Humanoid Robotics

3D World Representations

Maren Bennewitz, Rohit Menon
Goal of This Chapter

- Overview of different 2.5/3D world representations with their strengths and weaknesses
- Understand which representation is useful for which application
- Know how to calculate transformations between point clouds
Robots in 3D Environments

source: Honda
Motivation

- Robots live in the 3D world
- Collision avoidance, motion planning, and localization require accurate 3D world models
- Given: 3D point cloud data from known robot and sensor poses
- Question: How to represent the 3D structure of the environment?
Laser Scanning Principle

Range scanners measure the distance to the closest obstacle

Image courtesy: Sick
Depth Image of RGB-D Camera

Image courtesy: Intel

Intel Realsense D435i

90 mm x 25 mm x 25 mm
Example: Data Acquisition
Popular Representations of the 3D World

- Point clouds
- Voxel grids
- Height maps
- Surface maps
- Meshes
- Distance fields
- ...

All models are wrong, but some are useful
- George Box
Point Clouds

- Set of 3D data points in world frame
- Obtained, e.g., by a laser scanner or depth camera
Point Clouds

Pros
- No discretization of data
- Mapped area not limited

Cons
- Unbounded memory usage
- No constant time access for location queries
- No distinction between free or unknown space
Point Clouds and Efficient Location Queries

- Naïve implementation (list, array) has a linear complexity for location queries
- More effective solutions through \textbf{kd-trees}
- \textbf{kd-trees} operate in k-dimensions
- Space-partitioning data structure for organizing k-dimensional points
- Search/insert/delete in \textbf{logarithmic} time on average

[see exercise sheet]
Example: kd-Tree (2-dim.)

Binary space partitioning

3D Voxel Grids
3D Voxel Grids

Pros
- Volumetric representation
- Constant access time
- Probabilistic update possible

Cons
- Memory requirement: Complete grid is allocated in memory
- Extent of the map has to be known/guessed
- Discretization errors
2.5D Maps: Height Maps

Average over all points that fall into a 2D cell and consider this as the height value
2.5D Maps: Height Maps

Pros
- Memory efficient (2D)
- Constant time access

Cons
- No vertical objects
- Only one level is represented
Example: Problem of Height Maps
Multi-Level Surface Maps (MLS)
MLS Map Representation

Each 2D cell stores a set of “patches” consisting of:

- The height mean $\mu$
- The height variance $\sigma$
- The depth value $d$

Note:

- A patch can have no depth (flat objects, e.g., floor)
- A cell can have one or many patches (vertical gap, e.g., bridges)
From Point Clouds to MLS Maps

- Determine the 2D cell for each 3D point
- Compute vertical intervals based on a threshold

- Determine for the vertical objects:
  - The **height** and its **variance**
  - The **depth** as the difference between the highest and the lowest measurement
Example: MLS Maps

Point cloud

Multi-level surface map
MLS Maps

Pros
- Can represent multiple surfaces per 2D cell

Cons
- No volumetric representation but a discretization in the vertical dimension
- Several tasks in a MLS map are not straightforward to realize
Octree-Based Representation

- Tree-based data structure
- Recursive subdivision of the space into octants
- Volumes allocated as needed
- “Smart” 3D grid
Octrees
Octrees

Pros
- Full 3D model
- Inherently multi-resolution
- Memory-efficient, volumes only allocated as needed
- Probabilistic update possible

Cons
- Efficient implementation can be tricky
  (memory allocation, update, map files, ...)

[see exercise]
Multi-Resolution Queries

\[ P(m_i) = \max_{j=1\ldots8} P(m_{i,j}) \text{ with } m_{i,j} \in \text{children}(m_i) \]
OctoMap Framework

- Based on octrees
- Probabilistic, volumetric representation of occupancy including unknown
- Supports multi-resolution map queries
- Memory efficient
- Generates compact map files (maximum likelihood map as bit stream)
- Open source implementation as C++ library available at http://octomap.github.io/
Ray Casting for Map Updates

- Ray casting from sensor origin to end point in the map along the beam
- Mark last voxel as occupied, all other voxels on ray as free
- Measurements are integrated probabilistically given the robot’s pose (recursive binary Bayes’ filter)
Probabilistic Map Update

- Occupancy probability modeled as recursive binary Bayes’ filter

\[
p(m_i \mid z_{1:t}, x_{1:t}) = \left[ 1 + \frac{1 - p(m_i \mid z_t, x_t)}{p(m_i \mid z_t, x_t)} \frac{1 - p(m_i \mid z_{1:t-1}, x_{1:t-1})}{p(m_i \mid z_{1:t-1}, x_{1:t-1})} \frac{p(m_i)}{1 - p(m_i)} \right]^{-1}
\]

- Efficient update using log-odds notation

\[
l(m_i \mid z_{1:t}, x_{1:t}) = l(m_i \mid z_t, x_t) + l(m_i \mid z_{1:t-1}, x_{1:t-1}) - l(m_i)
\]

[Inverse sensor model \quad Recursive term \quad Prior]

[Lecture on Cognitive Robotics]
Video: Large Outdoor Area

Freiburg computer science campus
(292 x 167 x 28 m³, 20 cm resolution)

Octree in memory: 130 MB
3D Grid: 649 MB
Octree file: 2 MB (bit stream)
Online Mapping With Octomap
Signed Distance Function
Signed Distance Function (SDF)

Key idea:

- Instead of representing occupancy values, represent the distance of each cell to the nearest measured surface.
- The surface can be extracted afterwards at sub-voxel accuracy.
Signed Distance Function (SDF)

- Grid maps: explicit representation

- SDF: implicit representation
SDF Approach

1. Compute the signed distance values
2. Extract the surface using interpolation
3. The surface is located at the zero-crossing

| -1.3 | -0.3 | 0.7  | 1.7  |

negative = outside obj.  positive = inside obj.
Properties

- Noise cancels out over multiple measurements

- Zero-crossing can be extracted at sub-voxel accuracy
Voxel Grid to Store SDF in 3D

D(x) < 0
D(x) = 0
D(x) > 0

Negative signed distance (=outside)
Positive signed distance (=inside)

in general, there are several measurements for the voxels
Weighting Function for Multiple Measurements

- For each **voxel along the beam**, weigh the observation according to its **confidence**

  - Small weights ensure that values can be updated when new observations are available
SDF

For each voxel along the beam, store

- Distance to the next surface $D$
- Weight $W$
Truncated SDF

- Compute the SDF from a depth image
- Compute the **distance** of the voxels to the observed surface **along the beam**
- Update only a **small region around the endpoint** for efficiency (truncation)
Weighted Update

For each voxel, calculate the weighted average over all its measurements.

observations from known camera poses

several measurements of the voxel
**Weighted Average**

- For each voxel, store two values
  - Weighted sum of signed distances $D$
  - Sum of all weights $W$
- When new data arrive, update the values of each voxel according to

\[
D \leftarrow \frac{WD + w_t d_t}{W + w_t}
\]

\[
W \leftarrow W + w_t
\]

incremental computation of the weighted mean
SDF Example

A cross section through a 3D signed distance function of a real scene

Surface with cross-section

brightness encodes distance
Surface Rendering

1. Ray casting (GPU, fast)
   For each camera pixel, shoot a ray and search for the zero crossing

2. Polygonization (CPU, slow)
   Use the marching cubes algorithm to generate a triangle mesh
Ray Casting

- For each pixel, shoot a ray and search for the first zero crossing in the SDF
- Value in the SDF can be used to skip along the ray when far from surface
Mesh Extraction Using Marching Cubes

- Process the whole grid
- Find zero-crossings in the signed distance function by interpolation

2D

3D
Marching Squares (2D)

- Evaluate each cell separately
- Check which vertices are inside/outside
- Generate triangles according to 16 lookup tables

Case 3

Case 2
Marching Cubes (3D)

http://users.polytech.unice.fr/~lingrand/MarchingCubes/algo.html
Signed Distance Functions

**Pros**
- Full 3D model
- Sub-voxel accuracy
- Supports fast GPU implementations

**Cons**
- Space-consuming voxel grid
- Polygonization: slow
Application: Estimation of Traversable Terrain using TSDF

[Fallon, Deits, Whelan, Antone, McDonald, and Tedrake, Humanoids 2015]
Application: Learning Accurate 3D Models

[Sturm, Bylow, Kahl, Cremers; GCPR 2013]
3D Polygonal Maps for Humanoid Navigation

- Traditional representations need further processing to answer footstep queries
- Are dense (or semi-dense) representations that are not memory-efficient
3D Polygonal Maps for Humanoid Navigation

- 3D planar polygonal map
- **Memory efficient:** Represent planes using only 4 parameters and a polygonal boundary
- Empty space is implicit
- Footstep attitude given directly from plane orientation
- No constraints on polygons
Overview of Our Approach

1. **Plane Segmentation and Polygonization**
2. **Plane and Edge Matching**
3. **Robot Pose Update**
4. **Map Update**
Plane Segmentation and Polygonization

- Segment point cloud using off the shelf RGBD plane segmentation
- Polygonize by contour detection
- Identify and remove ground plane
Plane Matching

- Visible plane extraction
- Matching
- Pose Update
- Map Update
Edge Matching for Improved Localization

- Absence of sufficient planes for accurate localization is common
- Edge matching performed between two plane polygons that are matched
Pose Update

- All sample consensus of matched pairs of planes and edges
- The hypothesis with the largest vote count wins
Map Update

- Get expected map planes visible from updated pose
- Re-compute unique plane matches
- Update of plane parameters and map polygons
- Add unmatched sensed planes to map
Experimental Setup
Resulting Maps

- Sequence 1

- Sequence 2
Localization Results

Sequence 1

Sequence 2

ground truth  planes only  planes and edges
# Memory Consumption

<table>
<thead>
<tr>
<th></th>
<th>Height Map</th>
<th>Ours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map Area</td>
<td>3m x 3m</td>
<td>3m x 3m</td>
</tr>
<tr>
<td>No. of Map Elements</td>
<td>360000 cells</td>
<td>46 planes</td>
</tr>
<tr>
<td>Memory Usage</td>
<td>~29 KB</td>
<td>~9 KB</td>
</tr>
</tbody>
</table>
Average runtime per frame: ~100 ms
3D Polygonal Mapping

Our Experimental Setup
3D Polygons

Pros
- Footstep attitude given directly from plane orientation
- Memory efficient
- Fast to compute

Cons
- Coarse representation
- General objects are not well represented
Summary of 3D Polygonal Mapping

- A novel 3D polygonal map for footstep planning
- Highly memory efficient
- Footstep orientation directly obtainable
- Online map building
- Localization using planes and edges
Semantic Aware Volumetric Mapping

Key idea:

- Combine semantic scene understanding with geometric information to produce semantic aware volumetric mapping
- Semantics about objects/environment enable better abstraction and long term interaction
Semantic Aware Volumetric Mapping

- Convolutional networks e.g. Mask RCNN, Panoptic DeepLab, enable to label objects in RGB images
- Fuse per frame semantic information from RGB images with geometric information from range data

Figure from “Volumetric Instance-Aware Semantic Mapping and 3D Object Discovery” by Grinvald et al., 2019
Deep Learning based Scene Understanding

- Semantic Segmentation
  - Pixel level class labeling
  - No explicit object detection
  - Different instances of same class grouped together
  - No foreground/background differentiation
  - Application: Geo-sensing, autonomous driving

Figure from V7 Labs: https://www.v7labs.com/blog/panoptic-segmentation-guide
Deep Learning based Scene Understanding

- **Instance Segmentation**
  - Hybrid of object detection and semantic segmentation
  - Unique instance id for two different objects of same class
  - Mostly used for countable foreground objects like humans, chairs, cars
  - Application: Grasping, object tracking

Figure from V7 Labs: https://www.v7labs.com/blog/panoptic-segmentation-guide
Deep Learning based Scene Understanding

- Panoptic Segmentation
  - Combination of semantic and instance segmentation
  - Foreground (things) with instance ids
  - Background (stuff) with no instance ids
  - Richer semantic information
  - Long term object interaction

Figure from V7 Labs:
https://www.v7labs.com/blog/panoptic-segmentation-guide
Volumetric Instance Aware Mapping

The resulting representation contains information about both recognized semantic instances and newly discovered, previously unknown objects.

Grinvald, Furrer, Novkovic, Chung, Cadena, Siegwart, and Nieto, IEEE RAL 2019
Semantic Aware Volumetric Mapping

**Pros**
- Semantic rich maps
- Enables semantic based planning and interaction
- Closer to how humans abstract environment

**Cons**
- Requires large neural network models
- Complex and expensive

Figure from “Volumetric Instance-Aware Semantic Mapping and 3D Object Discovery” by Grinvald et al., 2019
Summary

- The best model depends on the application
- Voxel representations allow for a full 3D representation
- Octrees are a compact, inherently multi-resolution, probabilistic 3D representation
- Surface models support traversability analysis
- Signed distance functions also use 3D grids but allow for a sub-voxel accuracy representation of the surface
- 3D polygonal maps support efficient footstep planning
- Semantic aware volumetric maps enable long term interaction
Literature 3D World Models (1)

- Multi-Level Surface Maps for Outdoor Terrain Mapping and Loop Closing,
  R. Triebel, P. Pfaff, and W. Burgard,
  IEEE/RSJ Int. Conf. on Int. Robots and Systems (IROS), 2006

- OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees,

- World Modeling,
  W. Burgard, M. Herbert, and M. Bennewitz.

- Real-Time Camera Tracking and 3D Reconstruction Using Signed Distance Functions,
  E. Bylow, J. Sturm, C. Kerl, F. Kahl, and D. Cremers,
  Robotics: Science and Systems (RSS), 2013
Literature 3D World Models (2)

- Continuous Humanoid Locomotion over Uneven Terrain using Stereo Fusion,
  M. F. Fallon, P. Marion, R. Deits, T. Whelan, M. Antone, J. McDonald, and R. Tedrake,
  Humanoids 2015

- 3D Polygonal Mapping for Humanoid Robot Navigation,
  A. Roychoudhury, M. Missura, and M. Bennewitz, Humanoids 2022

- Volumetric Instance-Aware Semantic Mapping and 3D Object Discovery,
  M. Grinvald, F. Furrer, T. Novkovic, J.J. Chung, C. Cadena, R. Siegwart, & J. Nieto,
  IEEE Robotics and Automation Letters, 2019
ICP
Iterative Closest Point
Goal of This Chapter

- Last week: Overview of different world representations with their strengths and weaknesses
- Today: Understand how to calculate transformations between point clouds using the iterative closest point (ICP) algorithm
- Application of ICP to estimate local robot motion and learn accurate 2.5D/3D models of the environment
Motivation

If the pose of the robot is known, we can just accumulate point clouds to obtain an accurate 3D representation of the environment.
Challenge

- Odometry estimates are error prone due to slippage of the feet and noisy sensors
- Accordingly, consecutive observations may not align
- Error accumulates over time!
Solution

- Odometry estimates are error prone due to slippage of the feet and noisy sensors
- Accordingly, consecutive observations may not align
- Error accumulates over time!
- **Align point clouds to reduce the error**
Registration of 3D Data Points

- Estimate the relative transformation between two point clouds

- **Goal**: Find the parameters of the transformation that best align corresponding data points

- Iterative closest point (ICP) algorithm
Find Transformation to Align Points
Point Cloud Registration

\[ \{ y_n \} \quad \text{and} \quad \{ x_n \} \]
Point Cloud Registration
Point Cloud Registration

\{y_n\} \xrightarrow{C} \{x_n\}
Point Cloud Registration

\[ \bar{x}_n = R x_n + t \]

\{y_n\} \quad \{x_n\}
Point Cloud Registration

\[ \tilde{x}_n = R x_n + t \]

\[ \sum ||y_n - \tilde{x}_n||^2 \rightarrow \text{min} \]
Point Cloud Registration

\[ \bar{x}_n = Rx_n + t \]

\[ \{y_n\} \{\bar{x}_n\} \]
Registration of 3D Data Points

- Given two point sets:
  \[ Q = \{q_1, \ldots, q_N\} \quad P = \{p_1, \ldots, p_M\} \]
  with correspondences \( C = \{(i, j)\} \)

- Wanted: Translation \( t \) and rotation \( R \) that minimize the sum of the squared point-to-point distances:
  \[
  E(R, t) = \sum_{(i, j) \in C} \|q_i - (Rp_j + t)\|^2
  \]
Key Idea

If the correct correspondences are known, compute a translation and rotation to align the points in closed form.

\[ R \ t \]
Key Idea

If the correct correspondences are known, the rotation and translation can be computed directly by

- Computing a **shift** involving the **center of masses** (COMs) of both point clouds
- Performing a **rotational** alignment using singular value decomposition (**SVD**)


Shift via Center of Mass

- Compute the COMs of the corresponding points in both sets:
  \[ \mu_Q = \frac{1}{|C|} \sum_{(i,j) \in C} q_i \quad \mu_P = \frac{1}{|C|} \sum_{(i,j) \in C} p_j \]

- Subtract the corresponding COM from each point:
  \[ Q' = \{ q_i - \mu_Q \} = \{ q'_i \} \]
  \[ P' = \{ p_j - \mu_P \} = \{ p'_j \} \]
Error Minimization

- Minimizing the sum of the squared point-to-point distances

\[ E(R, t) = \sum_{(i,j) \in C} \| q_i - (R p_j + t) \|^2 \]

- Can be solved through singular value decomposition (SVD)
Singular Value Decomposition (SVD)

- Compute the cross-covariance matrix
  \[ W = \sum_{(i,j) \in C} q_i^T p_j \]

- Use the SVD to decompose
  \[ W = UDV^T \]
  
- The matrices \( U, V \) are 3x3 rotation matrices

- Diagonal matrix \( D = \text{Diag}(\sigma_1, \sigma_2, \sigma_3) \) singular values of \( W \)
Direct Computing of the Rotation Matrix

- Reminder: The goal is to minimize
  \[ E(R, t) = \sum_{(i,j) \in C} \| q_i - (Rp_j + t) \|^2 \]

- \( R \) can be directly computed using \( U \) and \( V \) from the SVD

\[ W = \sum_{(i,j) \in C} q'_i p'_j^T \]

\[ W = UDV^T \]

\[ R = VU^T \]
Direct Computing of the Translation Vector

Rotation: \( R = U V^T \)

Translation: \( t = \mu_Q - R\mu_P \)
SVD-Based Alignment (1)

- Compute means of the point sets
  \[
  \mu_Q = \frac{1}{|C|} \sum_{(i,j) \in C} q_i \\
  \mu_P = \frac{1}{|C|} \sum_{(i,j) \in C} p_j
  \]

- Compute cross covariance matrix based on mean-reduced coordinates
  \[
  W = \sum_{(i,j) \in C} q'_i p'_{j}^T
  \]
SVD-Based Alignment (2)

- Compute SVD
  \[ W = UDV^T \]
- Rotation matrix is given by
  \[ R = UV^T \]
- Translation vector is given by
  \[ t = \mu_Q - R\mu_P \]
- Translate and rotate points:
  \[ p_j \leftarrow Rp_j + \mu_Q - R\mu_P \]

rotation  translation
SVD-Based Alignment Summary

Alignment through translation and rotation

translate points to make the center of masses overlap

rotate points

Image courtesy: Ju
Point Cloud Registration

So far: assumed to be known!
Point Cloud Registration
ICP with Unknown Data Association

If the correct correspondences are not known, it is generally impossible to determine the optimal relative rotation and translation in one step.

Leads to an iterative estimation approach
Iterative Closest Point (ICP) Algorithm

- **Idea:** Iteratively estimate the data association and transformation
- **Converges if initial guess “close enough”**
  - Point locations or
  - Point correspondences

[Besl & McKay 92]
ICP Illustrated

1. Select points of one point cloud
2. Find closest points in other point cloud
3. Minimize distances
4. Iterate

[Courtesy of Rusinkiewicz]
Basic ICP Algorithm

error = inf
while (error decreased and error > threshold)
  ▪ Determine corresponding points
  ▪ Compute rotation $R$, translation $t$ via SVD
  ▪ Apply $R$ and $t$ to the points of the set to be registered
  ▪ error = $E(R, t)$
ICP Variants

- Several variants of standard ICP have been proposed
- Variants applied for humanoid perception and mapping use:
  - Only point subsets
  - A good data association strategy
Feature-Based Subsampling of Points

- Try to find “important” points
- Simplifies the search for correspondences
- Higher efficiency and accuracy
- Requires preprocessing
- Example:
  - In robot navigation with a down-looking camera, the ground plane typically dominates the scene
  - Filter out points belonging to the ground plane
Data Association

- Has serious effect on convergence and speed
- Popular matching methods:
  - Closest point
  - Point-to-plane
Closest-Point Matching

- Find closest point in the other point set (using kd-trees)

- Highly depends on initial guess of alignment
- Might lead to slow convergence and requires preprocessing
Point-to-Plane Error

- Idea: find closest points, take advantage of surface normal information
- Project point-to-point error vector onto surface normal

(source points)

destination points

point-to-point

point-to-plane
Point-to-Plane Error

- Idea: find closest points, take advantage of surface normal information
- Project point-to-point error vector onto surface normal
Point-to-Plane Error Metric

- Using point-to-plane distance instead of point-to-point lets flat regions “slide along each other”
- Each iteration is generally slower than the point-to-point version, however, often significantly better convergence rates
- Solved using non-linear least squares methods using an error metric that considers the surface normals

[see Low, 2004]
Rejecting Outlier Point Pairs

- Add a maximum matching threshold
- Accounts for the fact that some points will not have any correspondence in the second point set
- For example, points that are outside the boundary of the first scan A, or outliers
2D Point-to-Plane Example

Image courtesy: Bogoslavskyi
2D Point-to-Plane Example

Image courtesy: Bogoslavskyi
2D Point-to-Plane Example

Image courtesy: Bogoslavskyi
2D Point-to-Plane Example

Image courtesy: Bogoslavskyi
2D Point-to-Plane Example

Image courtesy: Bogoslavskyi
2D Point-to-Plane Example
2D Point-to-Plane Example

Image courtesy: Bogoslavskyi
2D Point-to-Plane Example

Image courtesy: Bogoslavskyi
2D Point-to-Plane Example

Image courtesy: Bogoslavskyi
2D Point-to-Plane Example

Image courtesy: Bogoslavskyi
Recap: Considered Problem

- Odometry estimates are error prone due to slippage of the feet and noisy sensors
- Accordingly, consecutive depth camera observations may not align
- Error accumulates over time!
- **Align point clouds to reduce the error**
Recap: Considered Problem

- Odometry estimates are error prone due to slippage of the feet and noisy sensors
- Accordingly, consecutive depth camera observations may not align
- Error accumulates over time!
- **Align point clouds to reduce the error**
Point Cloud Registration During Walking

- Initialize ICP from odometry estimate
- Transform $T^t_{\text{odom}}$: origin of the camera frame wrt. the world frame at time $t$ according to odometry
Point Cloud Registration During Walking

- Initialize ICP from odometry estimate with transform $T^t_{odom}$ describing the origin of the camera frame wrt. the world frame at time $t$

- Corrected pose $T^t = T^t_{corr} T^t_{odom}$ given by

$$T^t_{corr} = \arg \min_{T'} \sum_i \| T^{t-1} q_i - T' T^t_{odom} p_i \|^2$$

for a set of corresponding points $q_i \leftrightarrow p_i$ of two subsequently acquired point clouds
Point Cloud Registration During Walking

- Initialize ICP from odometry estimate with transform $T^t_{\text{odom}}$ describing the origin of the camera frame wrt. the world frame at time $t$

- Corrected pose $T^t = T^t_{\text{corr}} T^t_{\text{odom}}$ given by

$$T^t_{\text{corr}} = \arg\min_{T'} \sum_i \| T^{t-1} q_i - T'T^t_{\text{odom}} p_i \|^2$$

for a set of corresponding points $q_i \leftrightarrow p_i$ of two subsequently acquired point clouds

- ICP iteratively improves $q_i \leftrightarrow p_i$ and $T^t_{\text{corr}}$
Point Cloud Registration During Walking

- Estimated corrected pose of the camera at time $t-1$

\[ T^{t-1} = T_{\text{corr}}^{t-1} T_{\text{odom}}^{t-1} \]
Point Cloud Registration During Walking

- Estimated corrected pose of the camera at time $t-1$
- Estimated pose of the camera at time $t$ according to odometry between the time steps and $T_{t-1}^{t}$

\[ T_{t-1}^{t} = T_{corr}^{t-1} T_{odom}^{t-1} \]

$T_{odom}^{t}$
Point Cloud Registration During Walking

- Estimated corrected pose of the camera at time $t$ after scan matching via ICP
Application: Learning a Height Map from Depth-Camera Observations

- 2D grid map
- Height estimate for each cell
Evaluation of Pose Tracking

- Drift 5.9 cm/m after ICP vs. 15.3 cm/m error from odometry per 1m distance
- Ground truth from external motion capture system
- Low drift allows construction of accurate height maps
ICP Summary (1)

- Registering point clouds is a central task in perception and mapping
- ICP is the standard algorithm for point cloud alignment
- Computes the translation and rotation between point clouds
- Given the data association of the two point sets, the transformation can be computed efficiently using SVD
ICP Summary (2)

- Challenge: Determination of the correct data associations
- ICP does it iteratively (data association and transformation)
- ICP does not always converge to the correct alignment (but to a local minimum)
- Performance depends on the initial guess (e.g., odometry)
- In practice often used: least squares considering surface normals, closest point data association, and outlier rejection
- ICP be used to estimate local motion and learn accurate models of the environment
Literature ICP

- Efficient Variants of the ICP Algorithm, S. Rusinkiewicz and M. Levoy, Prof. of the Int. Conf. on 3D Digital Imaging and Modeling, 2001
- Linear Least-Squares Optimization for Point-to-Plane ICP Surface Registration, K.-L. Low, Technical Report Univ. of North Carolina, 2004
Acknowledgment

- Previous versions of parts of the slides have been created by Wolfram Burgard, Cyrill Stachniss, and Jürgen Sturm