Registering point clouds using the Point Cloud Library.

Dirk Holz, University of Bonn
January 27, 2011
Registration = *aligning point clouds* and determining where they have been acquired

- where = position and orientation in 3D space
- goal = constructing consistent 3D (point) models
Introduction (2/3)

Standard: Iterative Corresponding/Closest Points (ICP)

Given an input point cloud and a target point cloud

1. determine pairs of corresponding points,
2. estimate a transformation that minimizes the distances between the correspondences,
3. apply the transformation to align input and target.

Dirk Holz
Intern Presentation
Possible Application: 3D model construction using Microsoft Kinect

Microsoft Kinect: (Image from microsoft.com)

- acquires both 3D information and color (xyz + rgb)
- Point clouds contain $640 \times 480 = 307200$ points
- Measurement frequency: 30Hz

Goal:

- Online registration of point clouds in real-time
- Combining the “best” from the 2D and the 3D world
1. Correspondence Estimation
2. Correspondence Rejection
3. Transformation Estimation
4. Incremental Registration
5. Examples
6. Outlook
Determining Correspondences:

- for all points in the input point cloud (searching through all points in the target point cloud) e.g. closest points in 3D space as in original ICP
- for a subset of points in input and target point cloud
- for a set of keypoints + feature descriptors

“using the best of both worlds”

- 3D keypoints+descriptors: NARF, PFH, FPFH, VPFH, ...
- 2D keypoints: FAST, STAR, SIFT, SURF, ...
- 2D descriptors: BRIEF, SIFT, SURF, ...
Correspondence Estimation in `::pcl::registration`

- `pcl::registration::CorrespondenceEstimation`
- matches pcl point types, custom point types, and 3D/pcl feature descriptors
- but no OpenCV dependency in pcl!
- 2D keypoints/descriptors need to be computed and matches outside of pcl
- can be fed back and used in pcl using . . .

Correspondences in `::pcl::registration`

- `pcl::registration::Correspondence`
- Defined as a tuple \((\text{index}_{\text{query}}, \text{index}_{\text{match}}, \text{distance})\) where
  - \(\text{index}_{\text{query}}\): index of a point in the input point cloud
  - \(\text{index}_{\text{match}}\): index of the point in the target point cloud that matches the query point
  - \(\text{distance}\): distance between the two points
Rejection of False Correspondences

Problem:
- False correspondences negatively affect the registration
- The estimated transformation does not correctly align input and target point clouds

Correspondence Rejection:
- Sorting out correspondences that are probably false w.r.t some criteria \(\rightarrow\) different rejection methods
Correspondence Rejection Methods

- **Distance-based:** Remove all correspondences with a distance larger than some threshold
  \[ \text{pcl::registration::CorrespondenceRejectorDistance} \]

- Based on estimated **overlap:** Only keep the best \( k \% \) of the determined correspondences (as used in the "TrimmedICP")
  \[ \text{pcl::registration::CorrespondenceRejectorTrimmed} \]

- **Reciprocal** rejection / correspondence estimation: Only keep those correspondences where the query point is the best match for the matching point
  \[ \text{pcl::registration::CorrespondenceEstimation} \]
Correspondence Rejection Methods

**RANSAC-based rejection:**
- randomly selects three correspondence pairs,
- determines a transformation aligning these pairs, and
- uses all correspondences to evaluate the quality of the transformation
- robustly detects outliers (up to a certain amount)

`pcl::registration::CorrespondenceRejectorSAC`
Transformation estimation
(minimizing the distances between correspondences)

- using **closed-form solutions**, e.g., SVD-based
  `pcl::registration::TransformationEstimationSVD`
- using linear/non-linear optimization
- using RANSAC (correspondence rejection)
  `pcl::registration::CorrespondenceRejectorSAC`
- more to come . . .
Incremental Registration (1/1)

Multi-view point cloud registration

- **Pairwise Registration**: cloud0 ← cloud1 ← cloud2 ← ...  
  ➞ Registration errors accumulate, larger loops are not possible

- **Global Registration**:  
  either match all point clouds against each other (off-line) or using SLAM approach with frontend and backend.  
  ➞ loop closing and consistent models

- **Incremental Registration** (somewhere in between):  
  ✭ incrementally build a model using all registered clouds  
  ✭ avoid duplicate storage of points (e.g. subsampling)  
  ✭ can close smaller loops (without corrections)  
  ✭ is fast
Examples (1/4)

Incrementally building an object model
Examples (2/4)

Fast Incremental Registration using Range Image Borders
Examples (3/4)

Incrementally building a subsampled environment model
Examples (4/4)

Incrementally building a subsampled environment model
Currently under construction:

- Full SLAM approach using
- A robust frontend using both 2D and 3D keypoints/descriptors
- Efficient and robust backend using graph optimization
Introduction Correspondences Rejection Transformation Registration Examples Outlook

Questions?

http://pcl.ros.org/
Correspondences

```
pcl::registration::CorrespondenceRejectionOneToOne
pcl::registration::CorrespondenceRejectionSAC
```