Point Cloud Registration
Including slides from Dirk Holz, Jochen Sprickerhof

Matteo Munaro
matteo.munaro@dei.unipd.it
Intelligent Autonomous Systems Lab (IAS-Lab) – University of Padova
PCL Tutorial – IAS 2014
Padova, July 15, 2014
Registering 3D Point Clouds

- Given a **source** point cloud and a **target** point cloud
  1. determine **correspondence pairs**,
  2. estimate a transformation that aligns the **correspondences**,
  3. apply the transformation to align **source** and **target**.

- **correspondences** can be anything (points/features/...)!
Registration Pipeline in PCL

Data acquisition → Keypoints estimation → Feature descriptors estimation → Correspondences estimation (matching) → Correspondence rejection method 1 → Correspondence rejection method 2 → ... → Correspondence rejection method N → Transformation estimation
Alignment via Feature Matching

1. Compute sets of keypoints
2. Compute (local) feature descriptors (e.g. FPFH)
3. Match features to find correspondences
4. Estimate transformation from correspondences
Alignment via Feature Matching

**Code:** Matching Features

```cpp
CorrespondenceEstimation<FeatureT, FeatureT> est;
est.setInputCloud (source_features);
est.setInputTarget (target_features);
est.determineCorrespondences (correspondences);
```

**Example:** Found correspondences / matches

```
$ correspondence_viewer robot0 robot1 -n 50
```
Transformation Estimation

- Several methods for computing a transformation $T = (R, t)$ given correspondence pairs $(d_i, m_i)$:
  - Point-to-point
  - Point-to-plane
  - Plane-to-plane
  - ... and many others

- Simple solution (based on SVD) for minimizing point-to-point distance (least squares error $E$):

$$E(T) = \sum_i (m_i - (Rd_i + t))^2$$

```cpp
pcl::registration::TransformationEstimationSVD
```
Alignment via Feature Matching

**Code: Transformation estimation**

```cpp
Eigen::Matrix4f transformation;
TransformationEstimationSVD<PointT, PointT> svd;
svd.estimateRigidTransformation(source, target,
correspondences, transformation);
```

**Example: Simple initial alignment**

**Problem: False correspondences!**
Rejecting false correspondences (outliers) using SAC

- Draw three correspondences pairs \( d_i, m_i \)
- Estimate transformation \( (R, t) \) for these samples
- Determine inlier pairs with \( ((Rd_i + t) - m_i)^2 < \epsilon \)
- Repeat \( N \) times, and use \( (R, t) \) having most inliers

**Code:** SAC-based correspondence rejection

```cpp
CorrespondenceRejectorSampleConsensus<PointT> sac;
sac.setInputCloud(source);
sac.setTargetCloud(target);
sac.setInlierThreshold(epsilon);
sac.setMaxIterations(N);
sac.setInputCorrespondences(correspondences);
sac.getCorrespondences(inliers);
Eigen::Matrix4f transformation = sac.getBestTransformation();
```
Example: SAC-based correspondence rejection

Initial correspondences:

Inliers:
With SAC-based correspondence rejection:
Let $M$ be a model point set. Let $S$ be a scene point set. We assume:

1. $N_M = N_S$.
2. Each point $S_i$ corresponds to $M_i$.

If correct correspondences are known, we can find correct relative rotation/translation.
How to find correspondences?

- User input?
- Feature detection?
- Signatures?

Alternative: assume closest point correspond.
Converges if starting position is “close enough”

Finding each match is performed in $O(N_M)$ worst case

At every iteration $(R, T)$ are estimated and $S$ is updated according to them:

- $S_{new} = R \times S + T$
The algorithm:

1. Init the error to ∞
2. Calculate correspondence
3. Calculate alignment (R,T)
4. Apply alignment
5. Update error
6. If error > threshold

Iterative Closest Point
The algorithm:

function ICP(Scene,Model)
begin
E\` \(\leftarrow\) + \(\infty\);
(Rot,Trans) \(\leftarrow\) In Initialize-Alignment(Scene,Model);
repeat
repeat
E \(\leftarrow\) E`;
Aligned-Scene \(\leftarrow\) Apply-Alignment(Scene,Rot,Trans);
Pairs \(\leftarrow\) Return-Closest-Pairs(Aligned-Scene,Model);
(Rot,Trans,E`) \(\leftarrow\) Update-Alignment(Scene,Model,Pairs,Rot,Trans);
Until |E` - E| < Threshold
return (Rot,Trans);
end
Rejecting pairs:
- distance thresholding
Rejecting pairs:
- points on end vertices
Rejecting pairs:
- inconsistent pairs
The **ICP API**

```cpp
pcl::IterativeClosestPoint<InType, OutType> icp;
Provide a pointer to the **input point cloud**
icp.setInputCloud (input_cloud);
Provide a pointer to the **target point cloud**
icp.setInputTarget (target_cloud);
Align **input** to **target** to obtain
icp.align (aligned_cloud);
Eigen::Matrix4f transformation = icp.
getFinalTransformation ();
```

- **the aligned cloud** (transformed copy of input cloud),
- and **transformation** used for alignment.
The **ICP API: Termination Criteria**

- **Max. number of iteration steps**
  → set via `setMaximumIterations(nr_iterations)`

- **Convergence**: Estimated transformation doesn't change (the sum of differences between current and last transformation is smaller than a user-defined threshold)
  → set via `setTransformationEpsilon(epsilon)`

- **A solution was found**: (the sum of squared errors is smaller than a user-defined threshold)
  → set via `setEuclideanFitnessEpsilon(distance)`
With ICP refinement:
Multi-cloud Registration
Hands-on session
Any questions?

matteo.munaro@dei.unipd.it