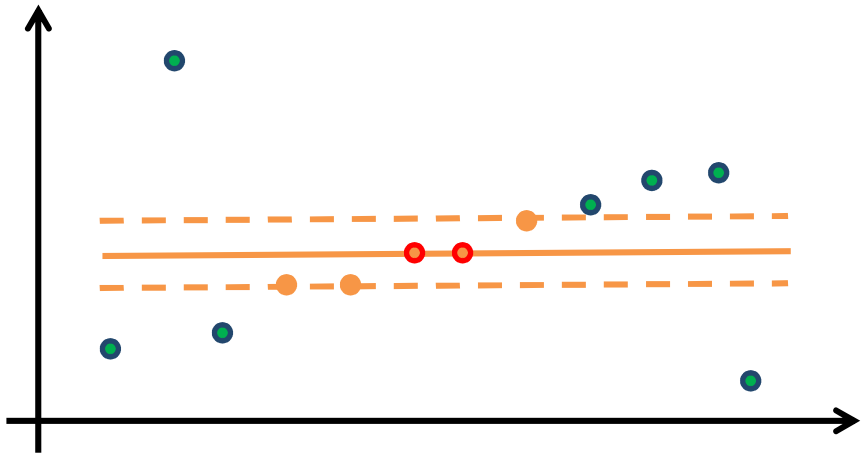
- Step 3: Classify points as inliers and outliers (e.g., using a threshold distance)



→ 10 inliers, 2 outliers

Example

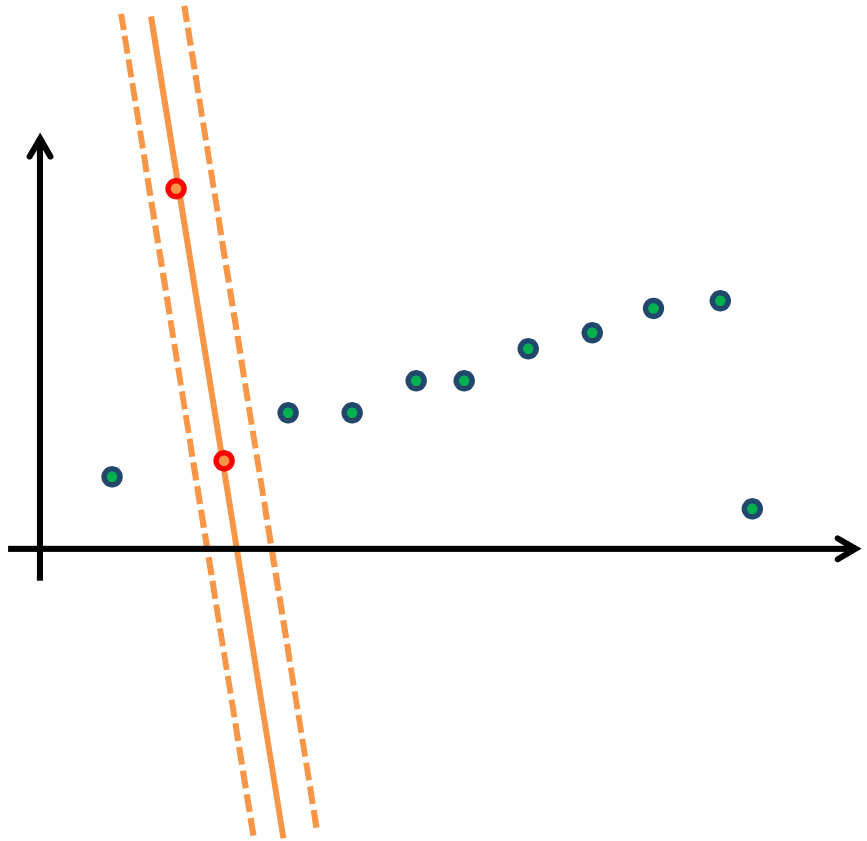
- Step 4: Repeat steps 1-3 for N iterations



Iteration 2:
→ 5 inliers, 7 outliers

Example

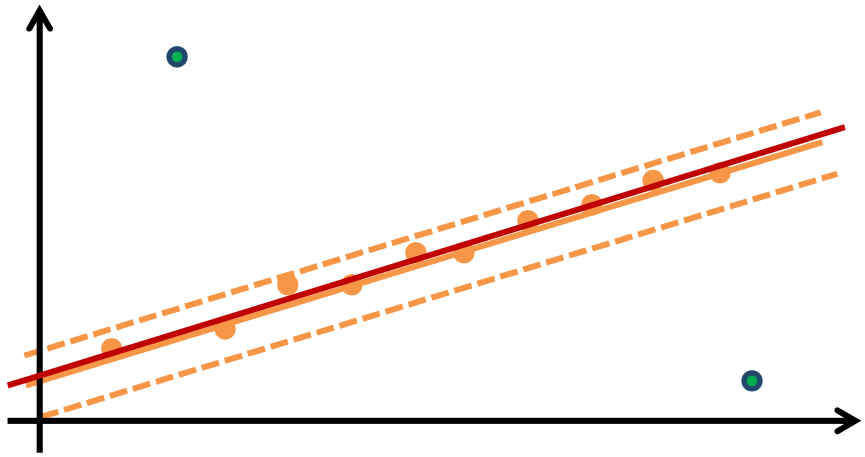
- Step 4: Repeat steps 1-3 for N iterations



Iteration 3:
→ 2 inliers, 10 outliers

Example

- Step 5: Select the best model (most inliers), the re-fit model using all inliers



Best model:
Iteration 1
(10 inliers, 2 outliers)

How Many Iterations Do We Need?

- For a probability of success p, we need

$$N = \frac{\log(1-p)}{\log(1-(1-\epsilon)^s)} \text{ iterations}$$

  for subset size s and outlier ratio ϵ

- E.g., for p=0.99:

	Required points s	Outlier ratio ε						
		10 %	20 %	30 %	40 %	50 %	60 %	70 %
Line	2	3	5	7	11	17	27	49
Plane	3	4	7	11	19	35	70	169
Essential matrix	8	9	26	78	272	1177	7025	70188

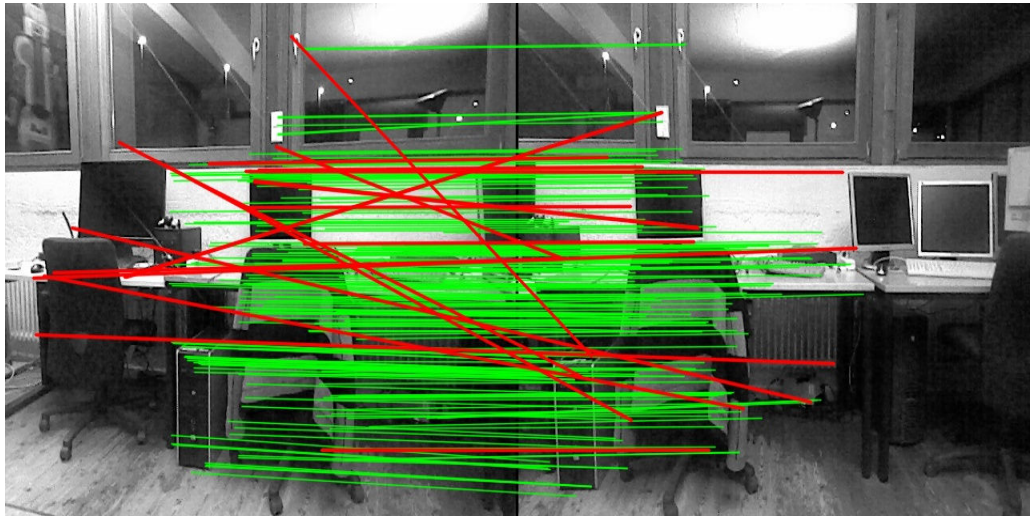
Summary on RANSAC

- Efficient algorithm to estimate a model from noisy and outlier-contaminated data
- RANSAC is used today very widely
- Often used in feature matching / visual motion estimation
- Many improvements/variants (e.g., PROSAC, MLESAC, …)

Approach

[Engelhard et al., 2011; Endres et al., 2012]

- Step 1: Extract 2D features (SIFT)
- Step 2: Associate features with 3D points
- Step 3: Find corresponding points (RANSAC)
- Step 4: Estimate camera motion (SVD)



Iterative Closest Point (ICP)

- **Given:** Two corresponding point sets (clouds)

$$P = \{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$$
$$Q = \{\mathbf{q}_1, \ldots, \mathbf{q}_n\}$$

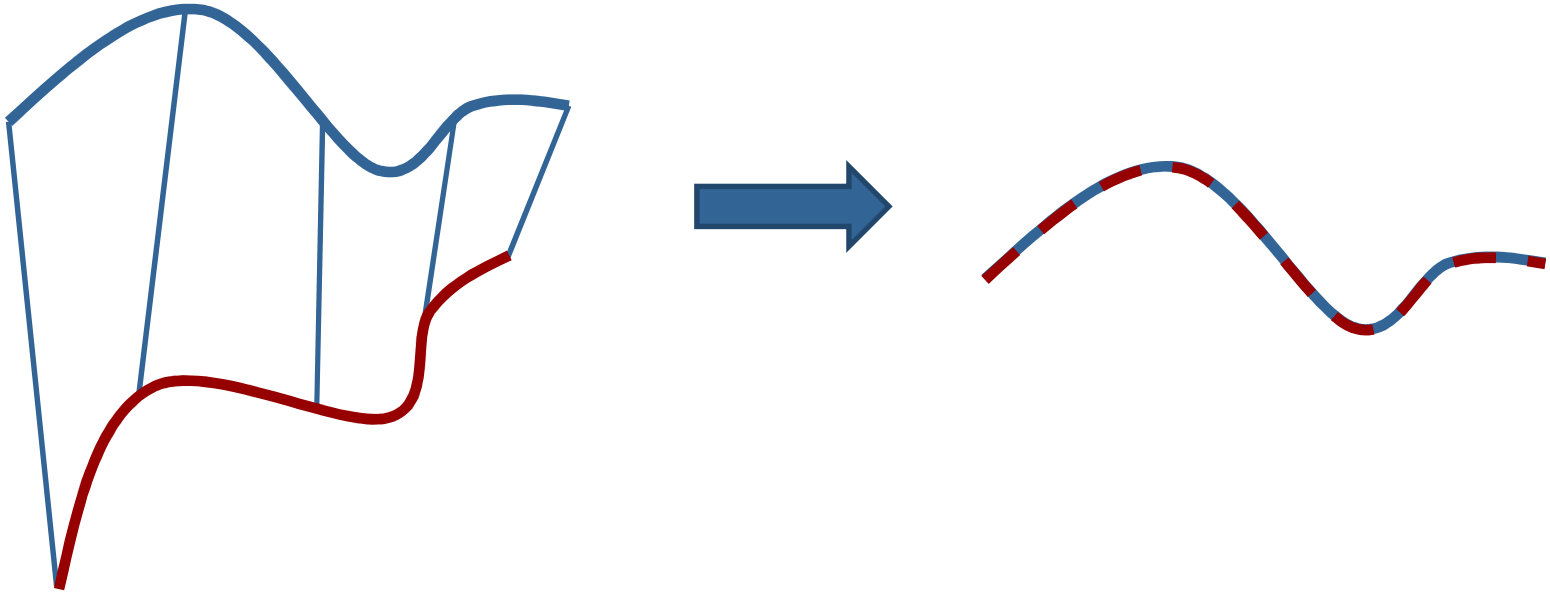
- **Wanted:** Translation $\mathbf{t}$ and rotation R that minimize the sum of the squared error

$$E(R, \mathbf{t}) = \frac{1}{n} \sum_{i=1}^{n} ||\mathbf{p}_i - R\mathbf{q}_i - \mathbf{t}||^2$$

where $\mathbf{p}_i$ and $\mathbf{q}_i$ are corresponding points

Known Correspondences

Note: If the correct correspondences are known, both rotation and translation can be calculated in **closed form**.



Known Correspondences

- **Idea:** The center of mass of both point sets has to match

$$\bar{\mathbf{p}} = \frac{1}{n}\sum_i \mathbf{p}_i \qquad \bar{\mathbf{q}} = \frac{1}{n}\sum_i \mathbf{q}_i$$

- Subtract the corresponding center of mass from every point
- Afterwards, the point sets are zero-centered, i.e., we only need to recover the rotation…

Known Correspondences

- Decompose the matrix

$$W = \sum_i (\mathbf{p}_i - \bar{\mathbf{p}})(\mathbf{q}_i - \bar{\mathbf{q}})^\top = USV^\top$$

using singular value decomposition (SVD)

- **Theorem**

If $\operatorname{rank} W = 3$, the optimal solution of $E(R, \mathbf{t})$ is unique and given by

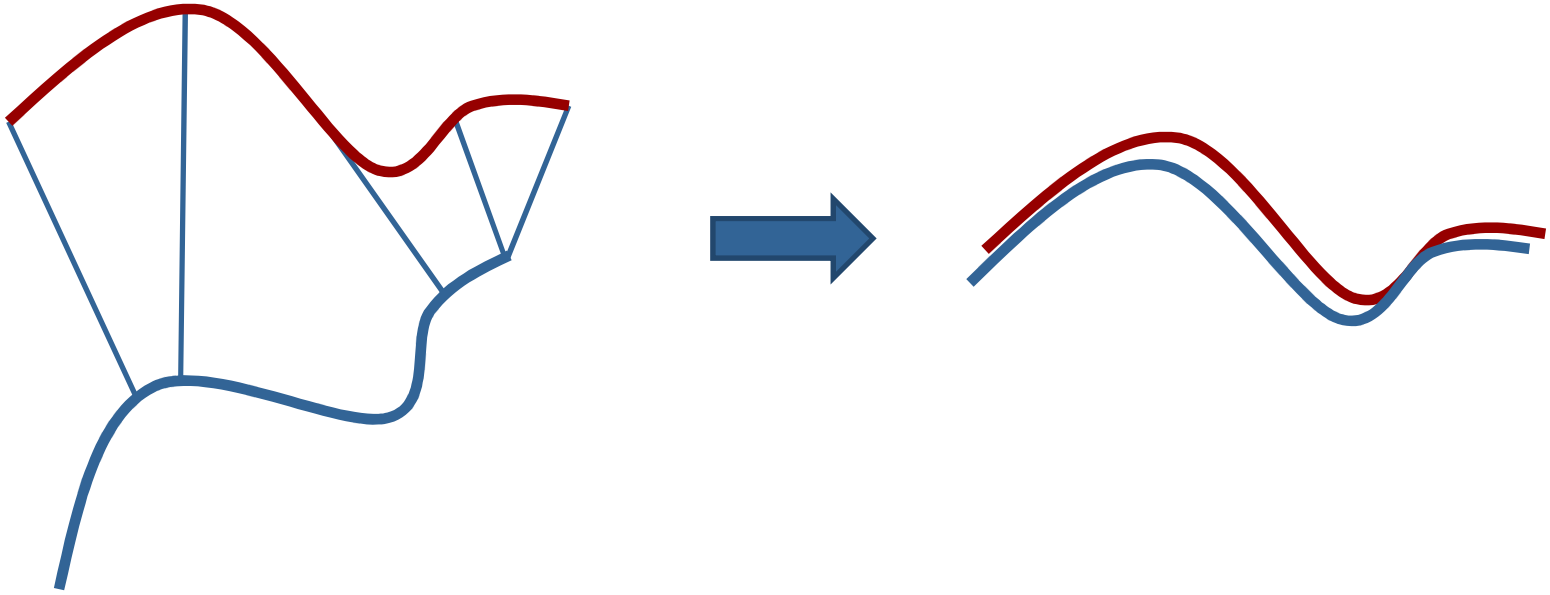
$$R = UV^\top$$

$$\mathbf{t} = \bar{\mathbf{p}} - R\bar{\mathbf{q}}$$

(for proof, see http://hss.ulb.uni-bonn.de/2006/0912/0912.pdf, p.34/35)

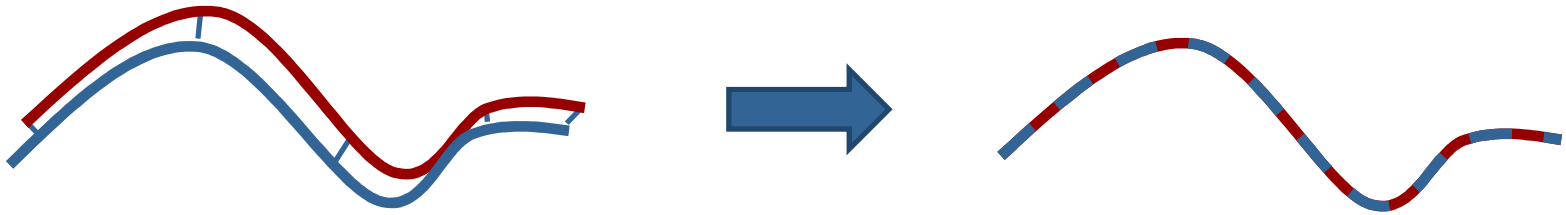
Unknown Correspondences

- If the correct correspondences are not known, it is generally impossible to determine the optimal transformation in one step



ICP Algorithm
[Besl & McKay, 92]

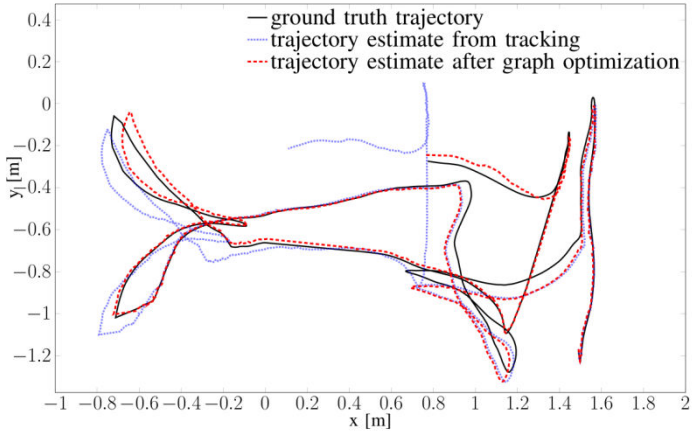
- **Algorithm:** Iterate until convergence
  - Find correspondences
  - Solve for R,t
- Converges if starting position is “close enough”



Approach

[Engelhard et al., 2011; Endres et al., 2012]

- Step 1: Extract 2D features (SIFT)
- Step 2: Associate features with 3D points
- Step 3: Find corresponding points (RANSAC)
- Step 4: Estimate camera motion (SVD)
- Step 5: Create a pose graph and optimize (g2o)



Wrap-Up: RGB-D SLAM

[Engelhard et al., 2011; Endres et al., 2012]

- Feature-based RGB-D SLAM
- Quick and easy acquisition of dense 3D maps
- Evaluation on public benchmark
- Software is available online (GPL)
  http://ros.org/wiki/rgbdslam

Evaluation and Benchmarking

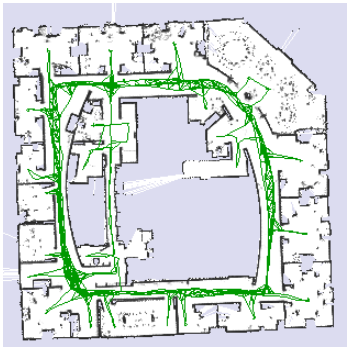
[Sturm et al., IROS 2012]

- How can we evaluate such methods?
- What are good evaluation criteria?

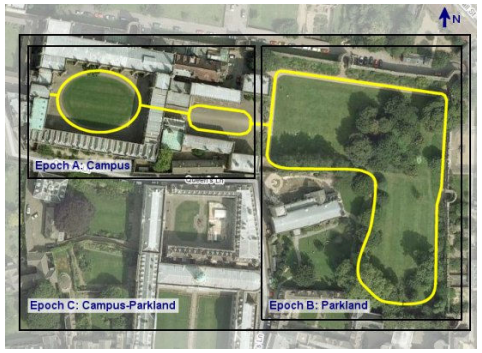
Existing Benchmarks

[Sturm et al., IROS 2012]

- Intel dataset: laser + odometry [Haehnel et al., 2004]
- New College dataset: stereo + omni-directional vision + laser + IMU [Smith et al., IJRR'2009]
- KITTI Vision benchmark: stereo [Geiger et al., CVPR'12]
- TUM Freiburg dataset for RGB-D evaluation



Intel



New College

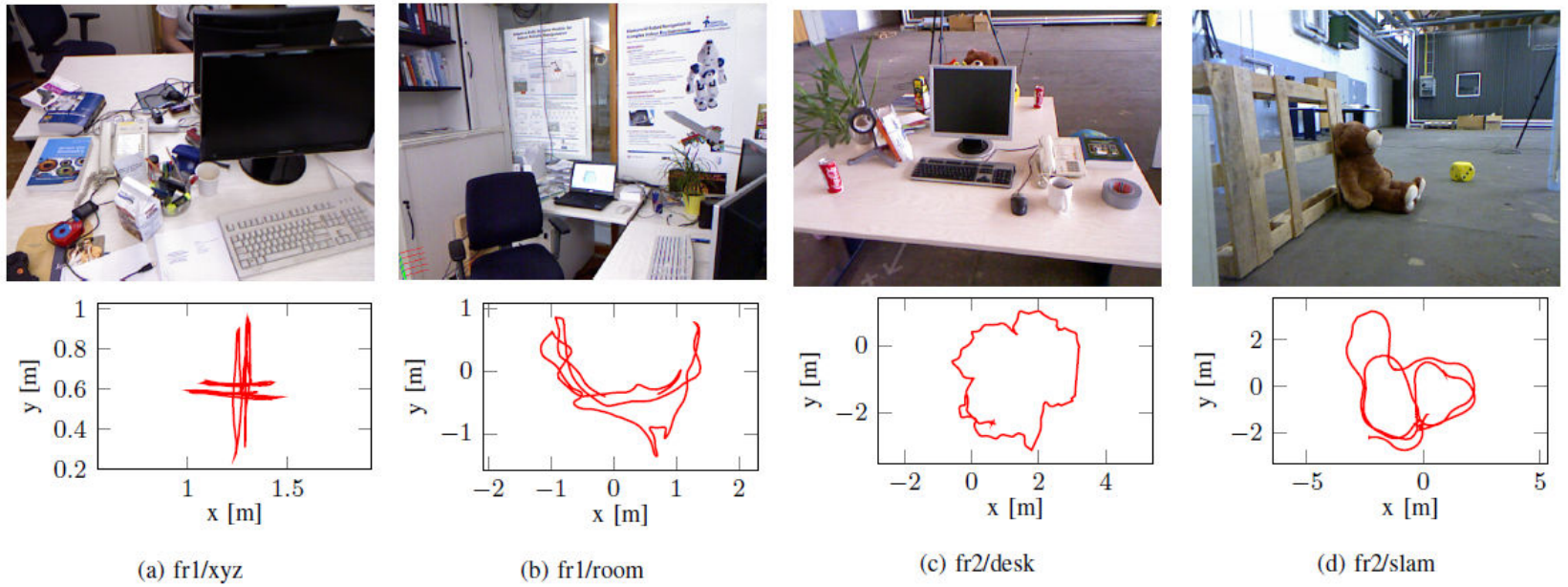
KITTI

RGB-D

Recorded Scenes

[Sturm et al., IROS 2012]

- Different environments (office, industrial hall, …)
- Large variations in camera speed, camera motion, illumination, number of features, dynamic objects, …
- Handheld and robot-mounted sensor



(a) fr1/xyz (b) fr1/room (c) fr2/desk (d) fr2/slam

Dr. Thomas Whelan, Dyson Robotics Lab, Imperial College London & Dr. Jürgen Sturm, Computer Vision Group, TUM

Dataset Acquisition

[Sturm et al., IROS 2012]

- Motion capture system
  - Camera pose (100 Hz)
- Microsoft Kinect (later: Asus Xtion Pro Live)
  - Color images (30 Hz)
  - Depth images (30 Hz)
- External video camera (for documentation)

Motion Capture System

[Sturm et al., IROS 2012]

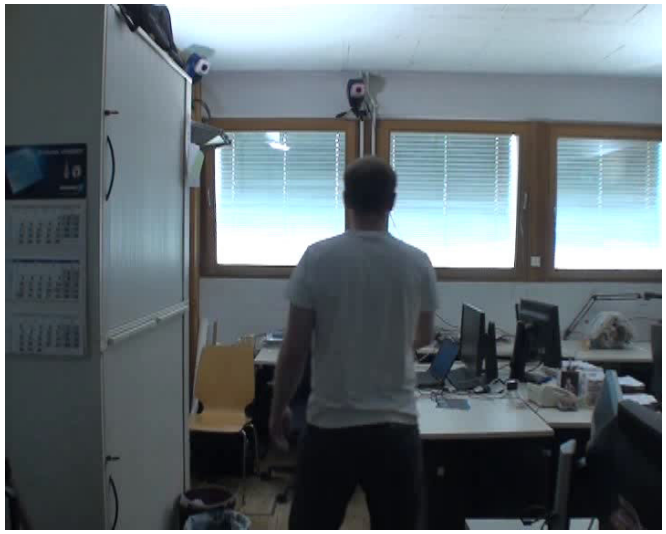
- 9 high-speed cameras mounted in room
- Cameras have active illumination and pre-process image (thresholding)
- Cameras track positions of retro-reflective markers

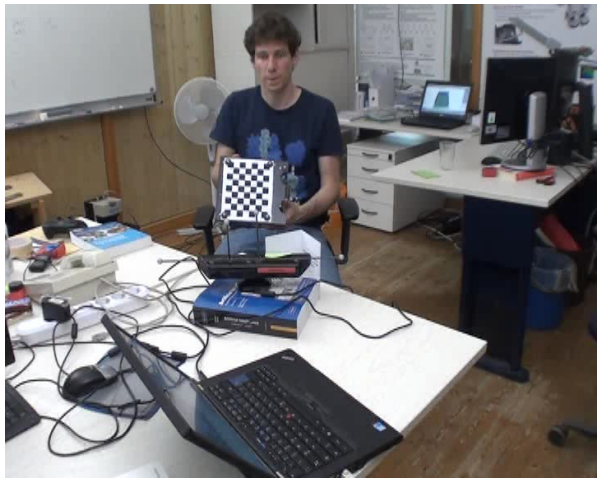
Calibration

[Sturm et al., IROS 2012]

Calibration of the overall system is not trivial:

1. Intrinsic calibration (Mocap + Kinect)
2. Extrinsic calibration (Kinect vs. Mocap)
3. Time synchronization (Kinect vs. Mocap)





Computer Vision Group - [×

vision.in.tum.de/data/datasets/rgbd-dataset

Computer Vision Group

Technische Universität München

Login

Search

- Home
- Publications
- Research
- **Datasets and Software**
- **Datasets**
  - Multiview Datasets
  - Deformable Shape Tracking Datasets
  - **RGB-D SLAM Dataset and Benchmark**
- Software
- Members
- Teaching
- Workshops
- Tutorials

Home › Datasets and Software › Datasets › **RGB-D SLAM Dataset and Benchmark**

RGB-D SLAM Dataset and Benchmark

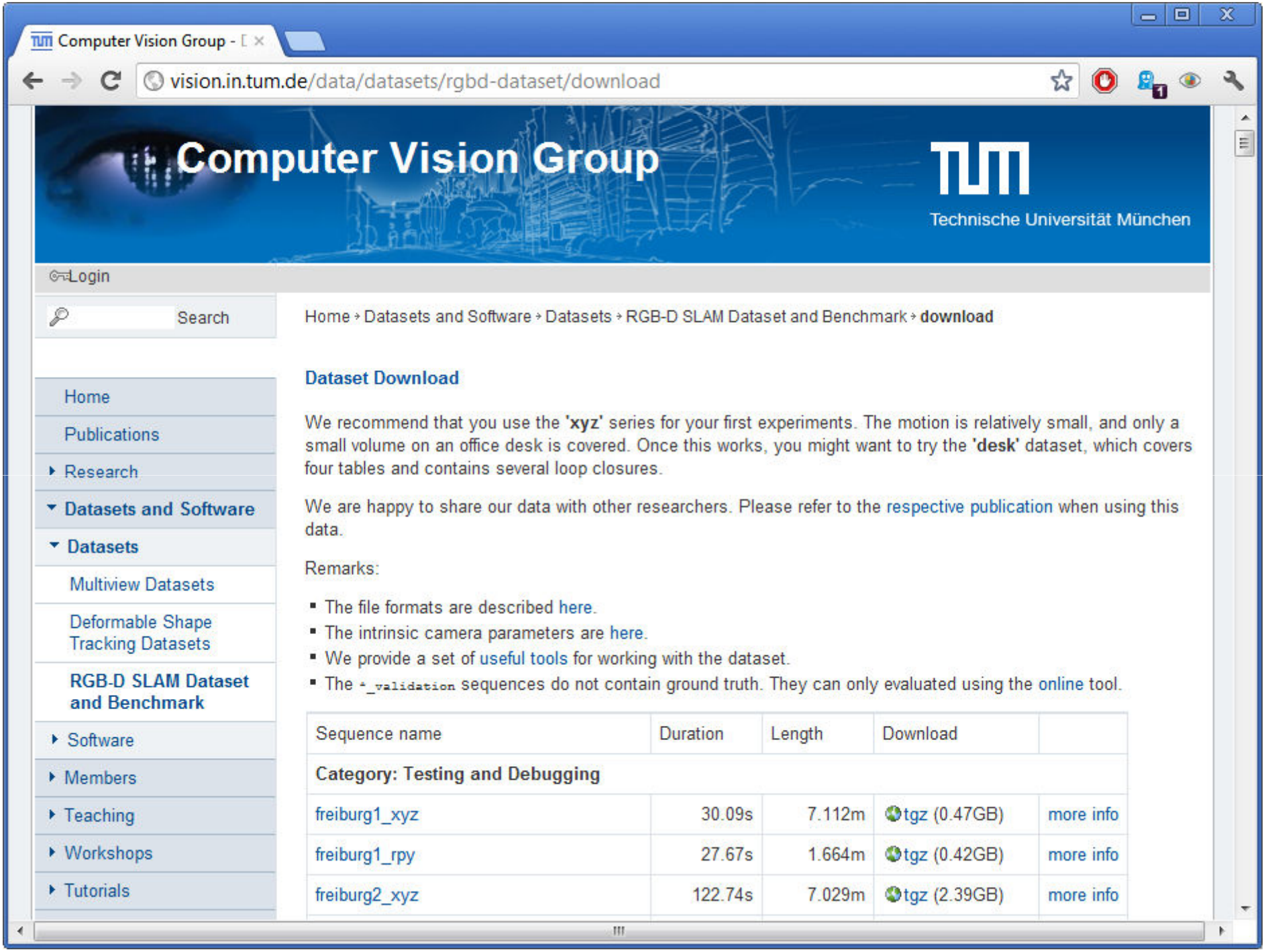
Quick Links
- Download page
- File formats
- Camera parameters
- Usful tools and scripts

Contact: Jürgen Sturm

We provide a large dataset containing RGB-D data and ground-truth data with the goal to establish a novel benchmark for the evaluation of visual odometry and visual SLAM systems. Our dataset contains the color and depth images of a Microsoft Kinect sensor along the ground-truth trajectory of the sensor. The data was recorded at full frame rate (30 Hz) and sensor resolution (640×480). The ground-truth trajectory was obtained from a high-accuracy motion-capture system with eight high-speed tracking cameras (100 Hz). Further, we provide the accelerometer data from the Kinect. Finally, we propose an evaluation criterion for measuring the quality of the estimated camera trajectory of visual SLAM systems.

How can I use the RGB-D Benchmark to evaluate my SLAM system?

1. Download one or more of the RGB-D benchmark sequences (file formats, useful tools)
2. Run your favorite visual odometry/visual SLAM algorithm (for example, RGB-D SLAM)
3. Save the estimated camera trajectory to a file (file formats, example trajectory)
4. Evaluate your algorithm by comparing the estimated trajectory with the ground truth trajectory. We provide an automated evaluation tool to help you with the evaluation. There is also an online version of the tool.


Computer Vision Group - [
vision.in.tum.de/data/datasets/rgbd-dataset/download
Computer Vision Group
Technische Universität München
Login
Search
Home
Publications
Research
Datasets and Software
Datasets
Multiview Datasets
Deformable Shape Tracking Datasets
RGB-D SLAM Dataset and Benchmark
Software
Members
Teaching
Workshops
Tutorials
Home › Datasets and Software › Datasets › RGB-D SLAM Dataset and Benchmark › download
Dataset Download
We recommend that you use the 'xyz' series for your first experiments. The motion is relatively small, and only a small volume on an office desk is covered. Once this works, you might want to try the 'desk' dataset, which covers four tables and contains several loop closures.
We are happy to share our data with other researchers. Please refer to the respective publication when using this data.
Remarks:
The file formats are described here.
The intrinsic camera parameters are here.
We provide a set of useful tools for working with the dataset.
The *_validation sequences do not contain ground truth. They can only evaluated using the online tool.
Sequence name
Duration
Length
Download
Category: Testing and Debugging
freiburg1_xyz 30.09s 7.112m tgz (0.47GB) more info
freiburg1_rpy 27.67s 1.664m tgz (0.42GB) more info
freiburg2_xyz 122.74s 7.029m tgz (2.39GB) more info


Computer Vision Group - [
vision.in.tum.de/data/datasets/rgbd-dataset/download
Computer Vision Group
Technische Universität München
Login
Search
Home
Publications
Research
Datasets and Software
Datasets
Multiview Datasets
Deformable Shape Tracking Datasets
RGB-D SLAM Dataset and Benchmark
Software
Members
Teaching
Workshops
Tutorials
freiburg1_xyz: RGB movie
Download this movie as avi
Related movies: RGB movie and, depth movie, external camera view
freiburg1_rpy 27.67s 1.664m tgz (0.42GB) more info
freiburg2_xyz 122.74s 7.029m tgz (2.39GB) more info

File Formats

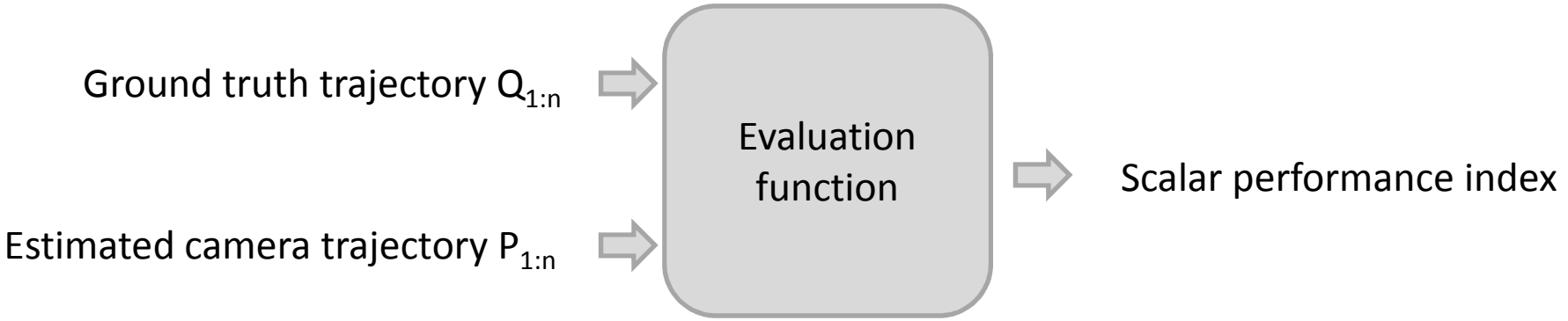
[Sturm et al., IROS 2012]

- In total: 69 sequences (33 training, 36 testing)
- One TGZ archive per sequence, containing
  - Color and depth images (PNG)
  - List of color images (timestamp filename)
  - List of depth images (timestamp filename)
  - List of camera poses (timestamp tx ty tz qx qy qz qw)

Evaluation Procedure

[Sturm et al., IROS 2012]

- Trajectory comparison
  - Ground truth trajectory $Q_1, \ldots, Q_n \in \mathrm{SE}(3)$
  - Estimated camera trajectory $P_1, \ldots, P_n \in \mathrm{SE}(3)$
- Evaluate
  - Drift per second
  - Global consistency



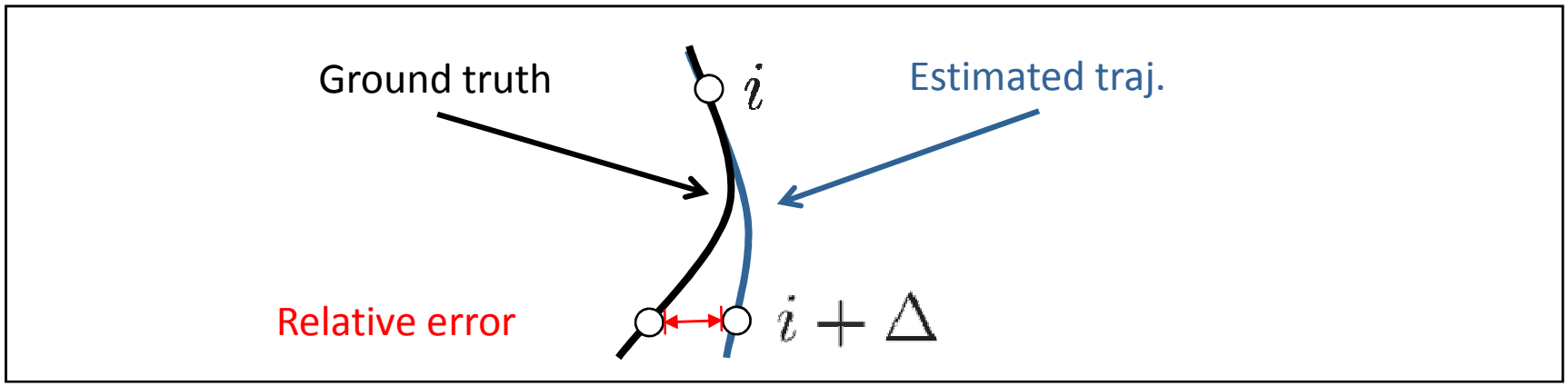
Relative Pose Error (RPE)

[Sturm et al., IROS 2012]

- Measures the (relative) drift between the i-th frame and the (i+Δ)-th frame

$$E_i := \left(Q_i^{-1} Q_{i+\Delta}\right)^{-1} \left(P_i^{-1} P_{i+\Delta}\right)$$

Relative error — True motion — Estimated motion



Relative Pose Error (RPE)

[Sturm et al., IROS 2012]

How to choose the time delta Δ?

- For odometry methods:
  - Δ=1: Drift per frame
  - Δ=30: Drift per second
- For SLAM methods:
  - Average over all possible deltas
  - Measures the global consistency

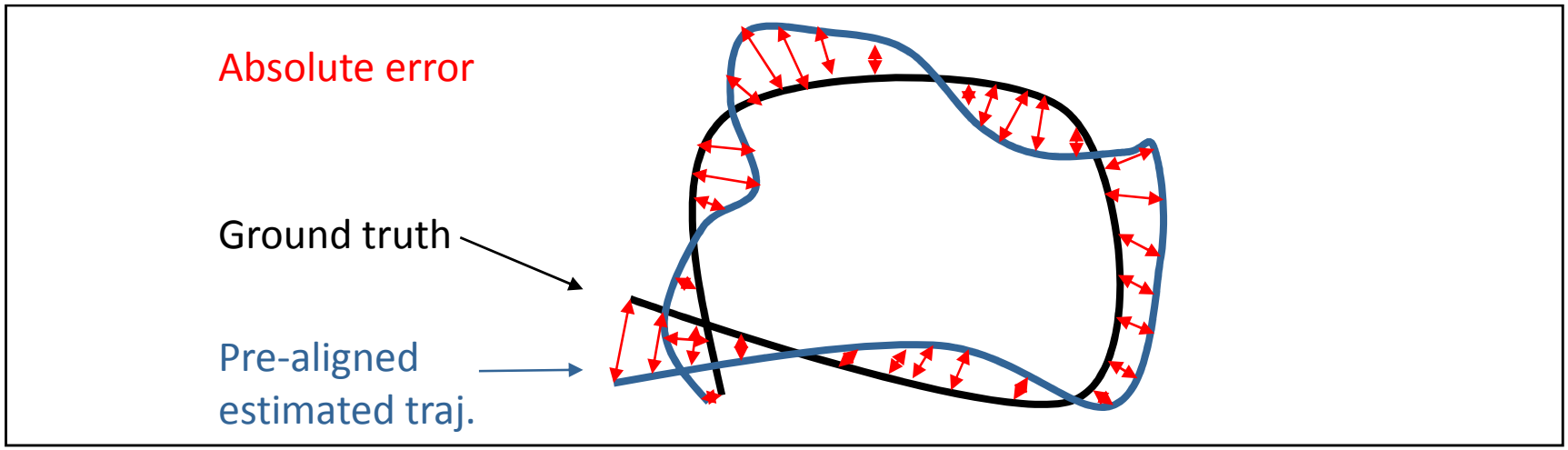
Absolute Trajectory Error (ATE)

[Sturm et al., IROS 2012]

- Alternative method to evaluate SLAM systems
- Requires pre-aligned trajectories

$$E_i := Q_i^{-1} S P_i$$

Absolute error — Groundtruth — Alignment — Estimated



Evaluation Tools

[Sturm et al., IROS 2012]

- Average over all time steps

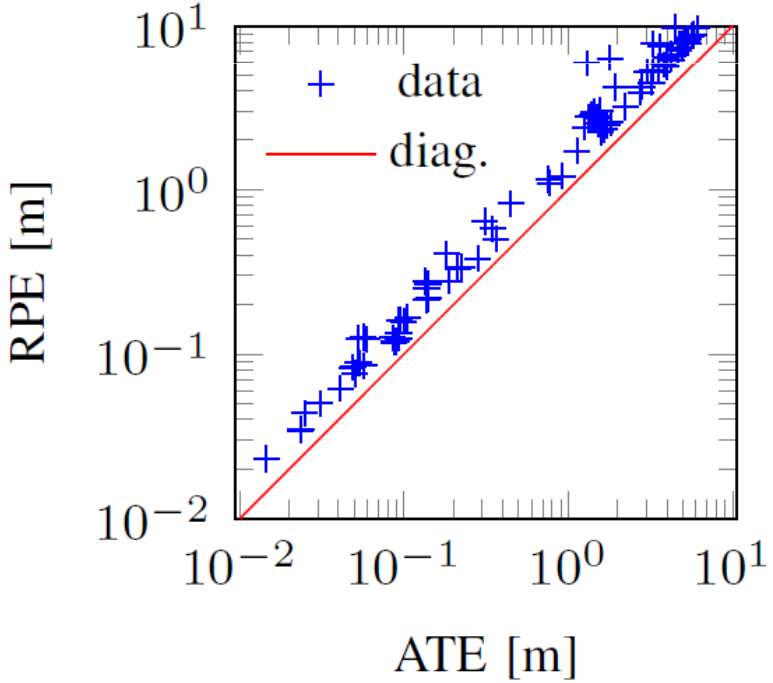
$$\mathrm{RMSE}(E_{1:n}) := \left(\frac{1}{m}\sum_{i=1}^{m} \|trans(E_i)\|^2\right)^{1/2}$$

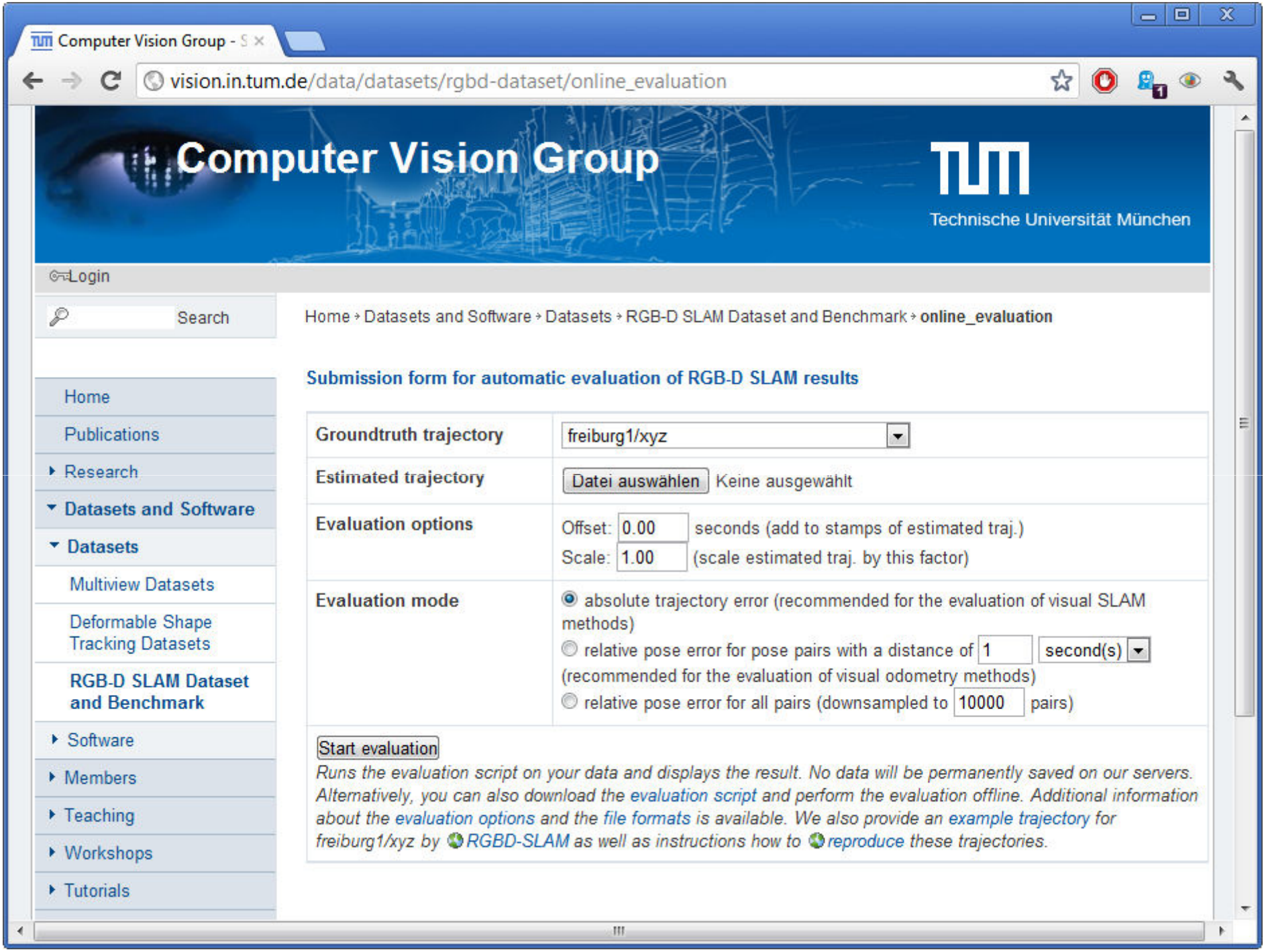
- Evaluation scripts for both evaluation metrics available (Python)
- Output: RMSE, median, mean
- Plot to png/pdf (optional)

Comparison of RPE and ATE

[Sturm et al., IROS 2012]

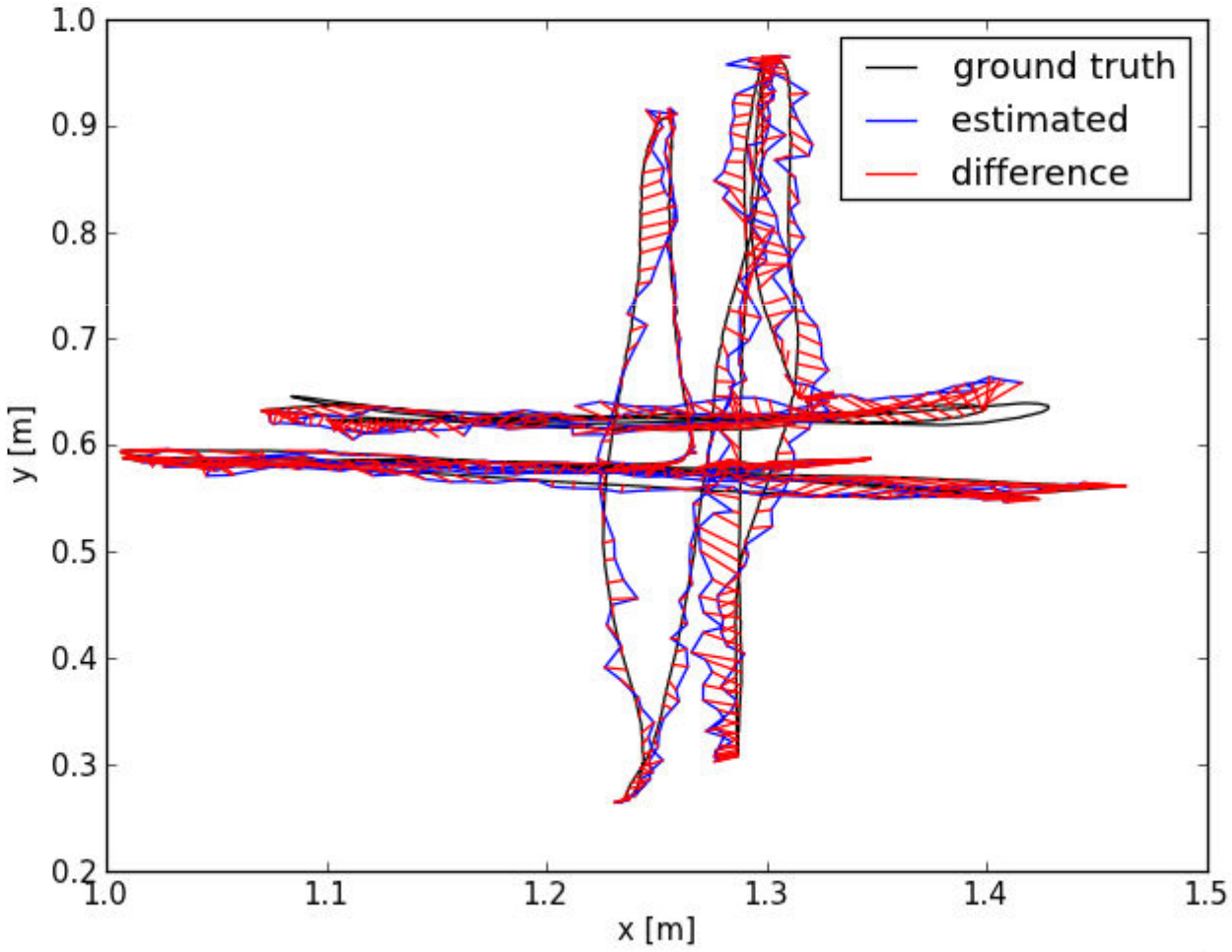
- RPE and ATE are strongly related
- RPE considers additionally rotational errors
- RPE ≥ ATE




Computer Vision Group - S
vision.in.tum.de/data/datasets/rgbd-dataset/online_evaluation
Computer Vision Group
Technische Universität München
Login
Search
Home
Publications
Research
Datasets and Software
Datasets
Multiview Datasets
Deformable Shape Tracking Datasets
RGB-D SLAM Dataset and Benchmark
Software
Members
Teaching
Workshops
Tutorials
Home › Datasets and Software › Datasets › RGB-D SLAM Dataset and Benchmark › online_evaluation
Submission form for automatic evaluation of RGB-D SLAM results
Groundtruth trajectory
freiburg1/xyz
Estimated trajectory
Datei auswählen
Keine ausgewählt
Evaluation options
Offset: 0.00 seconds (add to stamps of estimated traj.)
Scale: 1.00 (scale estimated traj. by this factor)
Evaluation mode
absolute trajectory error (recommended for the evaluation of visual SLAM methods)
relative pose error for pose pairs with a distance of 1 second(s)
(recommended for the evaluation of visual odometry methods)
relative pose error for all pairs (downsampled to 10000 pairs)
Start evaluation
Runs the evaluation script on your data and displays the result. No data will be permanently saved on our servers. Alternatively, you can also download the evaluation script and perform the evaluation offline. Additional information about the evaluation options and the file formats is available. We also provide an example trajectory for freiburg1/xyz by RGBD-SLAM as well as instructions how to reproduce these trajectories.

vision.in.tum.de/data/datasets/rgbd-dataset/online_evaluation

```
compared_pose_pairs 786 pairs
absolute_translational_error.rmse 0.013473 m
absolute_translational_error.mean 0.012029 m
absolute_translational_error.median 0.011176 m
absolute_translational_error.std 0.006068 m
absolute_translational_error.min 0.000939 m
absolute_translational_error.max 0.034727 m
```



Wrap-Up: TUM RGB-D Benchmark

[Sturm et al., IROS 2012]

- Dataset for the evaluation of RGB-D SLAM systems
- Ground-truth camera poses
- Evaluation metrics + tools available
- Since August 2011:
  - >24.000 unique page views
  - >4.500 online trajectory evaluations
  - >20 published research papers using the dataset
- Next steps:
  - Possibility to upload own trajectories/publications
  - Results page and automated ranking

Thank You For Your Attention

- Any questions?
- Coffee break

VISUM Summer School

RGB-D Cameras

Dr. Thomas Whelan

Imperial College London

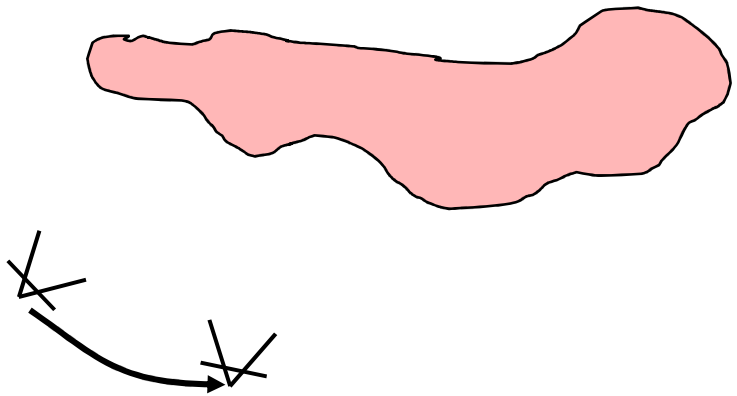
dyson

Outline: Second Session

- Pose graph optimization
- Dense tracking and 3D reconstruction
  - Motion estimation
  - Signed distance functions
  - Dense visual odometry
  - Large scale reconstruction and SLAM
  - Applications

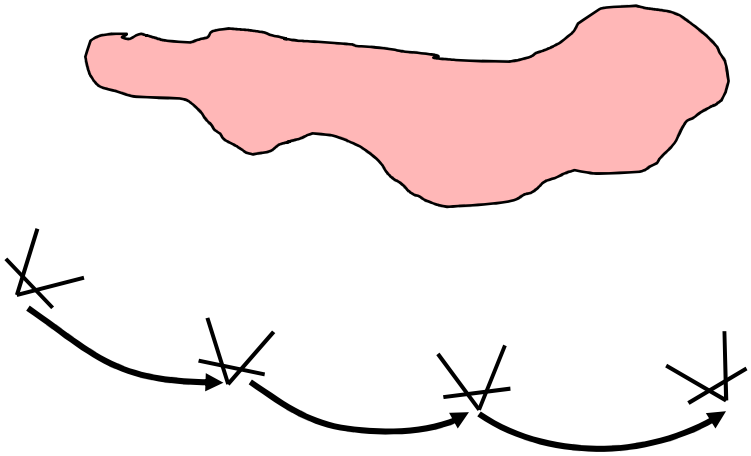
Incremental Motion Estimation

- **Idea:** Estimate camera motion from frame to frame



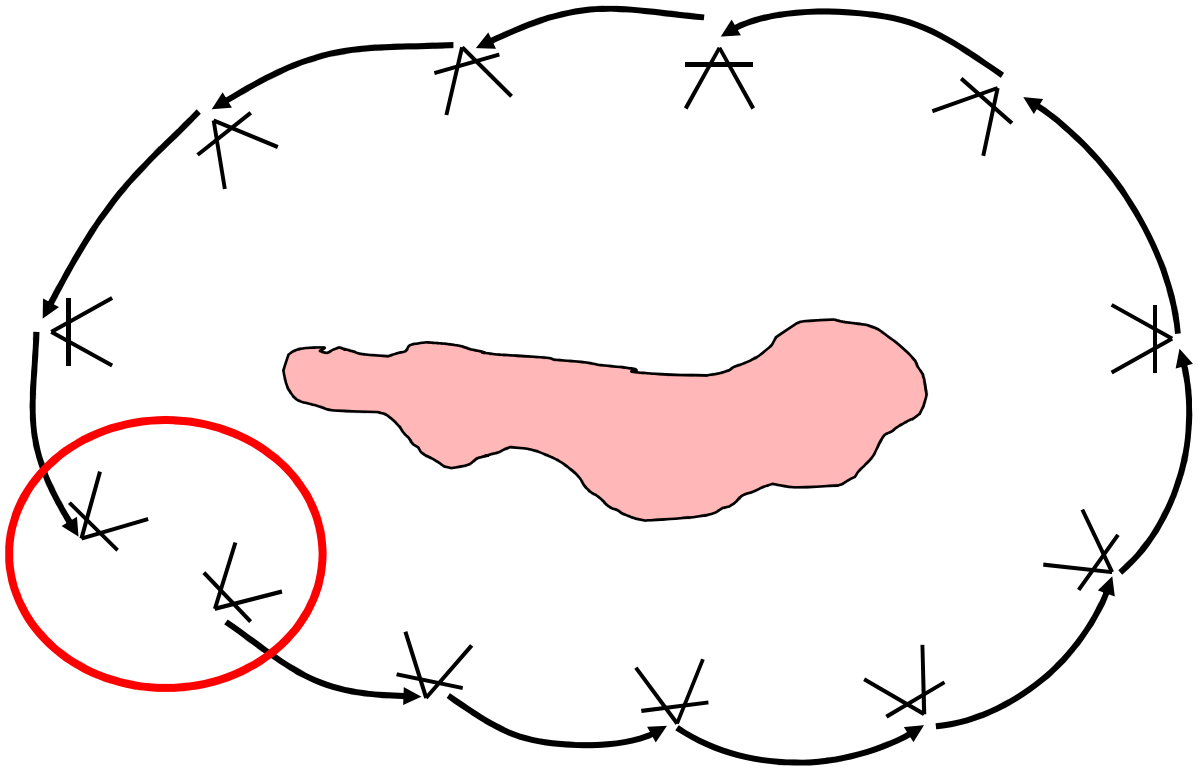
Incremental Motion Estimation

- **Idea:** Estimate camera motion from frame to frame



Incremental Motion Estimation

- **Idea:** Estimate camera motion from frame to frame

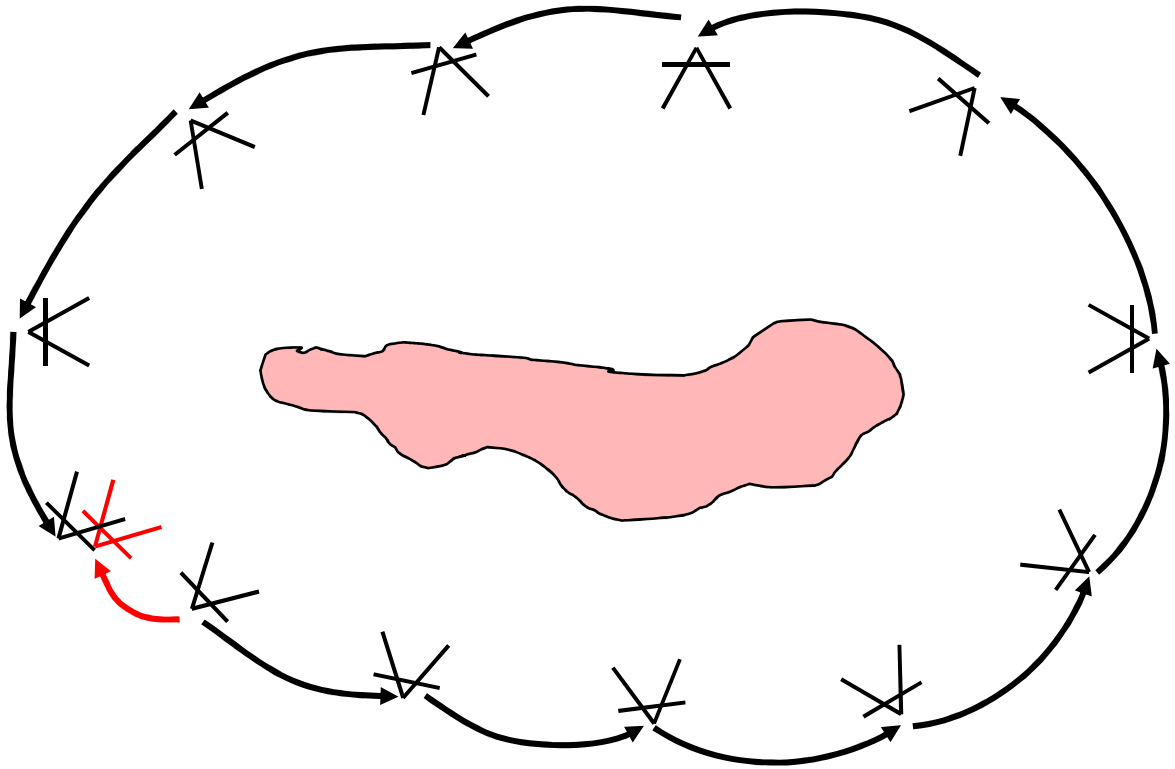


Loop Closures

- **Idea:** Estimate camera motion from frame to frame
- **Problem:**
  - Estimates are inherently noisy
  - Error accumulates over time → drift

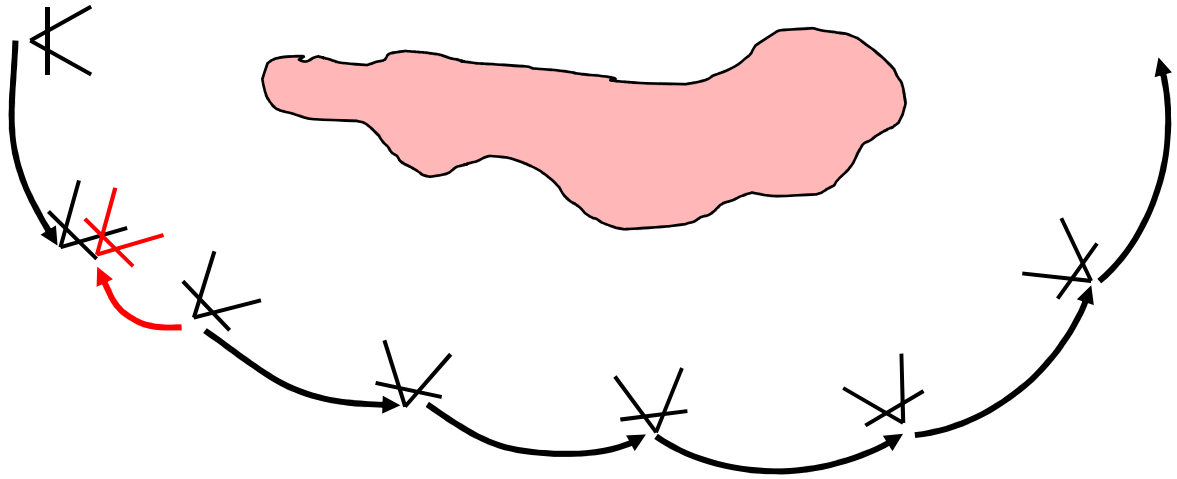
Incremental Motion Estimation

- **Idea:** Estimate camera motion from frame to frame



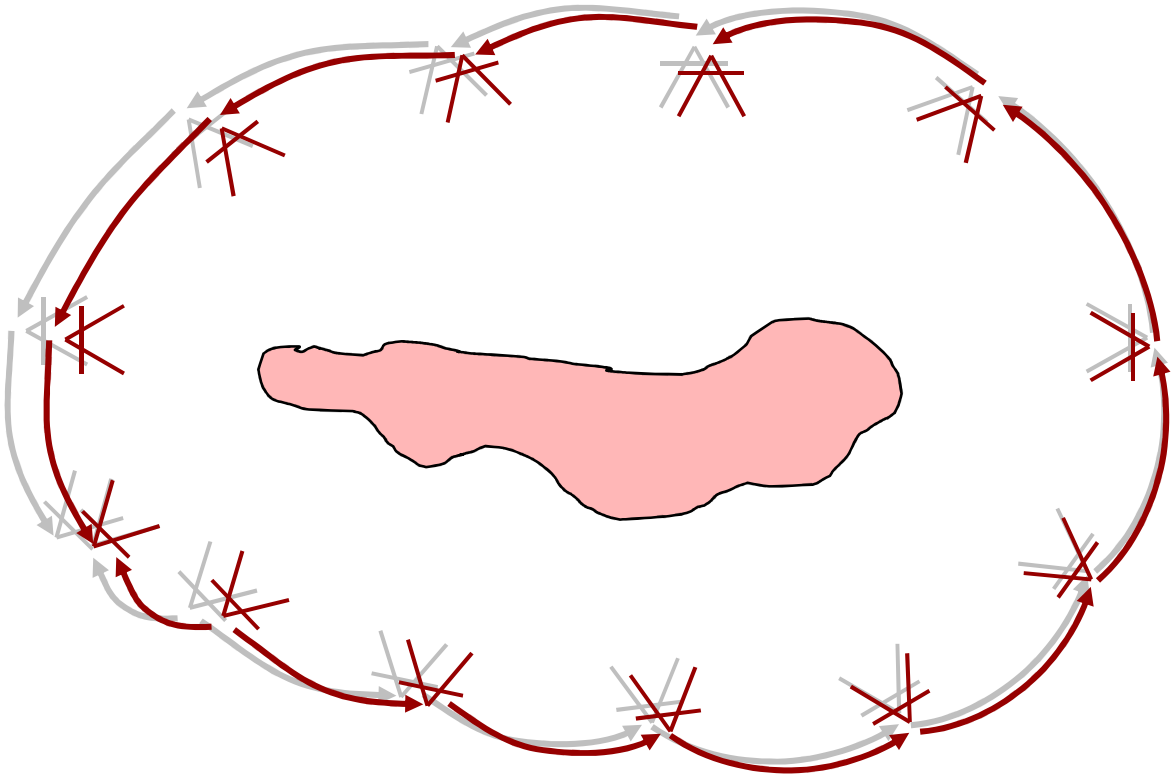
Closing the Loop

- **Idea:** Estimate camera motion from frame to frame
- What if we estimation additionally the motion between frame 1 and n?



Loop Closures

- **Solution:** Use loop-closures to minimize the drift / minimize the error over all constraints

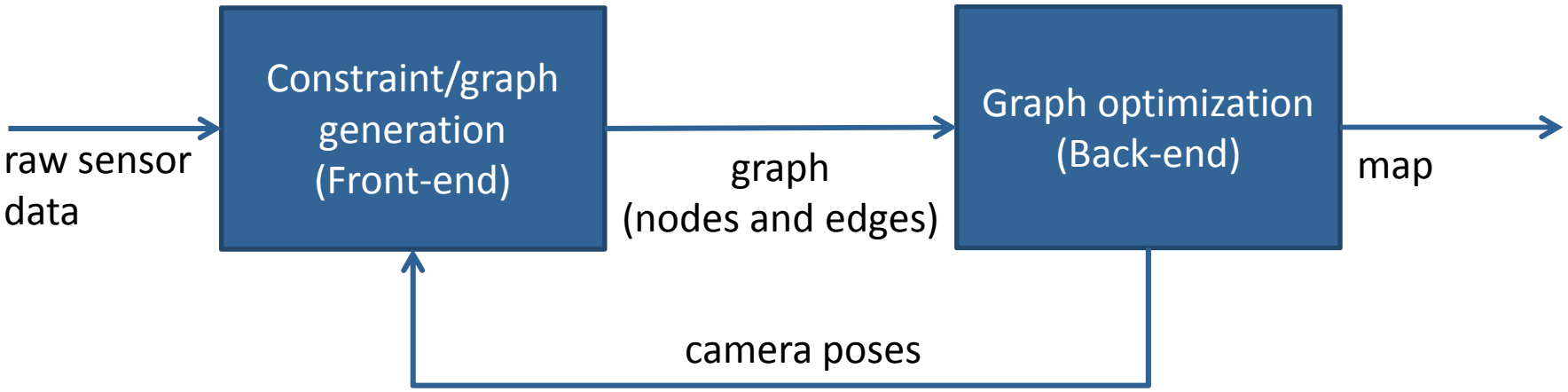


Graph SLAM

[Thrun and Montemerlo, 2006; Olson et al., 2006]

- Use a graph to represent the model
- Every **node** in the graph corresponds to a pose of the robot during mapping
- Every **edge** between two nodes corresponds to a spatial constraint between them
- **Graph-based SLAM:** Build the graph and find the robot poses that **minimize the error** introduced by the constraints

Graph SLAM Architecture

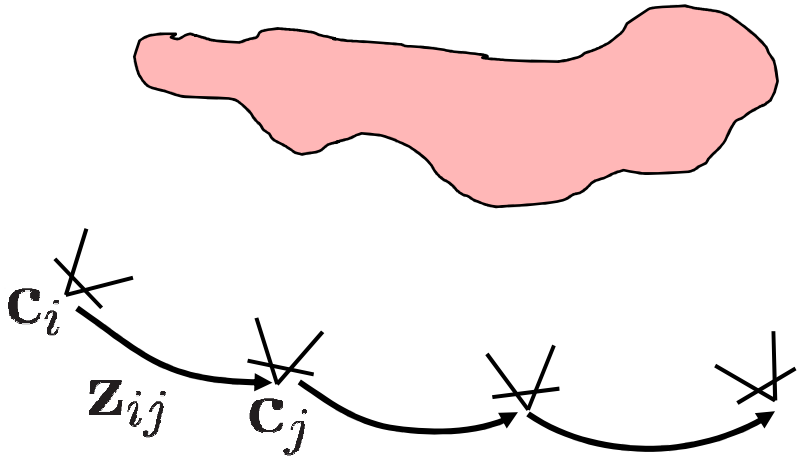


- Interleaving process of front-end and back-end
- A consistent map helps to determine new constraints by reducing the search space

Problem Definition

- **Given:** Set of relative pose observations $\mathbf{z}_{ij} \in \mathbb{R}^6$
- **Wanted:** Set of camera poses $\mathbf{c}_1, \ldots, \mathbf{c}_n \in \mathbb{R}^6$

  ➔ State vector $\mathbf{x} = (\mathbf{c}_1^\top, \ldots, \mathbf{c}_n^\top)^\top \in \mathbb{R}^{6n}$



Map Error

- Observation $\mathbf{z}_{ij}$
- Expected relative pose $\bar{\mathbf{z}}_{ij} = \mathbf{c}_j \ominus \mathbf{c}_i$

- Difference between observation and expectation

$$\mathbf{e}_{ij} = \mathbf{z}_{ij} \ominus \bar{\mathbf{z}}_{ij}$$

- Given the correct map $\mathbf{x}$, this difference is the result of observation/sensor noise…

Error Function

- **Assumption:** Observation noise is normally distributed

$$\mathbf{e}_{ij} \sim \mathcal{N}(\mathbf{0}, \Sigma_{ij})$$

- Error term for one observation (proportional to negative loglikelihood)

$$f_{ij}(\mathbf{x}) = -\log p(\mathbf{e}_{ij}) \propto \mathbf{e}_{ij}(\mathbf{x})^\top \Sigma_{ij}^{-1} \mathbf{e}_{ij}(\mathbf{x})$$

- Note: error is a scalar $f_{ij}(\mathbf{x}) \in \mathbb{R}$

Error Function

- Map error (over all observations)

$$f(\mathbf{x}) = \sum_{ij} f_{ij}(\mathbf{x}) = \sum_{ij} \mathbf{e}_{ij}(\mathbf{x})^\top \Sigma_{ij}^{-1} \mathbf{e}_{ij}(\mathbf{x})$$

- **Minimize this error** by optimizing the camera poses

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} \sum_{ij} \mathbf{e}_{ij}(\mathbf{x})^\top \Sigma_{ij}^{-1} \mathbf{e}_{ij}(\mathbf{x})$$

- How can we solve this optimization problem?

Non-Linear Optimization Techniques

- Gradient descend
- Gauss-Newton
- Levenberg-Marquardt

Gauss-Newton Method

1. Linearize the error function
2. Compute its derivative
3. Set the derivative to zero
4. Solve the linear system
5. Iterate this procedure until convergence

Linearization and Derivation

- Error function

$$f(\mathbf{x}) = \sum_{ij} f_{ij}(\mathbf{x}) = \sum_{ij} \mathbf{e}_{ij}(\mathbf{x})^\top \Sigma_{ij}^{-1} \mathbf{e}_{ij}(\mathbf{x})$$

- Linearize the error function around the initial guess

$$f(\mathbf{x}+\Delta\mathbf{x}) = \sum_{ij} \mathbf{e}_{ij}(\mathbf{x}+\Delta\mathbf{x})^\top \Sigma_{ij}^{-1} \mathbf{e}_{ij}(\mathbf{x}+\Delta\mathbf{x})$$

Let's look at this term first…

Linearizing the Error Function

- Approximate the error function around an initial guess $\mathbf{x} \in \mathbb{R}^{6n}$ using Taylor expansion

$$\mathbf{e}_{ij}(\mathbf{x} + \Delta\mathbf{x}) \simeq \mathbf{e}_{ij}(\mathbf{x}) + J_{ij}\Delta\mathbf{x} \qquad (\in \mathbb{R}^6)$$

with increment

$$\Delta\mathbf{x} \in \mathbb{R}^{6n}$$

and Jacobian

$$J_{ij}(\mathbf{x}) = \left(\frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{c}_1} \quad \frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{c}_2} \quad \cdots \quad \frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{c}_n} \right) \in \mathbb{R}^{6\times 6n}$$

Linearizing the Error Function

Linearize $f(\mathbf{x}) = \sum_{ij} \mathbf{e}_{ij}(\mathbf{x})^T \Sigma_{ij}^{-1} \mathbf{e}_{ij}(\mathbf{x})$

$$\simeq \mathbf{c} + 2\mathbf{b}^\top \Delta\mathbf{x} + \Delta\mathbf{x}^\top H \Delta\mathbf{x}$$

with $\mathbf{b}^\top = \sum_{ij} \mathbf{e}_{ij}^\top \Sigma_{ij}^{-1} J_{ij}$

$$H = \sum_{ij} J_{ij}^\top \Sigma_{ij}^{-1} J_{ij}$$

(Linear) Least Squares Minimization

1. Linearize error function

$$f(\mathbf{x} + \Delta\mathbf{x}) \simeq \mathbf{c} + 2\mathbf{b}^\top \Delta\mathbf{x} + \Delta\mathbf{x}^\top H \Delta\mathbf{x}$$

2. Compute the derivative

$$\frac{\mathrm{d} f(\mathbf{x} + \Delta\mathbf{x})}{\mathrm{d}\Delta\mathbf{x}} = 2\mathbf{b} + 2H\Delta\mathbf{x}$$

3. Set derivative to zero

$$H\Delta\mathbf{x} = -\mathbf{b}$$

4. Solve this linear system of equations, e.g.,

$$\Delta\mathbf{x} = -H^{-1}\mathbf{b}$$

Gauss-Newton Method

Problem: $f(\mathbf{x})$ is non-linear!

Algorithm: Repeat until convergence

1. Compute the terms of the linear system

$$\mathbf{b}^\top = \sum_{ij} \mathbf{e}_{ij}^\top \Sigma_{ij}^{-1} J_{ij} \qquad H = \sum_{ij} J_{ij}^\top \Sigma_{ij}^{-1} J_{ij}$$

2. Solve the linear system to get new increment

$$H \Delta \mathbf{x} = -\mathbf{b}$$

3. Update previous estimate

$$\mathbf{x} \leftarrow \mathbf{x} + \Delta \mathbf{x}$$

Levenberg-Marquardt Algorithm

- **Observations:**
  - Gauss-Newton method typically converges very quickly
  - Sometimes diverges when initial solution is far off
  - Gradient descent (with line search) never diverges
- **How can we combine the advantages of both minimization methods?**

Levenberg-Marquardt Algorithm

- **Idea:** Add a damping factor

$$(H + \lambda I)\Delta\mathbf{x} = -\mathbf{b}$$

$$(J^\top J + \lambda I)\Delta\mathbf{x} = -J^\top \mathbf{e}$$

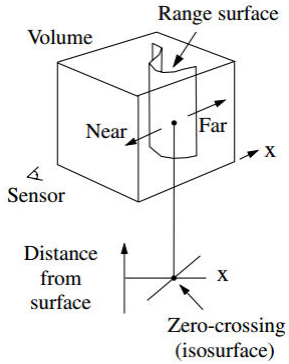
- What is the effect of this damping factor?
  - Small λ → same as least squares
  - Large λ → steepest descent (with small step size)
- **Algorithm**
  - If error decreases, accept $\Delta\mathbf{x}$ and reduce λ
  - If error increases, reject $\Delta\mathbf{x}$ and increase λ

Non-Linear Minimization

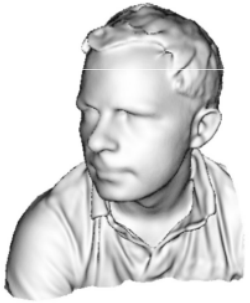
- One of the state-of-the-art solution to compute the maximum likelihood estimate
- Various open-source implementations available
  - g2o [Kuemmerle et al., 2011]
  - sba [Lourakis and Argyros, 2009]
  - iSAM [Kaess et al., 2008]
  - Ceres [Google, 2012]
- Other extensions:
  - Robust error functions
  - Alternative parameterizations

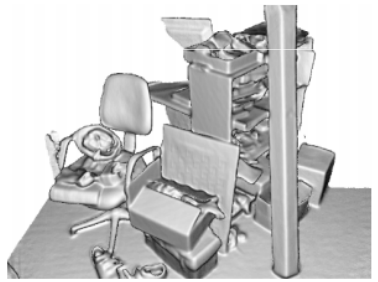
Dense Mapping

- Curless and Levoy (SIGGRAPH 1996)

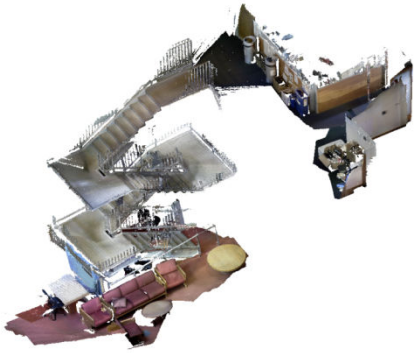


- Newcombe et al. (KinectFusion, ISMAR 2011)



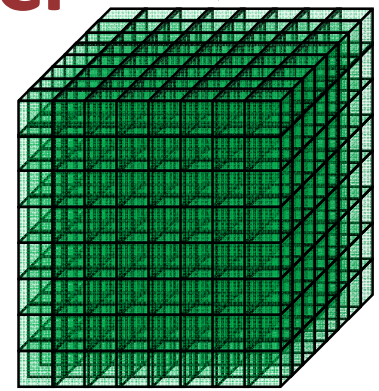


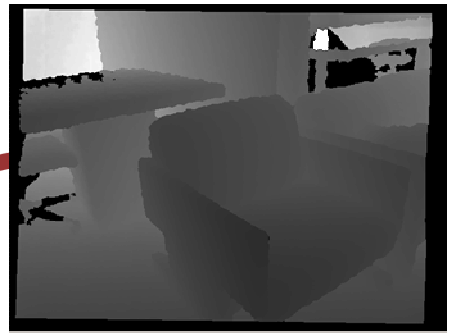
- Whelan et al. (Kintinuous, RSS 2012, ICRA 2013)



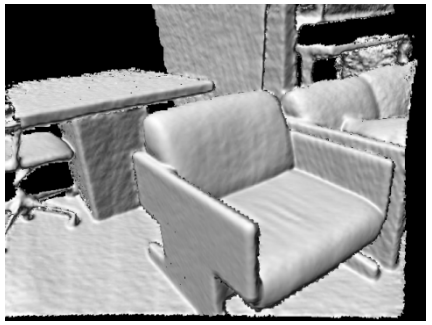
KinectFusion (Newcombe et al. ’11)

ICP





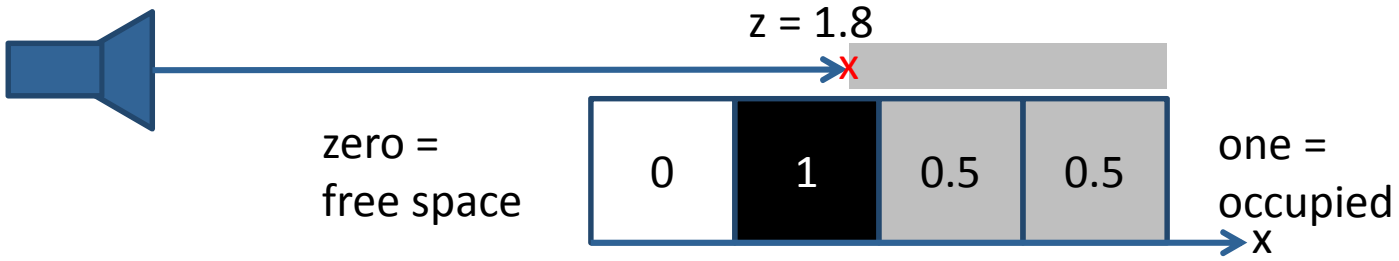
Depth Image



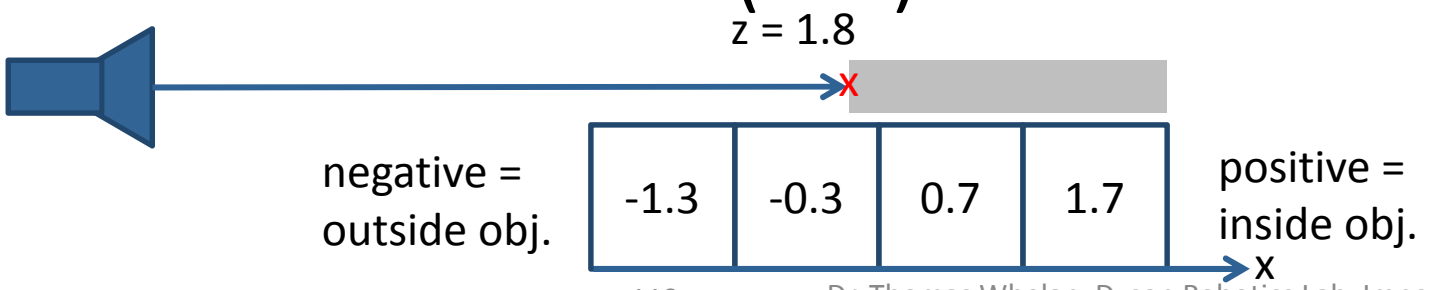
Volumetric Model (TSDF)

Representation of the 3D Model

- **Idea:** Instead of representing the cell occupancy, represent the distance of each cell to the surface (Levoy and Curless, 1996)
- Occupancy grid maps



- Signed distance function (SDF)



KinectFusion (Newcombe et al. ’11)

- Real-time
  - GPU is the enabler here
- Unprecedented quality
  - Essentially a moving average
- Some limitations
  - Limited area in space
  - Geometry only for camera pose estimation
  - No correction for drift
    - This is a grey area

Kintinuous (Whelan et al. ‘12, ‘13)

- Solve some of the limitations of KinectFusion
  - Make it more useful for robotics and SLAM

- Multiple goals
  - Overcome the three key limitations of
    - Limited area in space
    - Geometry only for camera pose estimation
    - No correction for drift
  - Maintain online operation

Kintinuous 1.0 (Whelan et al., RSS RGB-D '12)

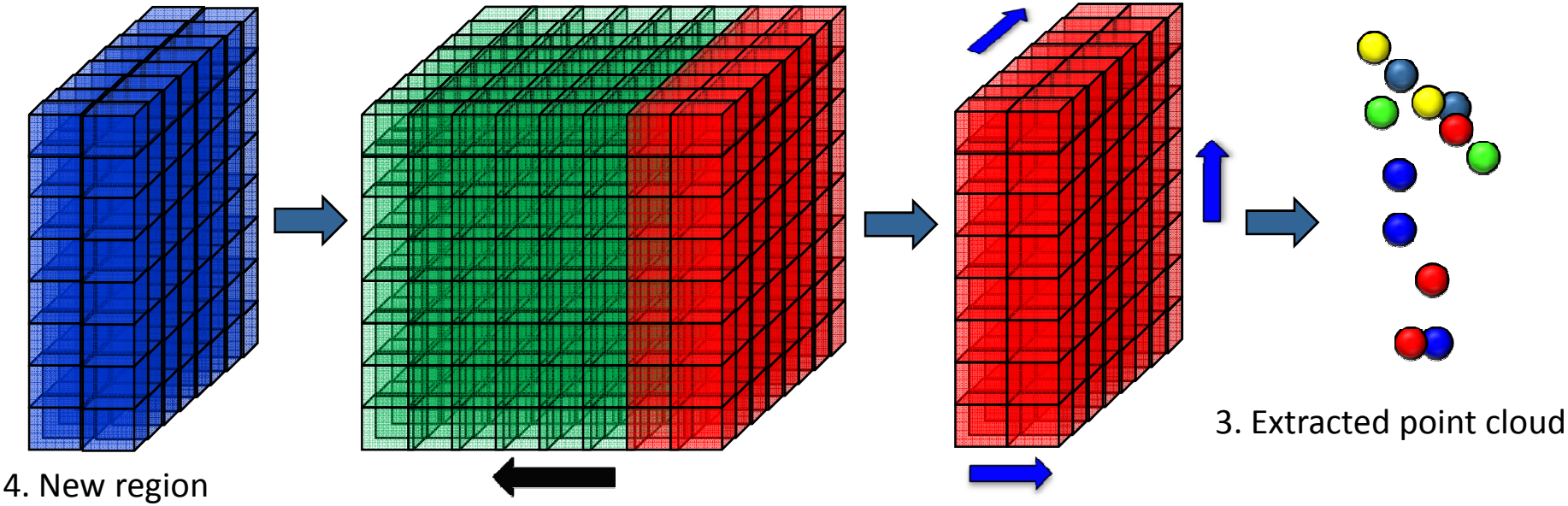
- Addresses issue of space restriction
  - Real-time
  - Alternative visual odometry (preliminary)
    - FOVIS dynamic switching [Huang et al. '11]

- "Kintinuous: Spatially Extended KinectFusion" by T. Whelan, M. Kaess, M.F. Fallon, H. Johannsson, J.J. Leonard and J.B. McDonald.
  RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras, July 2012.

Kintinuous 1.0 (Whelan et al., RSS RGB-D '12)

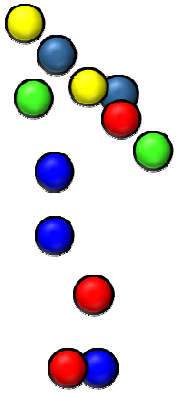
- Treat volumetric model as a cyclical buffer.
  - As region leaves the range of the buffer, extract the corresponding surface data.
  - As region enters the range of the buffer, initialise and track the new data.

Kintinuous 1.0 (Whelan et al., RSS RGB-D '12)



Kintinuous 1.0 (Whelan et al., RSS RGB-D '12)

- The result is a set of point cloud "slices" of the TSDF volume.



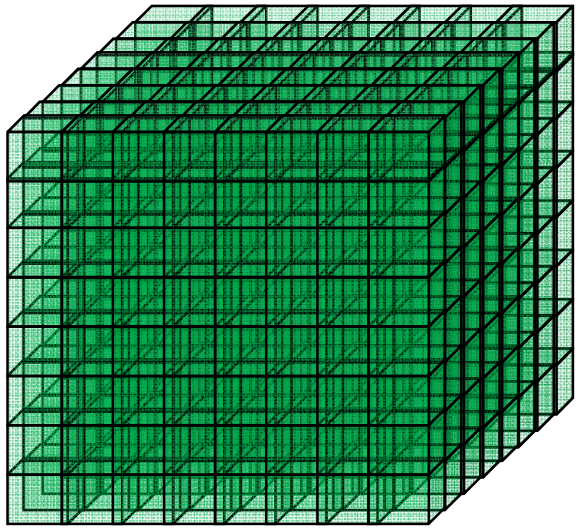
Extracted point cloud

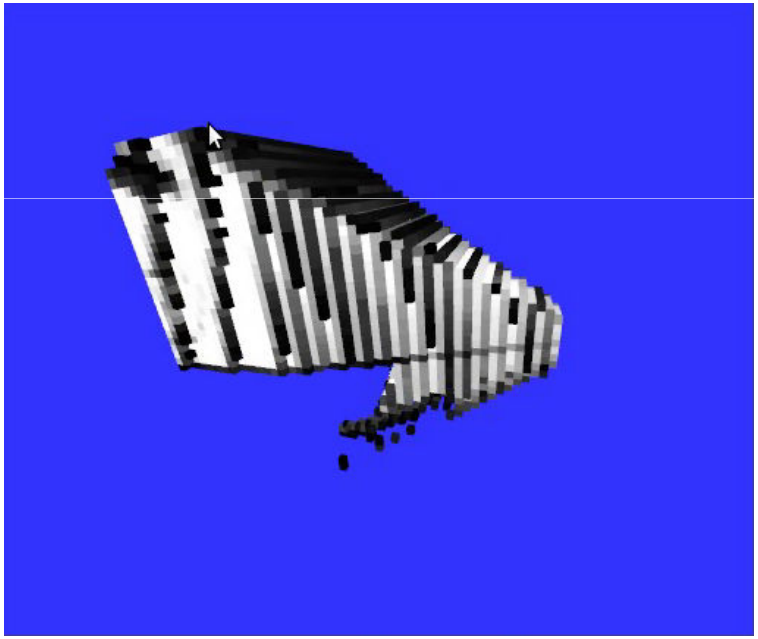
Marton *et al.* '09

Greedy mesh triangulation

Dense triangular mesh

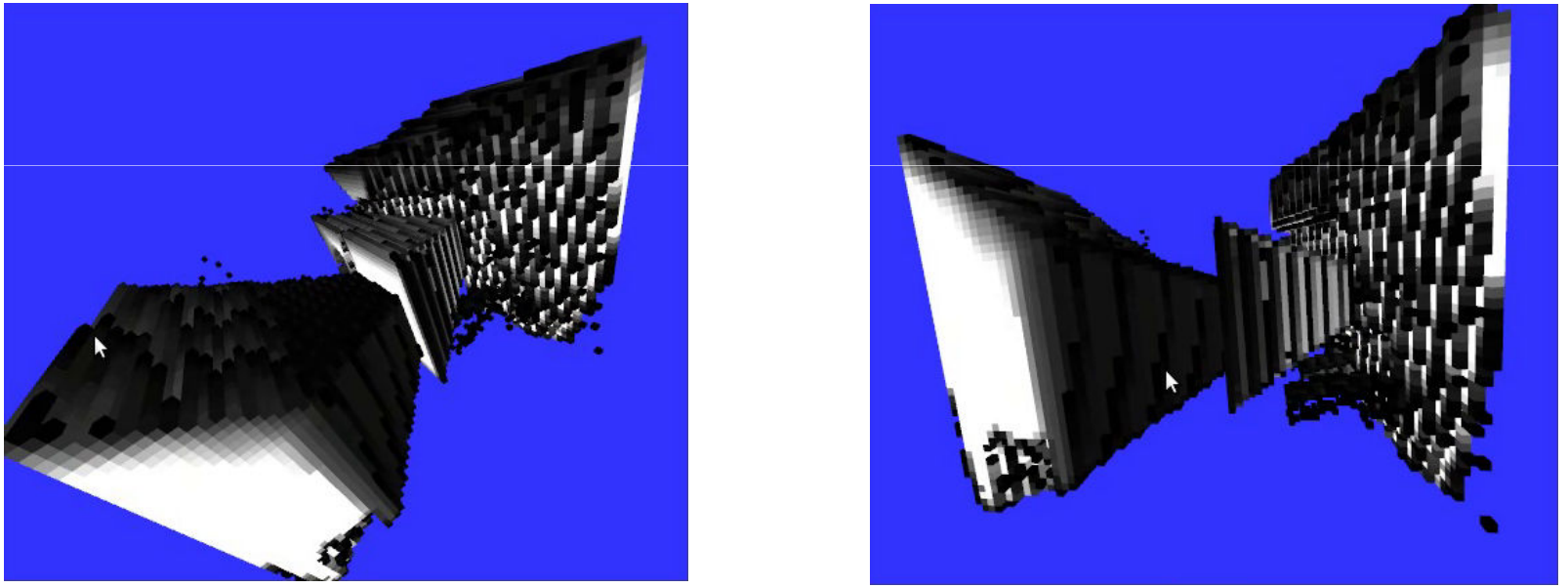
Kintinuous 1.0 (Whelan et al., RSS RGB-D '12)





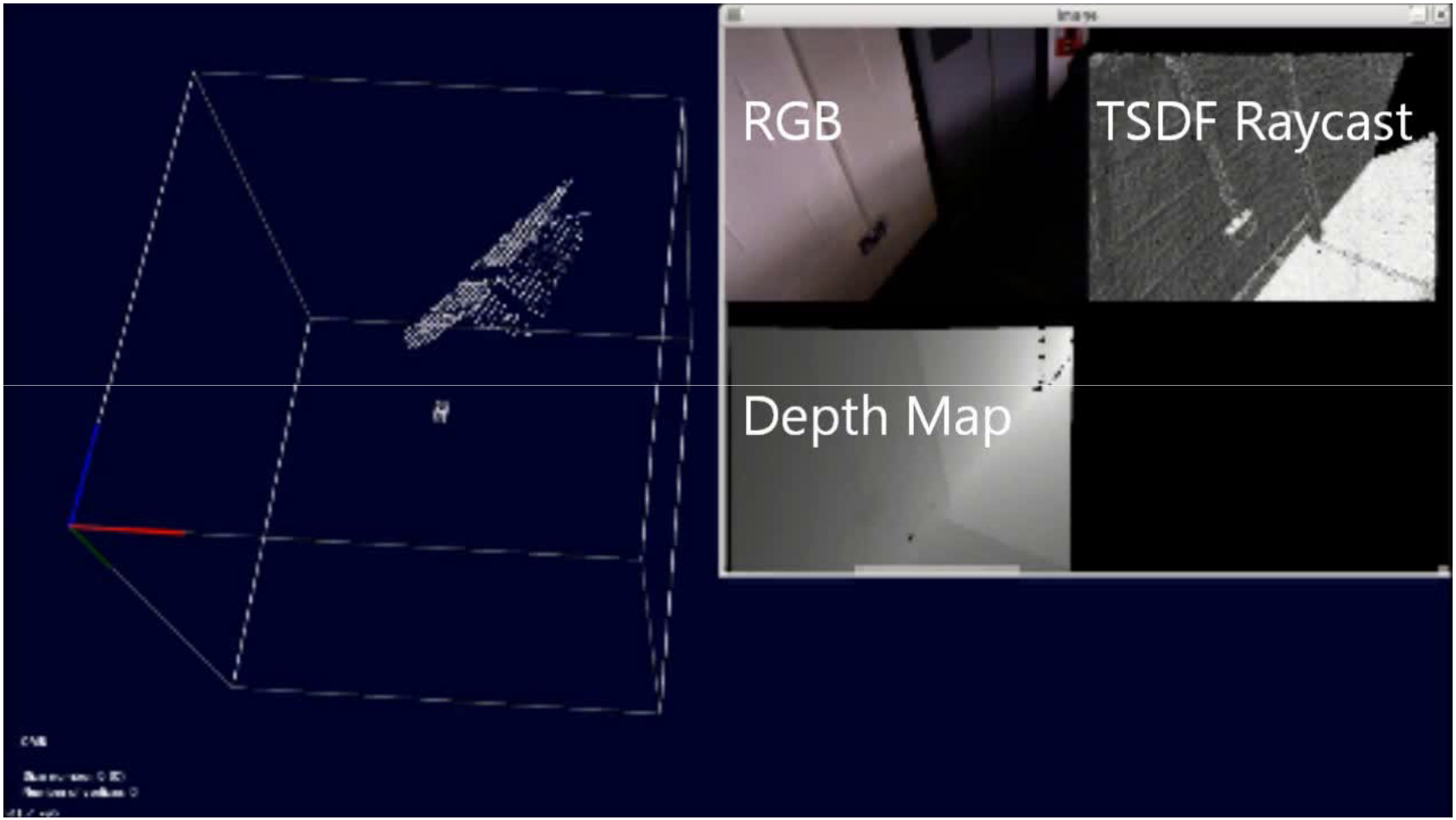
Actual volume projection

Kintinuous 1.0 (Whelan et al., RSS RGB-D '12)



Volume projections after cycling in Z dimension

Kintinuous 1.0 (Whelan et al., RSS RGB-D '12)



Kintinuous 1.0 (Whelan et al., RSS RGB-D '12)

- Who else extended the range?
- Heredia & Favier '12
  - Open sourced PCL "kinfu_large_scale"
  - Developed independently and parallel to Kintinuous.
- Roth & Vona, BMVC '12
  - Use double buffering with rotation remapping.
- Zeng et al., CVM '12
  - Range extended to 8x8x8m using an octree.
- Many more since (Microsoft Research, TUM)

Kintinuous 1.5 (Whelan et al., ICRA '13)

- Addresses issue of geometry reliant odometry
  - Real-time
  - GPU-based
  - Also improves fusion of colour in reconstruction

- "Robust Real-Time Visual Odometry for Dense RGB-D Mapping" by T. Whelan, H. Johannsson, M. Kaess, J.J. Leonard, and J.B. McDonald.
  IEEE Intl. Conf. on Robotics and Automation, ICRA, May 2013.

Kintinuous 1.5 (Whelan et al., ICRA '13)

- Motivation
  - ICP fails if
    - Depth lacks strong geometric features
    - No geometric features fall within the TSDF
  - Colour integration is poor if
    - RGB/Depth registration is inaccurate
    - Angle of incidence is extreme

- Essentially we observed performance in both SLAM-style applications and 3D model acquisition could be improved upon.

Kintinuous 1.5 (Whelan et al., ICRA '13)

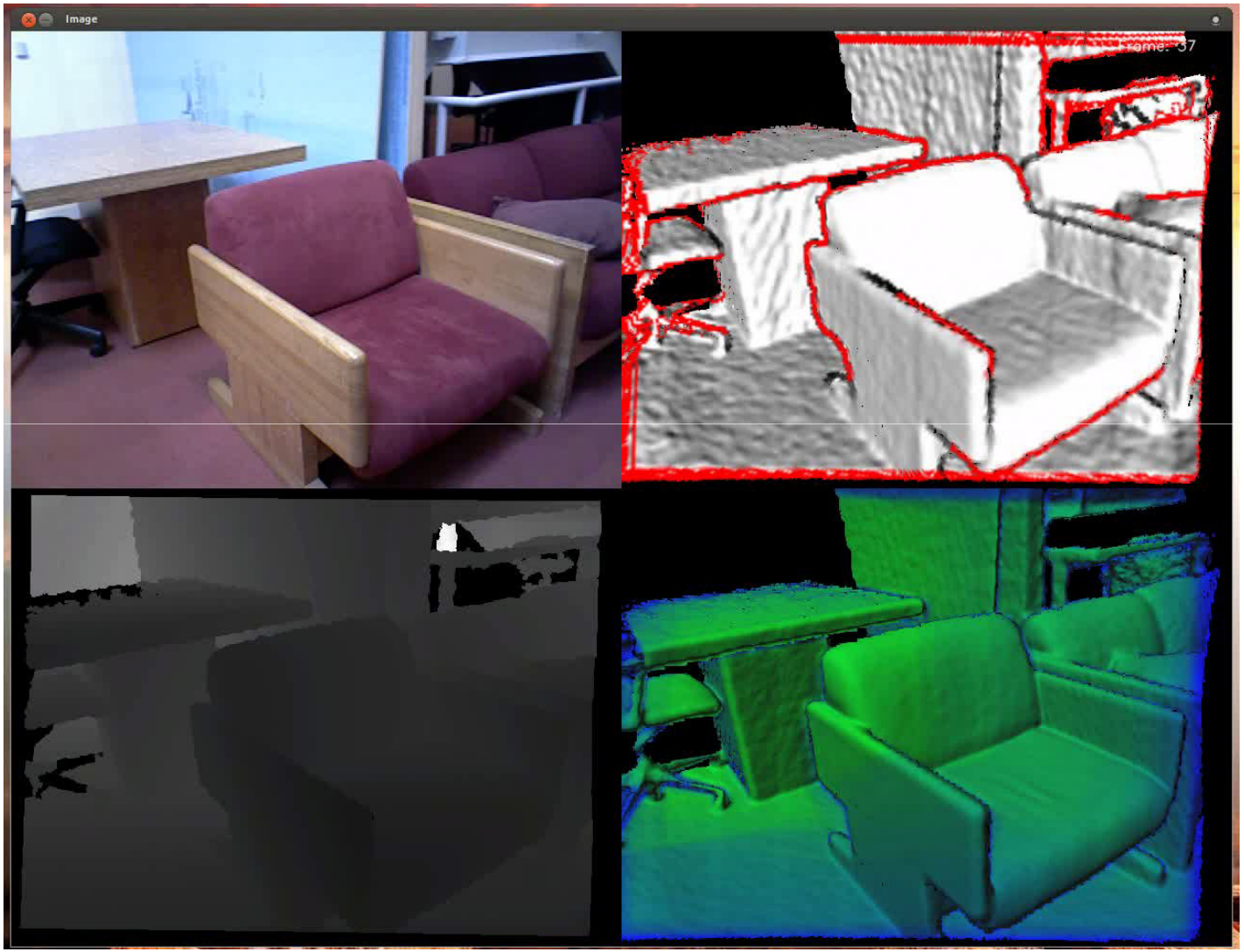
- Colour fusion
  - First introduced by Anatoly Baksheev into open sourced PCL "kinfu" – not widely known.
  - Second TSDF used to stored RGB colour components, only used for integration, not registration.
  - Otherwise identical to depth / surface fusion.
  - Usually has colour "bleeding" artefacts around edges of objects.
    - Coincides with depth discontinuities and poor angles of incidence.

Kintinuous 1.5 (Whelan et al., ICRA '13)

- Artefacts remedied by
  - Rejecting colour measurements on boundaries in the depth image.
    - 7x7 window is inspected around each pixel.
  - Weighting the integration by the angle of incidence.

$$cos(\theta) = [0 \;\; 0 \;\; 1] \cdot \mathbf{n} = n_z$$

Kintinuous 1.5 (Whelan et al., ICRA ‘13)



Kintinuous 1.5 (Whelan et al., ICRA ‘13)

Kintinuous 1.5 (Whelan et al., ICRA '13)

- Merging of geometric and photometric data
  - Combines photometric warping function based on intensity correspondences of Steinbrueker et al. with ICP
  - Done at the normal equation level
  - Computationally cheap
  - Easy to weight

$$(\mathbf{J}_{icp}^{\top}\mathbf{J}_{icp} + w\mathbf{J}_{rgbd}^{\top}\mathbf{J}_{rgbd})\xi = \mathbf{J}_{icp}^{\top}\mathbf{r}_{icp} + v\mathbf{J}_{rgbd}^{\top}\mathbf{r}_{rgbd}$$

Kintinuous 1.5 (Whelan et al., ICRA '13)

- GPU-ification of Steinbruecker et al.
  - Two parallel reductions
    - 2D to 1D to extract list of pixels with high gradient
    - 1D to 1D to register valid correspondences
  - Products computed with same tree reduction implementation of Gauss-Newton as ICP in KinectFusion
    - Really fast
    - Cholesky done on CPU, cheap 6x6 solve

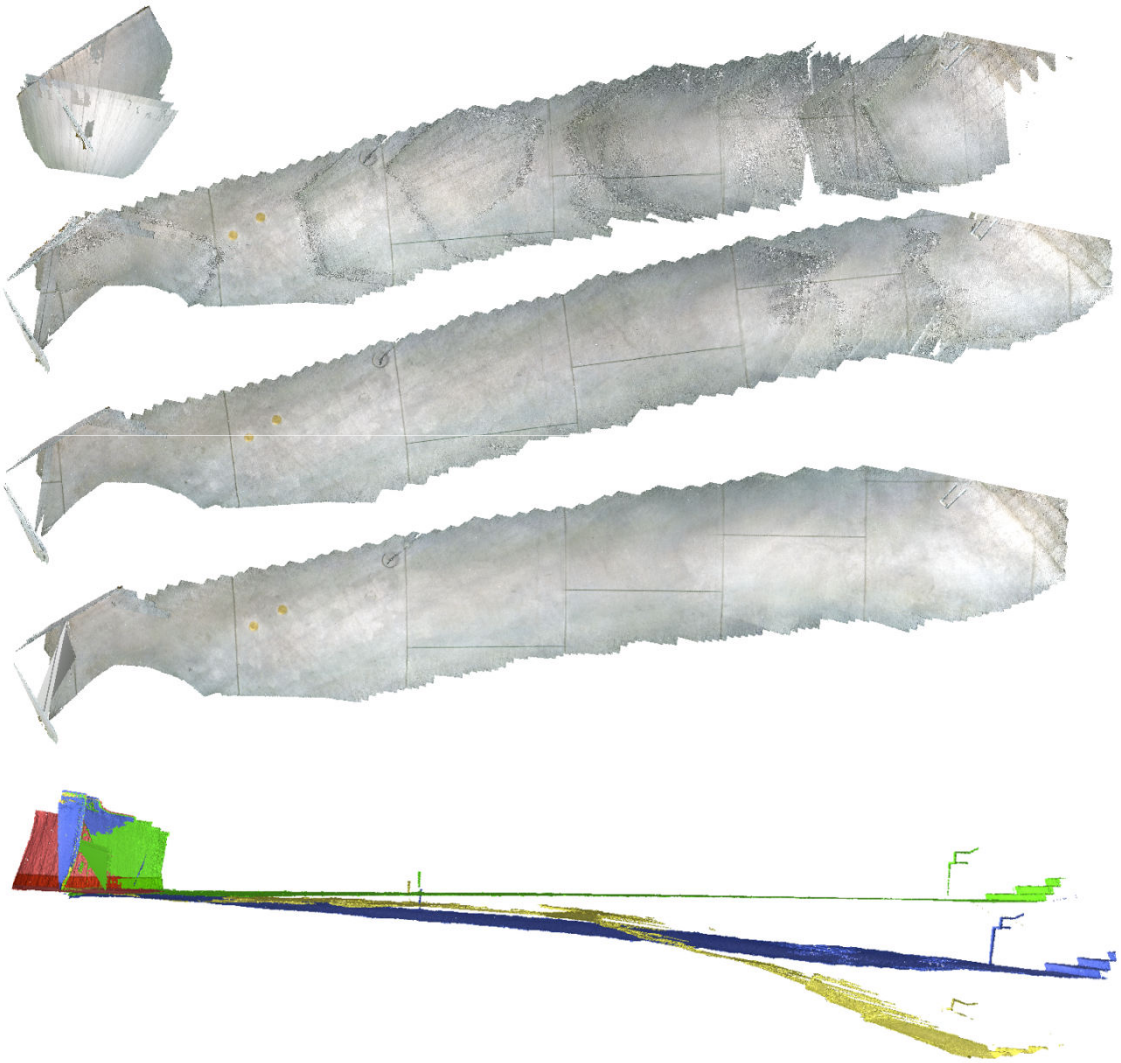
- "Real-Time Visual Odometry from Dense RGB-D Images" by F. Steinbruecker, J. Sturm, D. Cremers
  Workshop on Live Dense Reconstruction with Moving Cameras at the Intl. Conf. on Computer Vision (ICCV), 2011

Kintinuous 1.5 (Whelan et al., ICRA ‘13)

- FOVIS
- ICP
- RGB-D
- ICP+RGB-D

Kintinuous 1.5 (Whelan et al., ICRA ‘13)

- ICP
- RGB-D
- FOVIS
- ICP+RGB-D



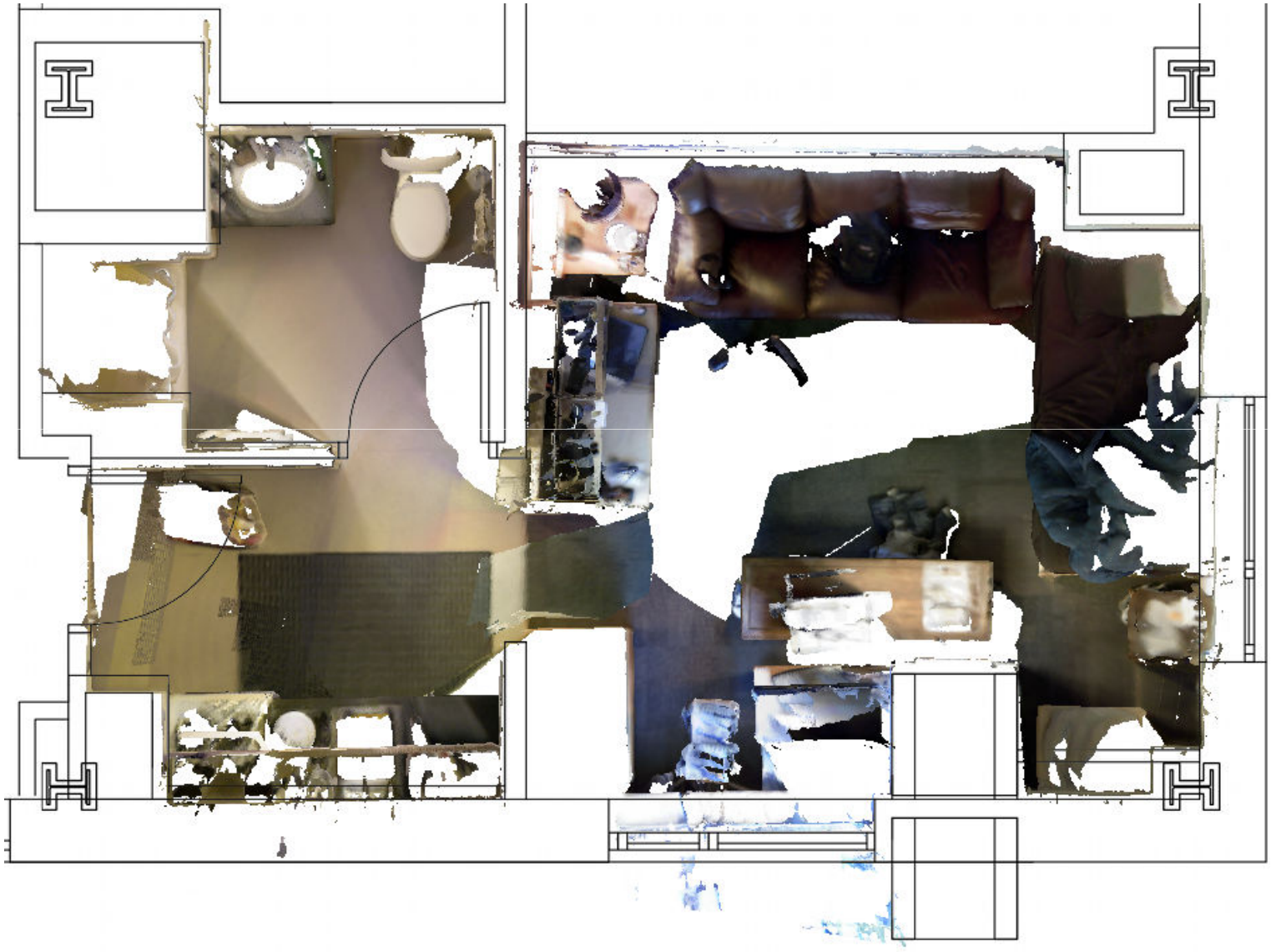
Dr. Thomas Whelan, Dyson Robotics Lab, Imperial College London & Dr. Jürgen Sturm, Computer Vision Group, TUM

Kintinuous 1.5 (Whelan et al., ICRA '13)

RELATIVE ROOT MEAN SQUARE ERROR (RMSE) OF DRIFT IN METERS PER SECOND FOR GROUND TRUTH DATA.

Dataset	fr1/desk	fr2/desk	fr1/room	fr2/large_no_loop
ICP	0.1198m	0.0453m	0.1187m	1.0130m
RGB-D	0.0558m	0.0321m	0.1028m	0.2254m
FOVIS	0.0604m	**0.0136m**	*0.0642m*	**0.0960m**
FOVIS/ICP	0.1108m	0.0459m	0.0776m	0.2329m
FOVIS/RGB-D	0.0555m	0.0342m	0.0708m	*0.1412m*
ICP+RGB-D	**0.0393m**	0.0208m	**0.0622m**	0.1795m
FOVIS/ICP+RGB-D	*0.0395m*	*0.0207m*	0.0755m	0.1652m

Kintinuous 1.5 (Whelan et al., ICRA ‘13)

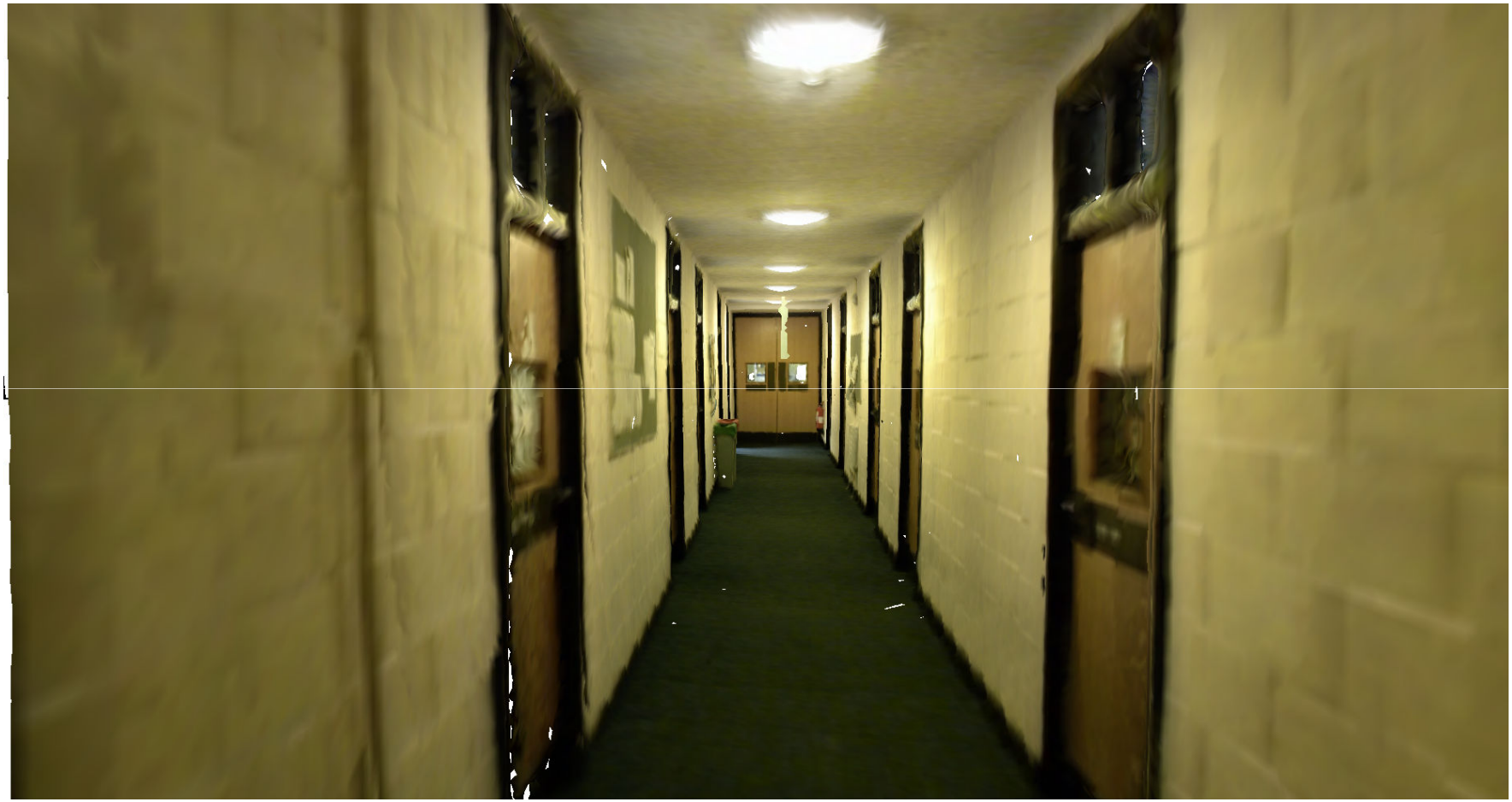


Kintinuous 1.5 (Whelan et al., ICRA '13)

- Handheld laser (tolerance ~1.5mm)
  - Width within 1cm
  - Laser length: 36.04m, Kintinuous length: 35.44m

- Ground truth for this is difficult?
  - We can evaluate trajectories quite well
  - How do we evaluate the actual maps?
    - Floor plans?
    - LIDAR?

Kintinuous 1.5 (Whelan et al., ICRA ‘13)



Corridor Reconstruction

Kintinuous 1.5 (Whelan et al., ICRA '13)

- Conclusions
  - ICP+RGB-D is superior because it registers to the model
  - FOVIS/RGB-D alone do not
    - Keyframe / last frame
  - Drift is minimal, but suffers from the usual problems
    - Needs far features to avoid rotational bias
  - Seems quite robust to auto exposure
    - Can be disabled in OpenNI 2.0

Kintinuous 2.0

- Addresses issue of loop closure
  - Online operation
  - Large scale

- "Deformation-based Loop Closure for Large Scale Dense RGB-D SLAM" by T. Whelan, M. Kaess, J.J. Leonard, and J.B. McDonald at IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems, IROS, (Tokyo, Japan), November 2013

Volumetric Fusion

- Brought to the forefront in real-time tracking and mapping by Newcombe et al. with KinectFusion, many attractive characteristics including;
  - Runs at frame rate (30Hz)
  - Very high quality reconstruction
  - Maintains occlusion relationships

Volumetric Fusion

- A number of initial restrictions;
  - Desk space (or resolution vs. area size)
  - Requires good geometric features in the scene
  - No explicit correction for drift (only implicit)

- Reliance on geometric features has been tackled (Whelan et al. 2013, Henry et al. 2013, Peasley et al. 2013).

- Overcoming the space restriction greatly amplifies the need for a method to correct for drift and ensure global consistency.

Volumetric Fusion: Space restriction

- A number of solutions presented to date
  - Streaming TSDF tiles (Yan Ma, Masters 2012)
  - Cyclic buffer with raycast (Whelan et al. 2012, kinfu_large_scale in PCL)
  - Double buffer with remapping (Roth and Vona 2012)
  - Octree TSDF (Zeng et al. 2012)
- Most recent;
  - Point-based fusion (Keller et al. 2013)
  - Hybrid hierarchical data structure (Chen et al. 2013)

Volumetric Fusion: Space restriction

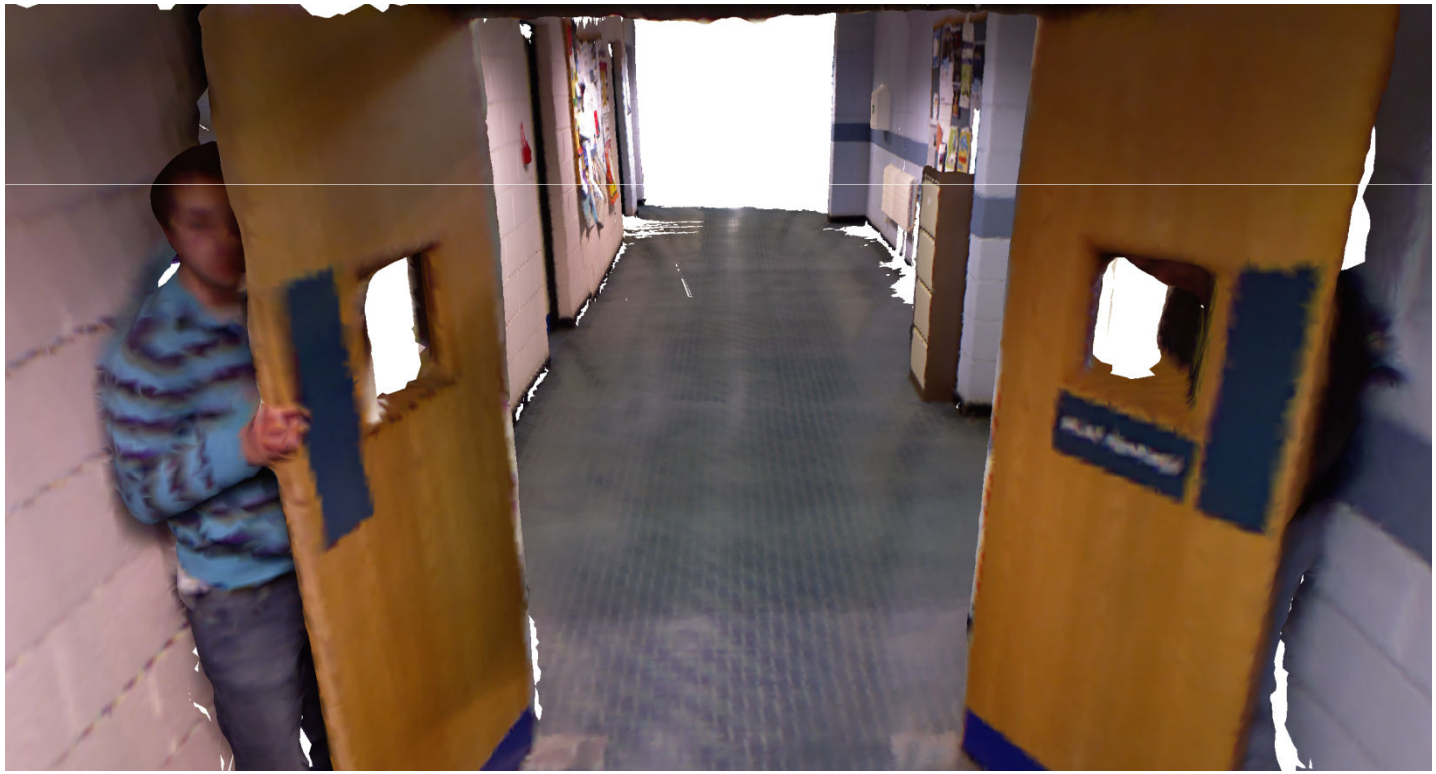
- None of these have a means for correcting for drift or ensuring global consistency in real-time.

- Offline multi-pass approach of Zhou and Kolton (Dense Scene Reconstruction with Points of Interest in ACM Transactions on Graphics, 2013)
  - Process data many times for best reconstruction.
  - Computation time measured in hours.

- Recent work by Henry et al. (Patch Volumes: Segmentation-based Consistent Mapping with RGB-D Cameras at Third Joint 3DIM/3DPVT Conference (3DV), 2013)
  - Almost real-time, locally disconnected surfaces

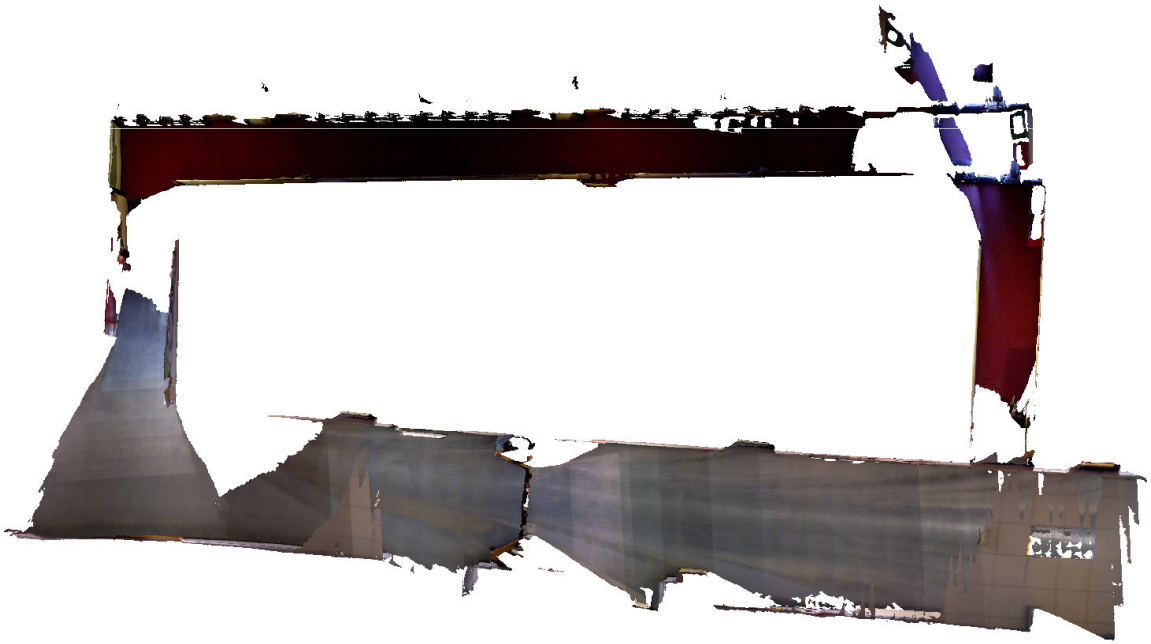
What's the problem?

- Surfaces are locally consistent but drift slowly over time/space



What's the problem?

- Surfaces are locally consistent but drift slowly over time/space



Correction

- Can we even correct it? Easy to check:
  - Build an every-frame pose graph
  - Detect visual loop closures (e.g. DBoW)
  - Optimise the camera pose for each frame (iSAM)
  - Rerun the data using the optimised trajectory

Correction

- It works! But how can we do this online?

Correction

- The surface must be non-rigidly deformed to smoothly correct the reconstruction.
  - An explicit voxel or volumetric representation makes this difficult to perform quickly.
  - Kintinuous benefits from using a vertex-based surface representation.

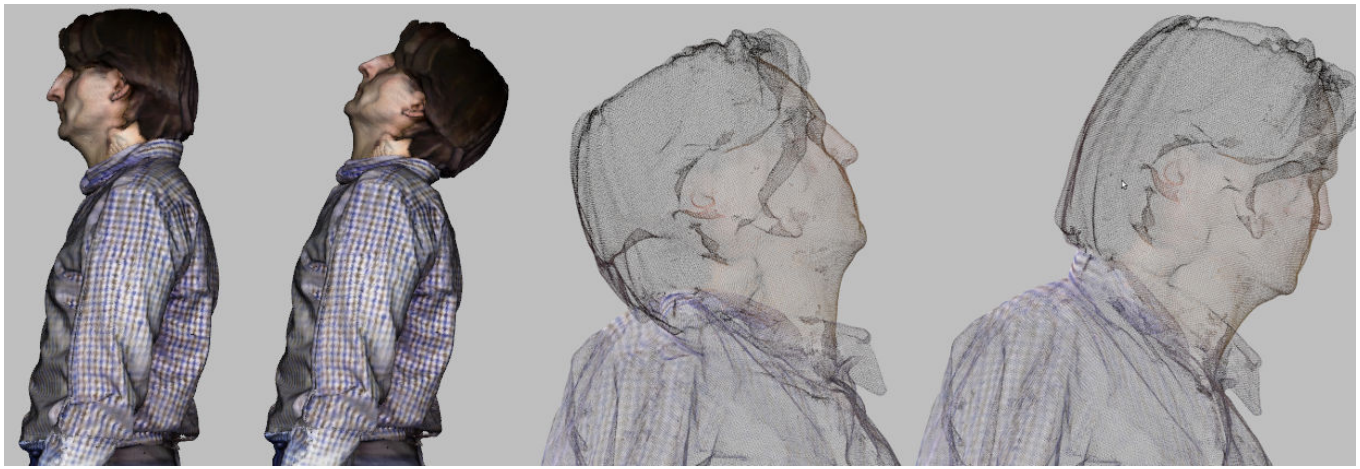
Deformation

- Countless literature on non-rigid surface manipulation in the graphics community.
- Something which is fast, easy to parameterise and doesn't require "pure" surfaces is ideal.
  - Scans, even with fusion, are "rough around the edges", i.e. non-oriented, not watertight etc...
  - A space deformation is perfect, which directly transforms anything lying in R^3

- We chose "Embedded Deformation for Shape Manipulation" by Robert W. Sumner, Johannes Schmid, Mark Pauly in SIGGRAPH 2007

Embedded Deformation

- Meant for real-time manipulation of 3D models
  - Can be a triangular mesh, polygon soup or vertices
- Parameterised by user specified control points
  - Essentially tries to pull points in R^3 to user specified points, smoothly deforming space to accomplish this.
- Cannot be directly applied to SLAM (more to follow on this...)



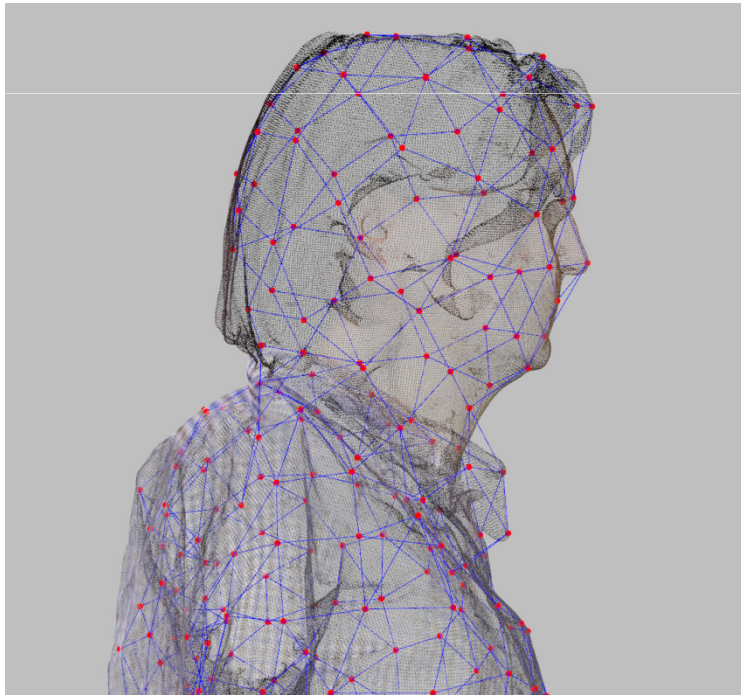
Embedded Deformation

- Constructs a graph of deformation nodes
  - Each node contains an affine transformation in the form of a 3x3 matrix and 3x1 translation vector
  - Each node has a set of neighbours
- Each vertex is affected by the transformation stored at the k-nearest nodes to that vertex
- The parameters of each node's transformations are optimised over (in a non-linear least squares sense)

- As highlighted by Sumner at al. this is framed as a sparse problem, which is crucial to achieving high performance.

Embedded Deformation

- Deformation graph traditionally constructed in a nearest neighbour manner (e.g. using a kd-tree)
- Individual vertex transformations are computed similarly

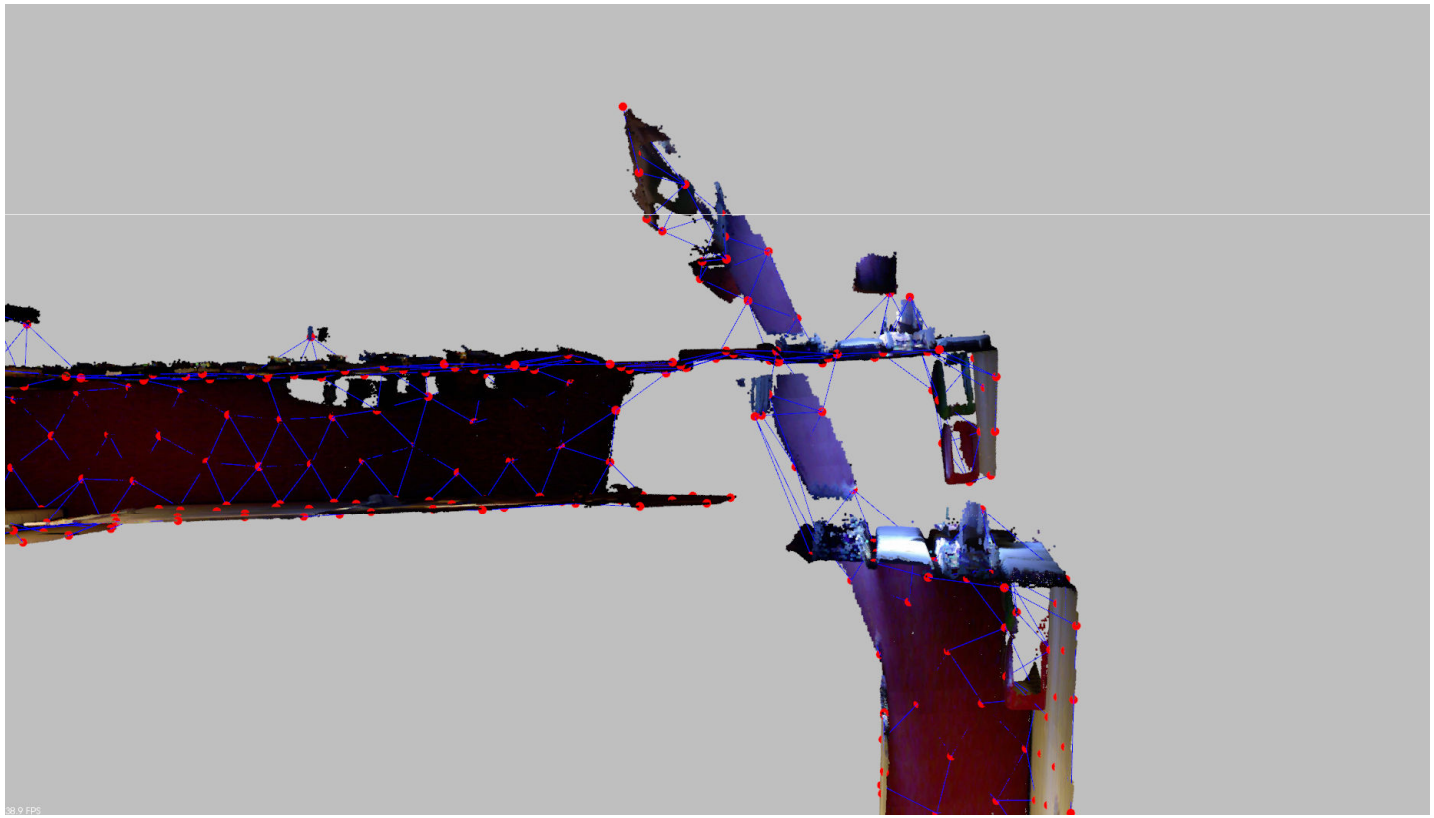


Embedded Deformation in SLAM

- A nearest neighbour sampling strategy for deformation graph construction (and vertex transformation) will fail catastrophically on certain maps.
  - Completely unrelated areas of the map may be connected together in the graph, causing instant divergence.
- There are no user specified control points in SLAM
  - We need something to drive the deformation, something that pulls the space (and subsequently the model) in the right direction.

Embedded Deformation in SLAM: Sampling

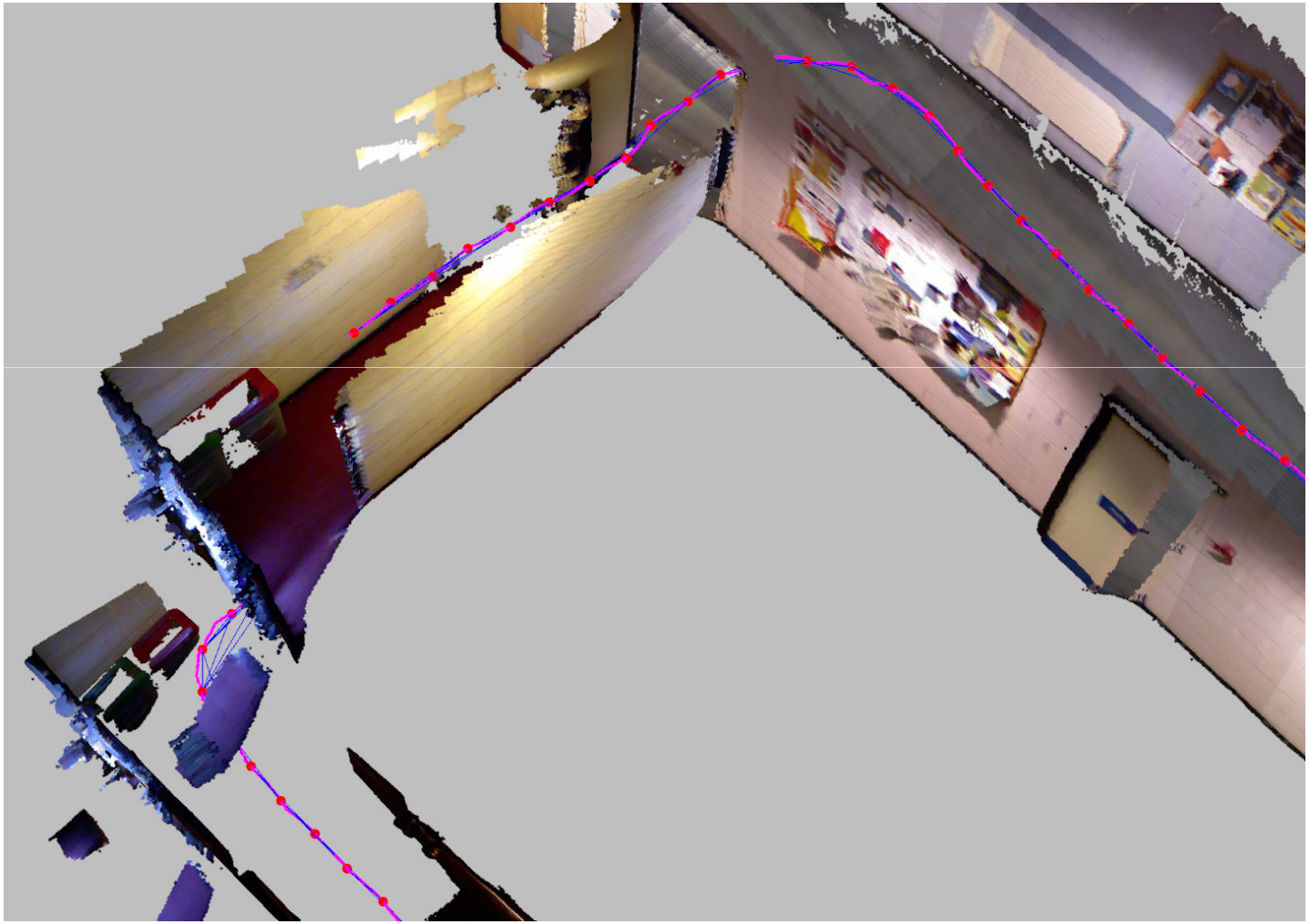
- Nearest neighbour graph sampling can connect unrelated areas of the map



Embedded Deformation in SLAM: Sampling

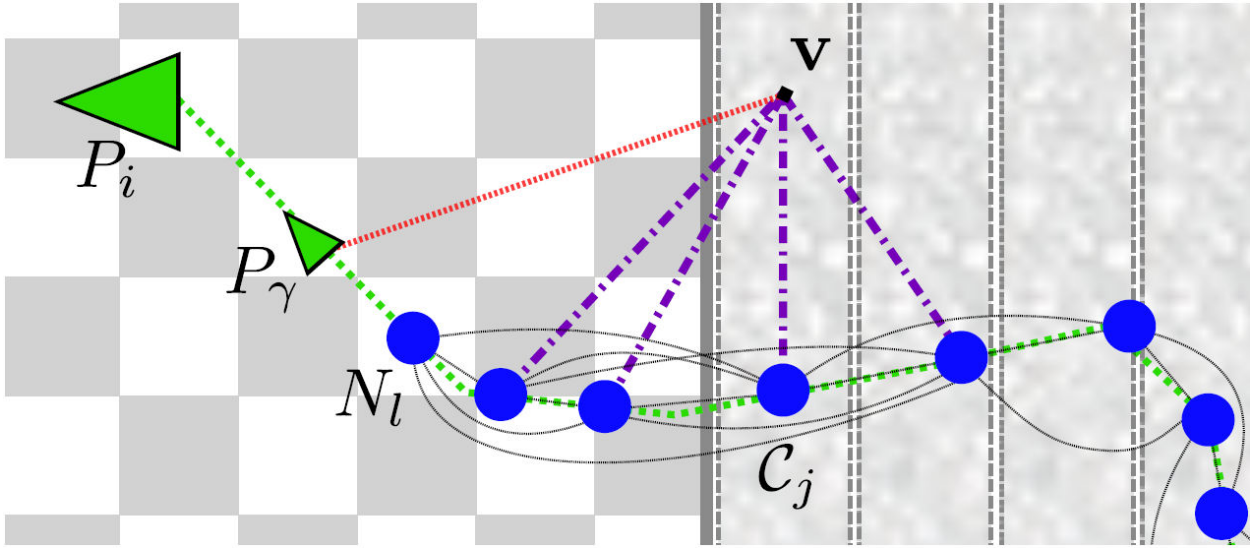
- Could try cutting edges or add additional sampling rules
  - Arbitrary points in the scene are not necessarily the best points to add nodes (as discovered by Chen et al. with KinÊtre)
- Instead, we use the pose graph as the deformation graph
  - Guarantees unrelated areas of the map won't be connected
  - Quick and easy to compute (basically free) and can be constructed incrementally as the pose graph grows.

Embedded Deformation in SLAM: Sampling



Embedded Deformation in SLAM: Sampling

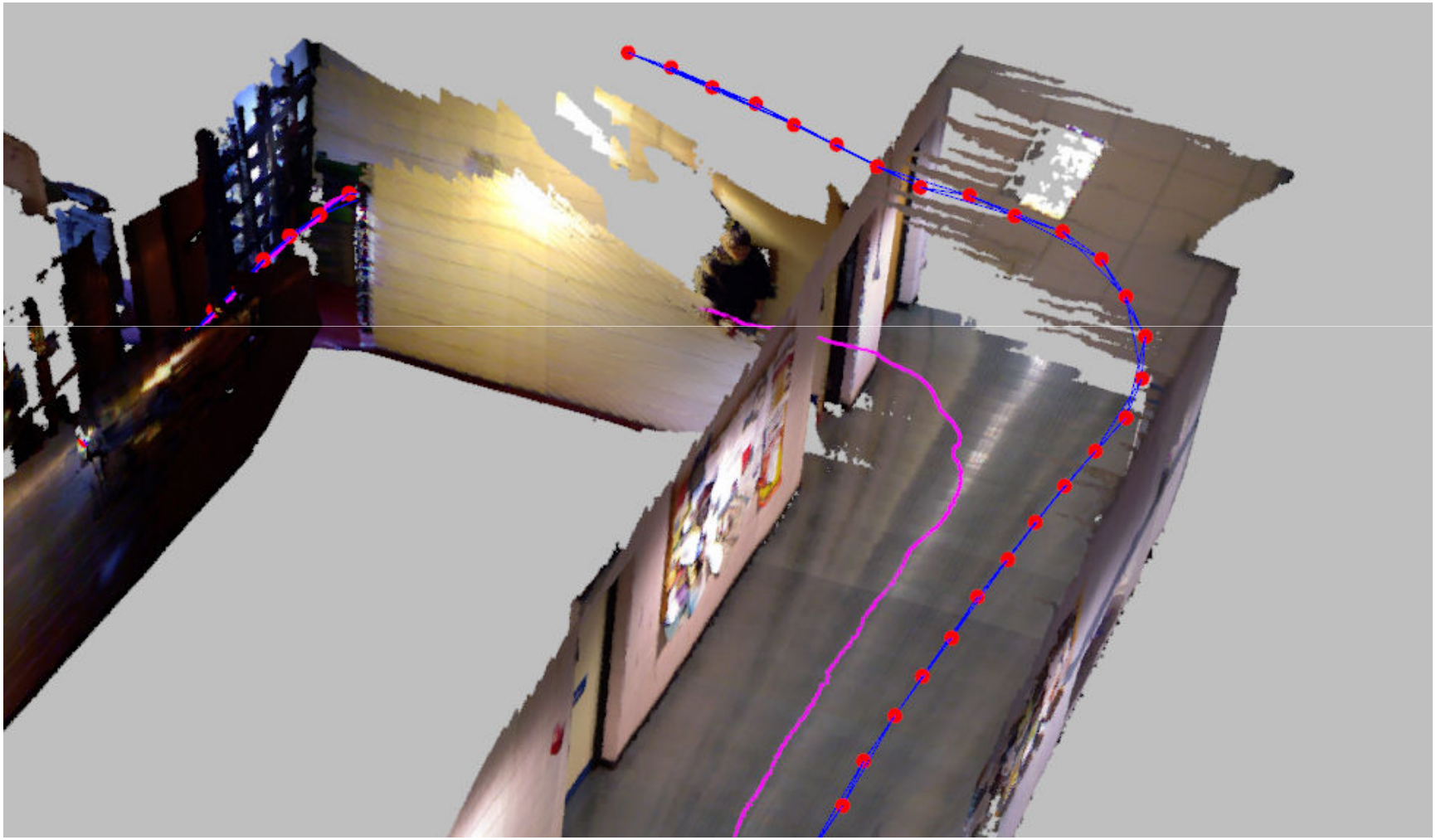
- What about vertices?
  - If we just take the k-nearest nodes to each vertex, we may select nodes which cam from much older poses in the graph.
  - We can restrict the set of nearest nodes to each vertex according to the pose which created a given vertex
    - Kintinuous incrementally produces “cloud slices” from the TSDF, each with an associated camera pose.



Embedded Deformation in SLAM: Control

- How do we constrain the deformation without user specified points?
- The camera pose graph can do just that
  - We use the unoptimised camera poses as the initial control point positions and use the optimised camera poses as the desired control point positions.
  - Space is effectively smoothly deformed and manipulated around the pose graph.

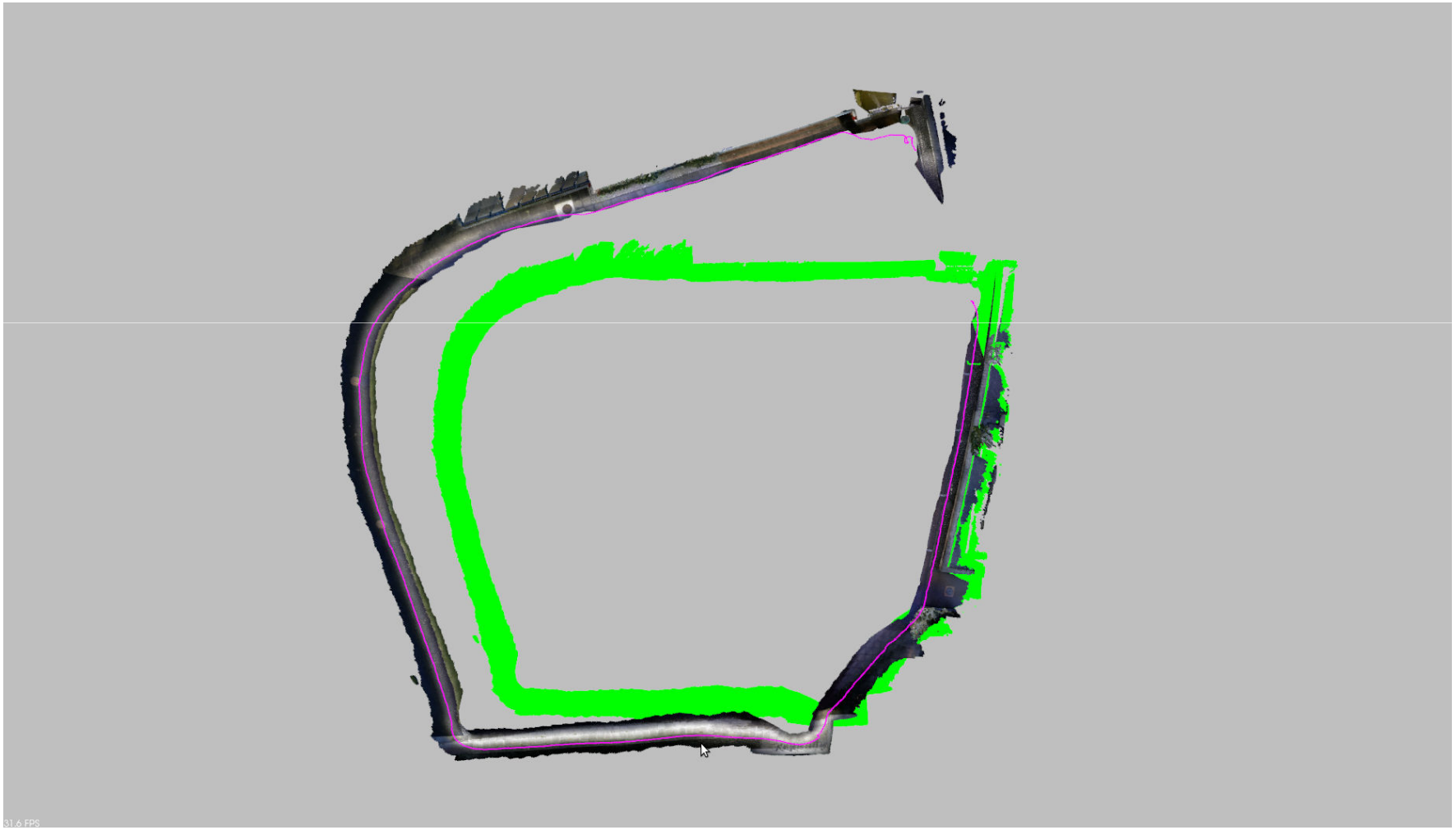
Embedded Deformation in SLAM: Control



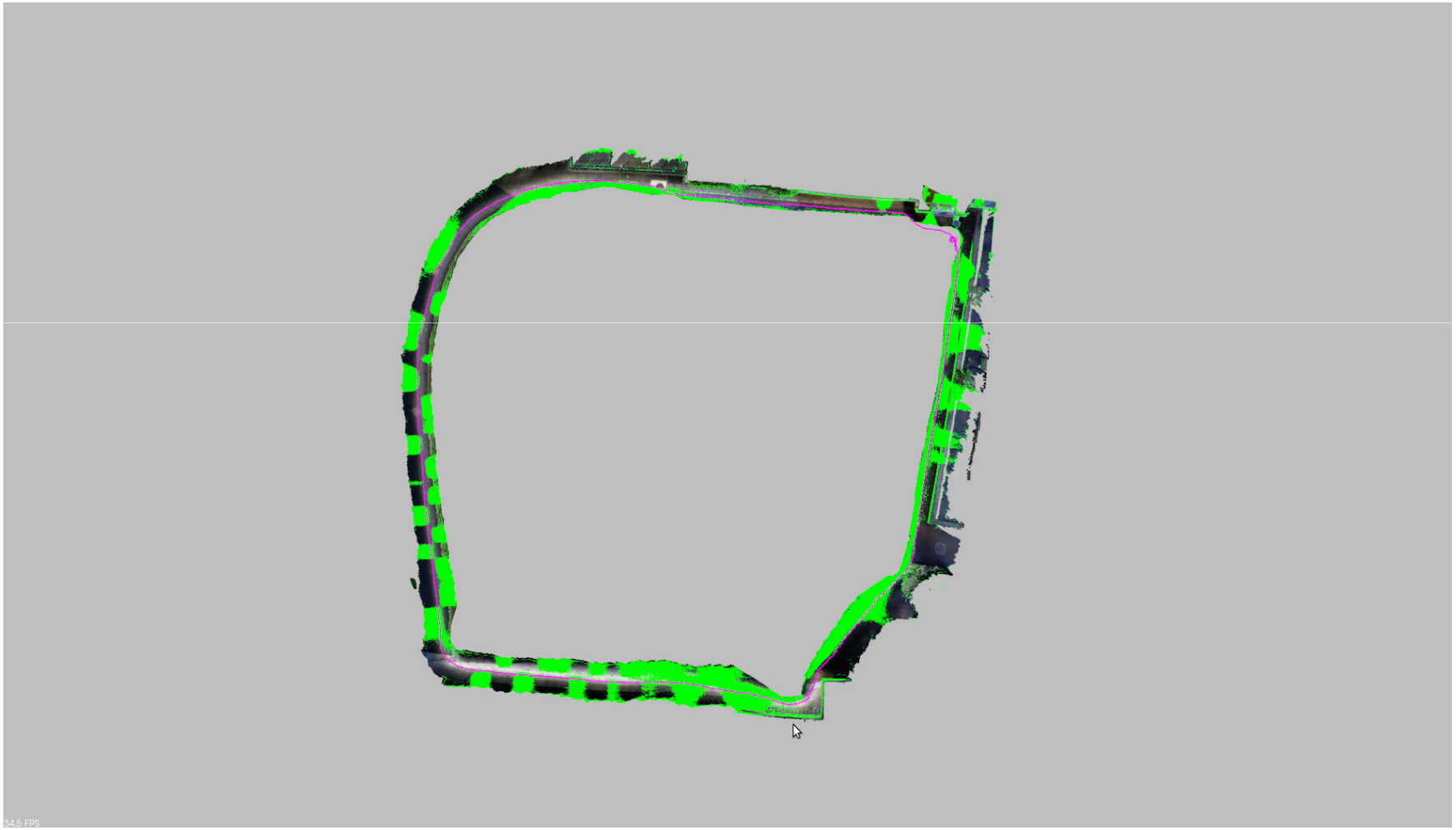
Results: 1-pass vs. 2-pass

- How do the resultant maps compare?
  - The pose graphs are identical, however the reconstructions differ slightly
- Without surface ground truth (as opposed to trajectory ground truth) it's difficult to evaluate which is more accurate
  - Large scale, accurate surface ground truth appears to be an open problem?

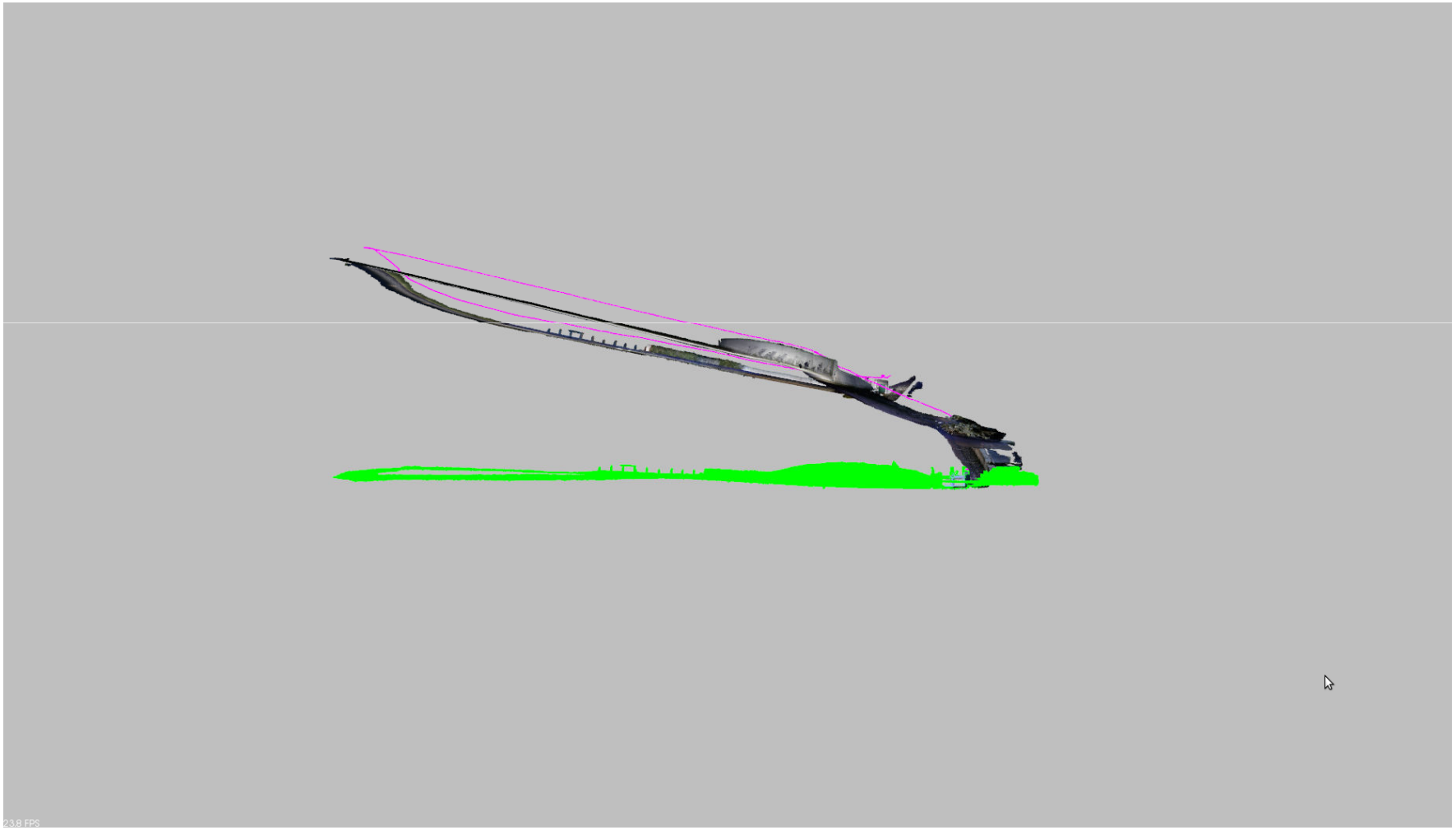
Results: 1-pass vs. 2-pass



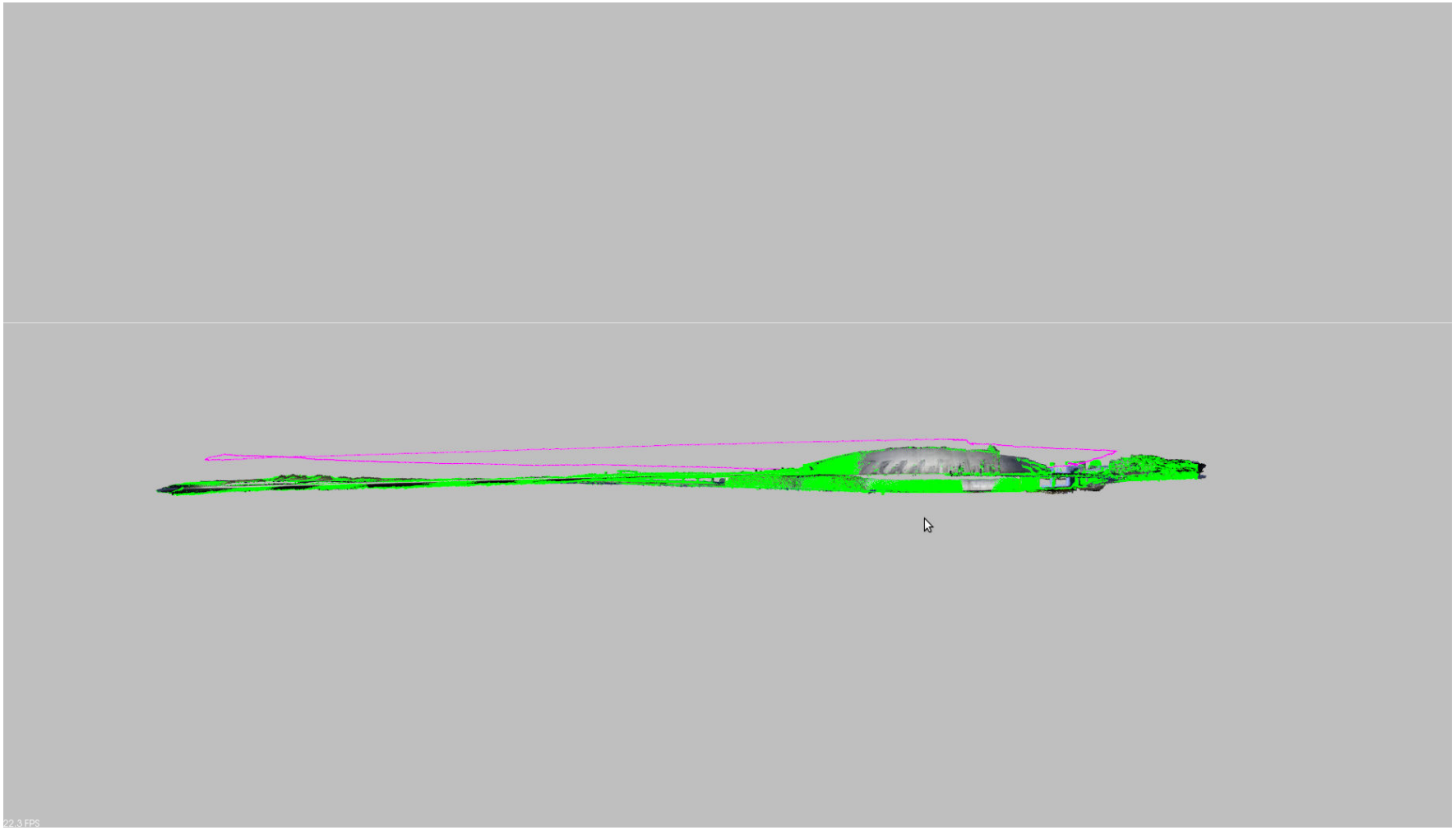
Results: 1-pass vs. 2-pass



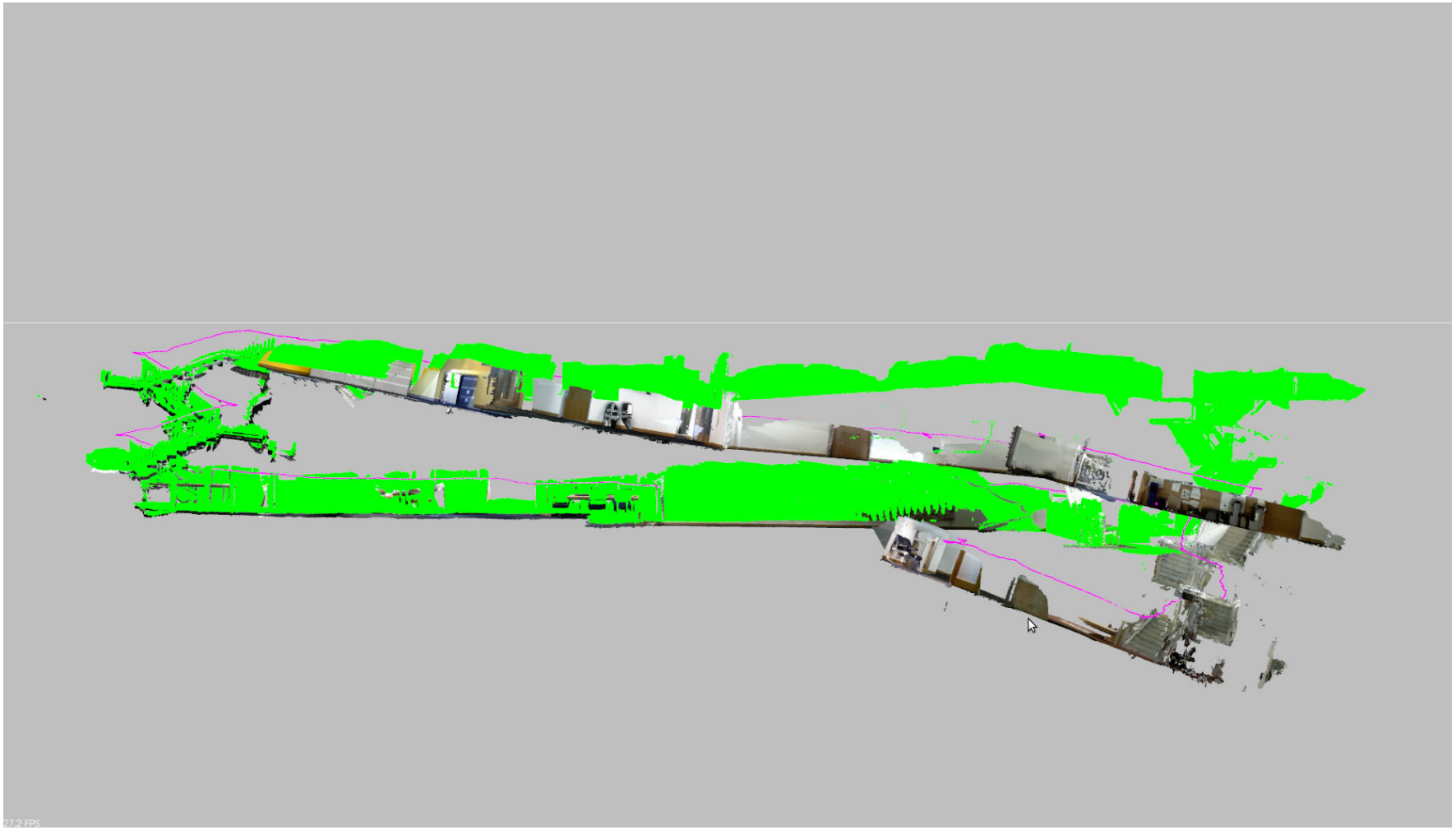
Results: 1-pass vs. 2-pass



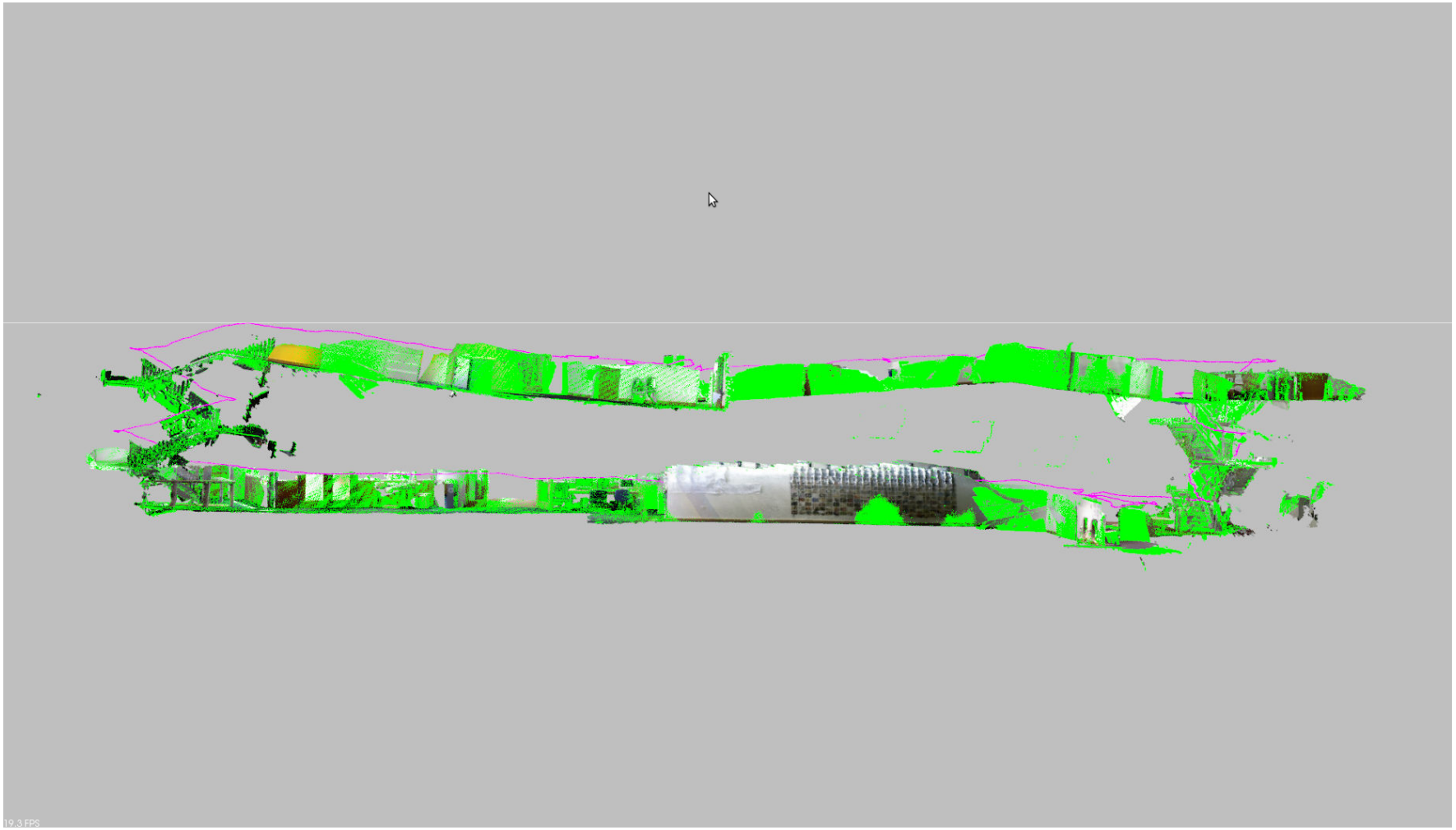
Results: 1-pass vs. 2-pass



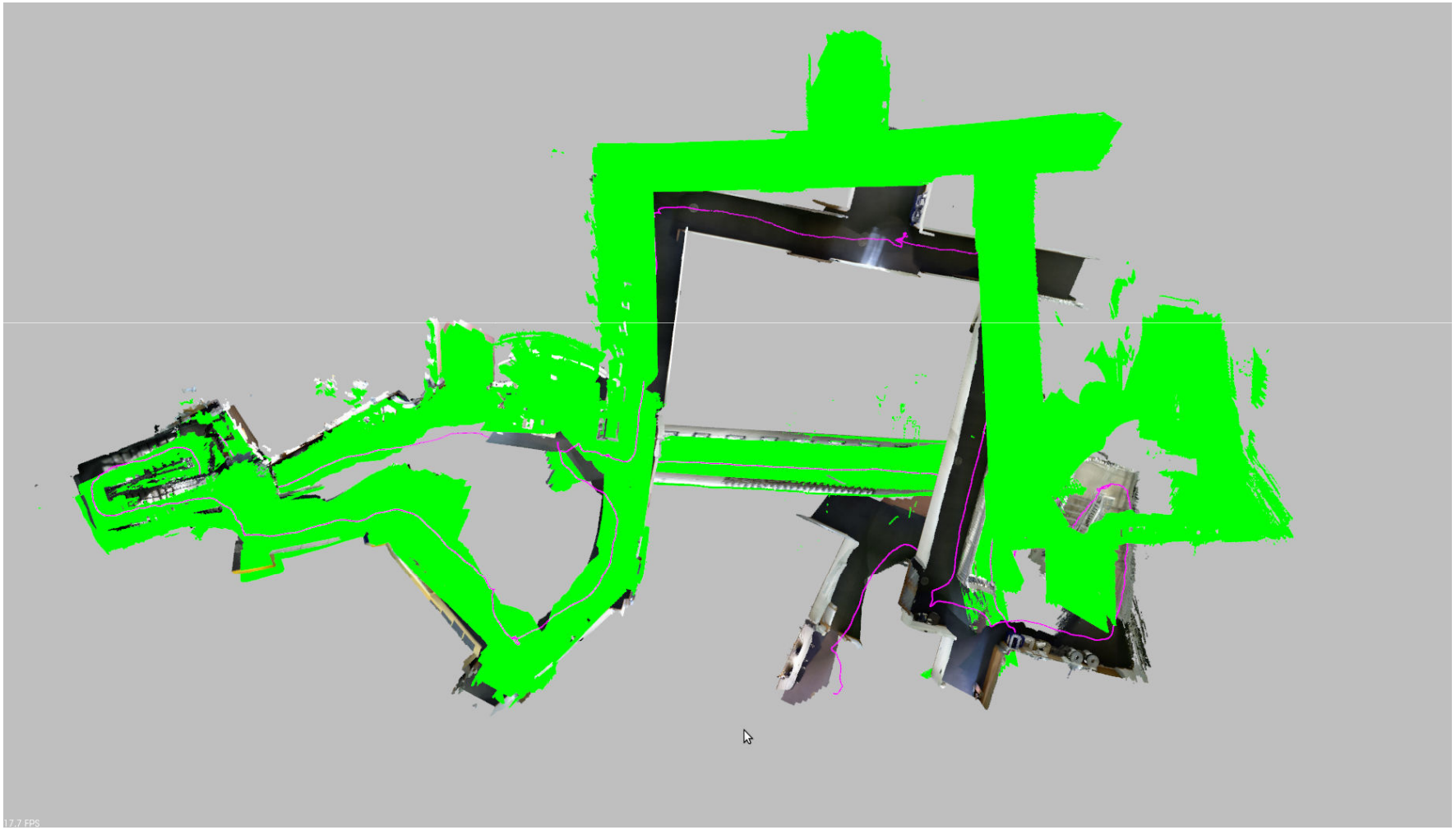
Results: 1-pass vs. 2-pass



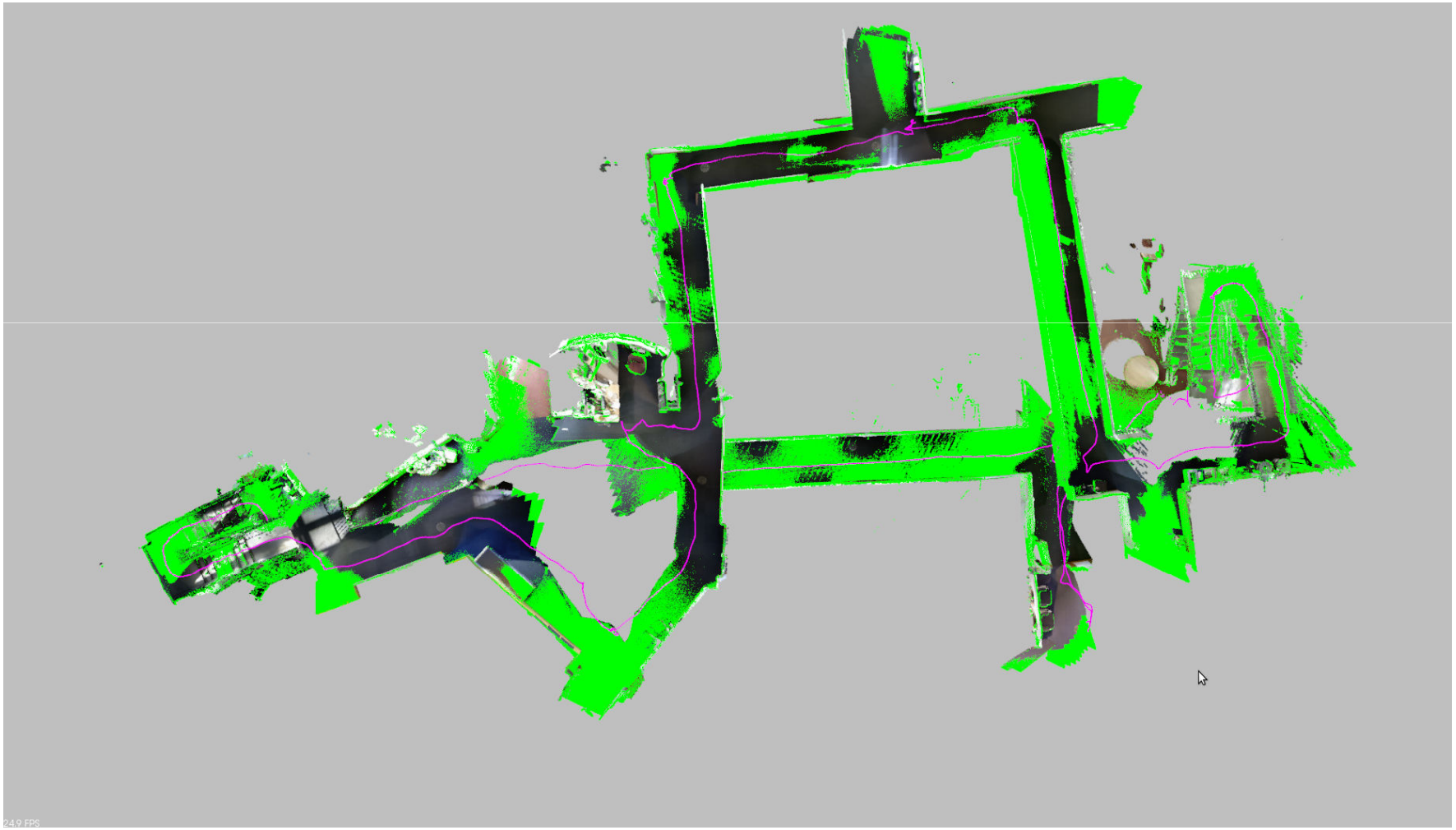
Results: 1-pass vs. 2-pass



Results: 1-pass vs. 2-pass



Results: 1-pass vs. 2-pass

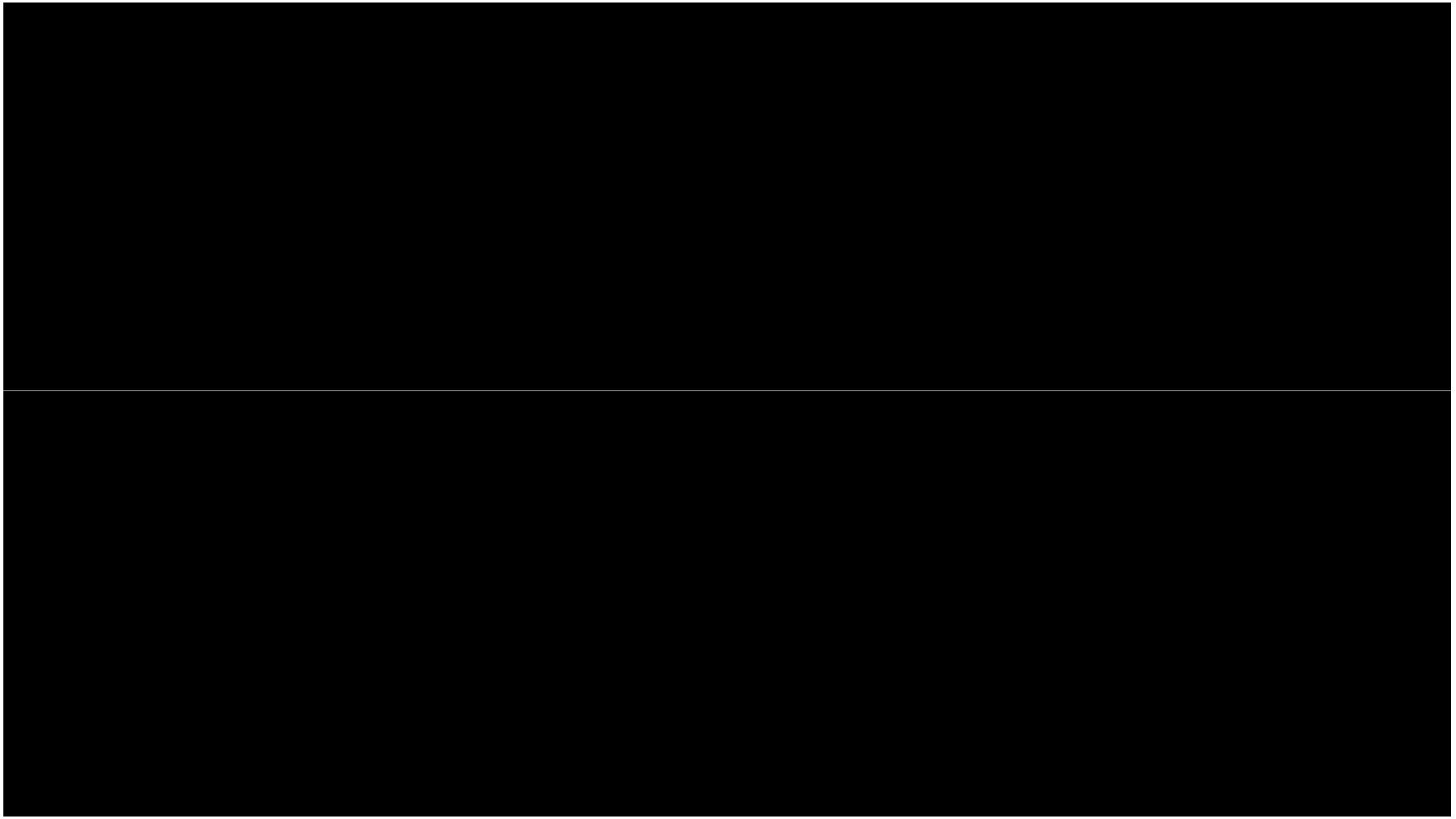


Results: Timing

- We measure computational performance in terms of latency

Quantities	Datasets			
	Indoors	Outdoors	Two floors	In/outdoors
DBoW images	311	1181	1603	2625
Poses	313	1182	1605	2627
Nodes	89	168	176	338
Vertices	1,598,253	3,190,669	4,296,449	6,909,051
Process	Timings (ms)			
Frontend	578	528	663	973
iSAM	50	239	392	986
Deformation	115	377	461	974
Total latency	743	1144	1516	2933

Results: Video



Alternative Sensors

- Camboard Nano from PMD
  - 160x120 ToF at 90FPS
  - 5cm – 50cm range
  - 0.52mm resolution
  - Overall very noisy
  - Tracking difficult
  - Better future sensors?
  - Smart phone scanning?
- Dense depth
  - Looks promising

Dr. Thomas Whelan, Dyson Robotics Lab, Imperial College London & Dr. Jürgen Sturm, Computer Vision Group, TUM

Applications of the data

- Dense maps are great but
  - Using them to plan is expensive/slow
  - Many parts are over represented

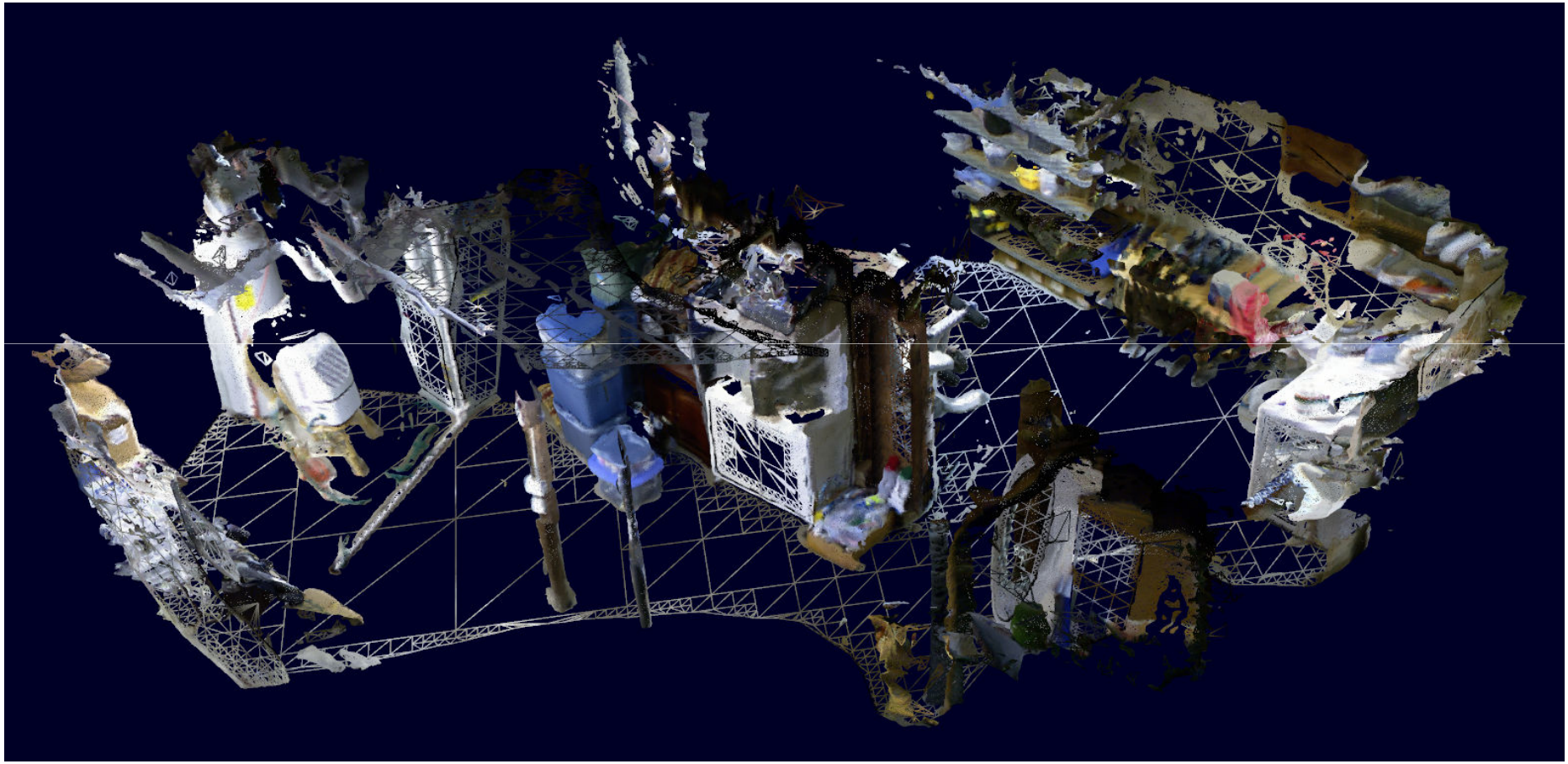
- Let's start with planes
  - Easy to detect given fidelity of Kintinuous output

Planar Simplification

- Collaboration with TU/e
  - Lingni Ma, Egor Bondarev, Peter H. N. de With
- Fully automatic
- Batch
  - Also works incrementally online
    - Under review

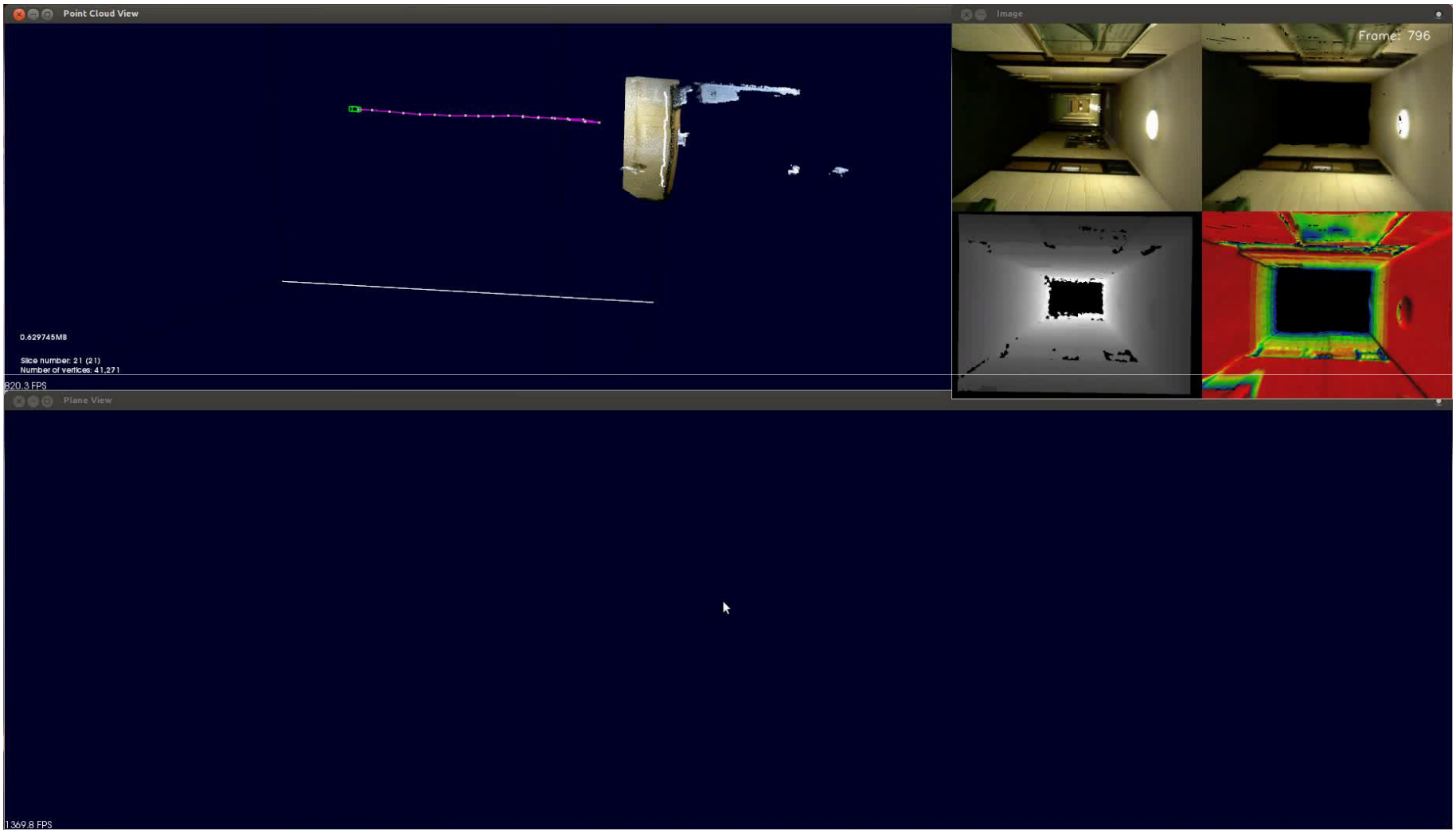
- "Planar Simplification and Texturing of Dense Point Cloud Maps" by L. Ma, T. Whelan, E. Bondarev, P. H. N. de With, and J.B. McDonald. In European Conference on Mobile Robotics, ECMR, (Barcelona, Spain), September 2013

Planar Simplification

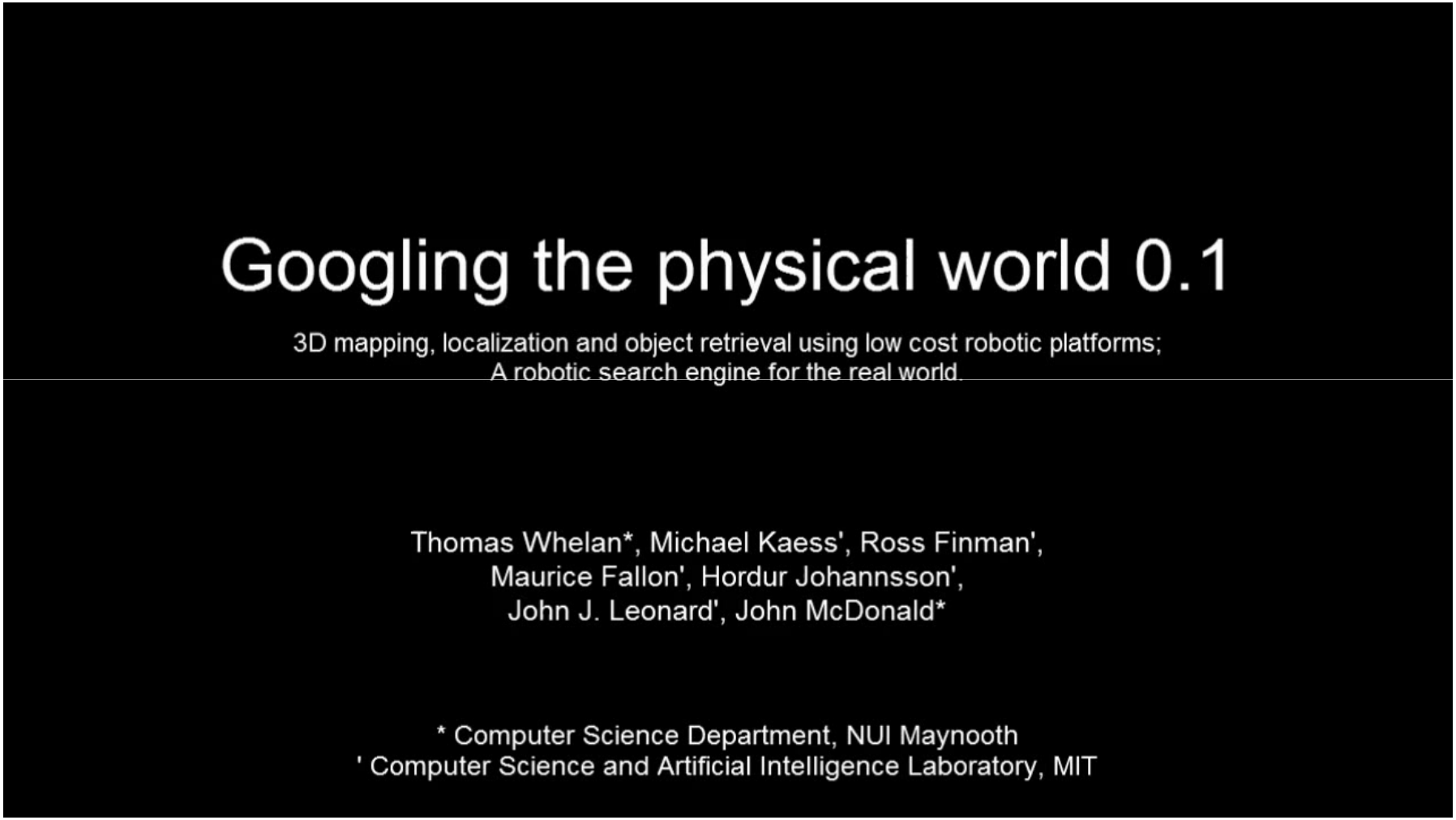


Combined planar and dense triangulation

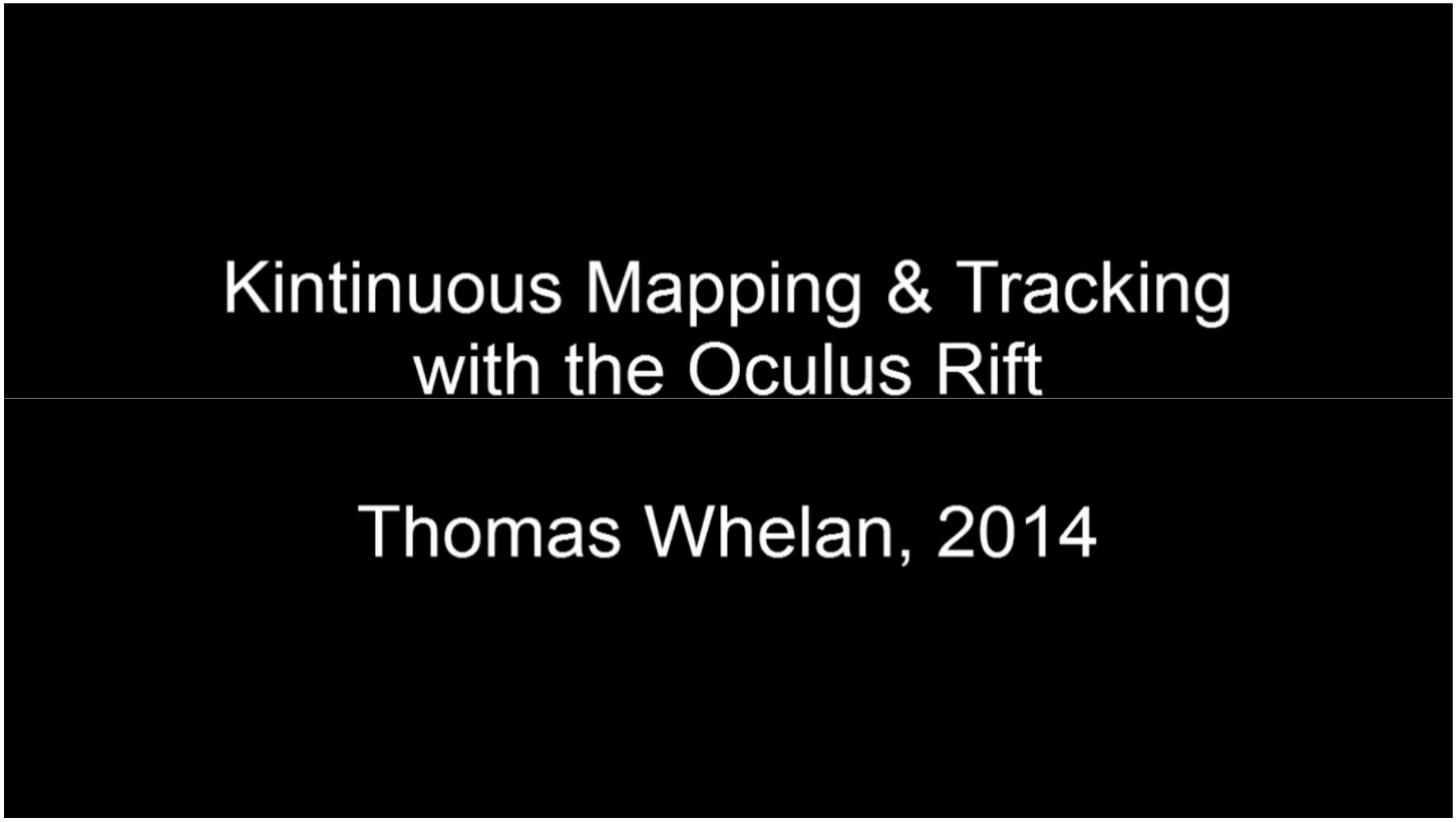
Planar Simplification



Closed-loop Robotics



Virtual Physical Reality



Conclusion

- Dense methods bear a large potential
  - Dense camera tracking
  - Dense 3D reconstruction
  - Dense SLAM
- Some open source implementations available
  - TUM, Stanford, PCL
- Many directions for future research
- RGB-D Cameras a real enabler here

Outline: Afternoon Session

- Get bootable USB stick + RGB-D camera
- Play around
  - Inspect color, infrared, depth images
  - Inspect 3D point clouds
  - Write your own program (given example code)
    - Simple operations on RGB-D data
    - A full SLAM system

Thank You

- Any questions?