3D Object Recognition in Clutter with the Point Cloud Library

Federico Tombari, Ph.D
federico.tombari@unibo.it

University of Bologna
Open Perception
Data representations in PCL

- PCL can deal with both organized (e.g. range maps) and unorganized point clouds
  - if the underlying 2d structure is available, efficient schemes can be used (e.g. integral images instead of kd-tree for nearest neighbor search)
- Both are handled by the same data structure (pcl::PointCloud, templated thus highly customizable)
  - Points can be XYZ, XYZ+normals, XYZI, XYZRGB, ...
  - Support for RGB-D data
- Voxelized representations are implemented by pcl::PointCloud + voxelization functions (e.g. voxel sampling)
  - no specific types for voxelized maps
- Currently rather limited support for 3D meshes
Object Recognition and data representations

- Usually Object Recognition in clutter is done on 2.5 data (model views against scene views).
- Can be done also 3D vs 3D, although scenes are usually 2.5D (and 3D vs. 2.5D does not work good).
- When models are 3D, we can render 2.5D views simulating input from a depth sensor:

```cpp
pcl::apps::RenderViewsTesselatedSphere render_views;
render_views.setResolution (resolution_);
render_views.setTesselationLevel (1); // 80 views
render_views.addModelFromPolyData (model); // vtk model
render_views.generateViews ();
std::vector< pcl::PointCloud<pcl::PointXYZ>::Ptr > views;
std::vector < Eigen::Matrix4f > poses;
render_views.getViews (views);
render_views.getPoses (poses);
```
Typical paradigm for finding similarities between two point clouds

1. Extract compact and descriptive representations (3D descriptors) on each cloud (possibly over a subset of salient points)

2. Match these representations to yield (point-to-point) correspondences

Applications: 3D Object recognition, cloud registration, 3D SLAM, object retrieval, ..
3D keypoints are

- **Distinctive**, i.e. suitable for effective description and matching (globally definable)
- **Repeatable** with respect to point-of-view variations, noise, etc… (locally definable)

The `pcl::keypoint` module includes:

- A set of detectors specifically proposed for 3D point clouds and range maps
  - Intrinsic Shape Signatures (ISS) [Zhong 09]
  - NARF [Steder 11]
  - (Uniform Sampling, i.e. voxelization)
- Several detectors «derived» from 2D interest point detectors
  - Harris (2D, 3D, 6D) [Harris 88]
  - SIFT [Lowe 04]
  - SUSAN [Smith 95]
  - AGAST [Mair 10]
  - ...

Results from [Tombari 13]
Global vs local descriptors

- **Pcl::Features: compact** representations aimed at detecting similarities between surfaces (*surface matching*)
- based on the support size
  - **Pointwise descriptors**
    - Simple, efficient, but not robust to noise, often not descriptive enough (e.g. normals, curvatures, ..)
  - **Local/Regional descriptors**
    - Well suited to handle clutter and occlusions
    - Can be vector quantized in codebooks
    - Segmentation, registration, recognition in clutter, 3D SLAM
  - **Global descriptors**
    - Complete information concerning the surface is needed (no occlusions and clutter, unless pre-processing)
    - Higher invariance, well suited for *retrieval and categorization*
    - More descriptive on objects with poor geometric structure (household objects..)
### Summing up..

<table>
<thead>
<tr>
<th>Method</th>
<th>Category</th>
<th>Unique LRF</th>
<th>Texture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Struct. Indexing</td>
<td>Signature</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>[Stein92]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PS</td>
<td>Signature</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>[Chua97]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3DPF</td>
<td>Signature</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>[Sun01]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3DGSS</td>
<td>Signature</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>[Novatnack08]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KPQ</td>
<td>Signature</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>[Mian10]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3D-SURF</td>
<td>Signature</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>[Knopp10]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SI</td>
<td>Histogram</td>
<td>RA</td>
<td>No</td>
</tr>
<tr>
<td>[Johnson99]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSP</td>
<td>Histogram</td>
<td>RA</td>
<td>No</td>
</tr>
<tr>
<td>[Chen07]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3DSC</td>
<td>Histogram</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>[Frome04]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ISS</td>
<td>Histogram</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>[Zhong09]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USC</td>
<td>Histogram</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>[Tombari10]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PFH</td>
<td>Histogram</td>
<td>RA</td>
<td>No</td>
</tr>
<tr>
<td>[Rusu08]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FPFH</td>
<td>Histogram</td>
<td>RA</td>
<td>No</td>
</tr>
<tr>
<td>[Rusu09]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tensor</td>
<td>Histogram</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>[Mian06]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RSD</td>
<td>Histogram</td>
<td>RA</td>
<td>No</td>
</tr>
<tr>
<td>[Marton11]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HKS</td>
<td>Other</td>
<td>-</td>
<td>No</td>
</tr>
<tr>
<td>[Sun09]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MeshHoG</td>
<td>Hybrid</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>[Zaharescu09]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHOT</td>
<td>Hybrid</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>[Tombari10]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PFHRGB</td>
<td>Hybrid</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Histogram</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

: in PCL
3D Object Recognition in clutter

- **Definition (typical setting):**
  A. a set of 3D models (often, in the form of views)
  B. one scene (at a time) including one or more models, possibly (partially) occluded, + clutter.

- Models can be present in **multiple instances** in the same scene

- **Goal(s):**
  - determine which model is present in the current scene
  - (often) estimate the 6DoF pose of the model wrt. the scene

- Applications: industrial robotics, quality control, service robotics, autonomous navigation, ..
Pipelines

LOCAL PIPELINE
- Keypoint Extraction
- Description
- Matching
- Correspondence Grouping
- Absolute Orientation

GLOBAL PIPELINE
- Segmentation
- Description
- Matching
- Alignment

ICP refinement
Hypothesis Verification
Problem:

- given a set of point-to-point correspondences $C = \{c_1, c_2, \ldots, c_n\}$
  - where $c_i = (p_{i,s}, p_{i,m})$
- divide C intro groups (or clusters) each holding consensus for a specific 6DOF transformation
- non-grouped correspondences are considered outliers

General approach: RANSAC [Fischler 81]

- the model is represented by the 6DOF transformation obtained via Absolute Orientation, its parameters being a 3D rotation and a 3D translation
- Approaches include specific geometric constraints deployed in the 3D space
Enforcing geometric consistency between pairs of correspondences [Chen 07]

- Choose a seed correspondence $c_i$
- Test the following pairwise geometric constraint between $c_i$ and all the other correspondences $c_j$:
  \[
  \left\| \|p_{i,m} - p_{j,m}\| - \|p_{i,s} - p_{j,s}\| \right\| < \epsilon, \forall j
  \]
- Add each correspondence holding the constraint to the group seeded by $c_i$ (and remove it from the list)
- Eliminate groups having a small consensus set
Geometric consistency (2)

- GC enforces a 1D constraint over a transformation with 6 DoF -> weak constraint, high number of ambiguities (might fail if the number of correspondences is low!)
- Pro: extremely simple and efficient
- Use if many correspondences, noisy data

```cpp
pcl::CorrespondencesPtr m_s_corrs; //fill it
std::vector<pcl::Correspondences> clusters; //output
pcl::GeometricConsistencyGrouping<PT, PT> gc_clusterer;
gc_clusterer.setGCSize (cg_size); //1st param
gc_clusterer.setGCThreshold (cg_thres); //2nd param
gc_clusterer.setInputCloud (m_keypoints);
gc_clusterer.setSceneCloud (s_keypoints);
gc_clusterer.setModelSceneCorrespondences (m_s_corrs);
gc_clusterer.cluster (clusters);
```

(m_s_corrs are correspondences with indices to m_keypoints and s_keypoints)
3D Hough Transform

- 3D Hough Transform [Vosselman 04], [Rabbani 05]
  - Extension of “classic” 2D Hough Transform
  - Can handle a small number of parameters
    - simple surfaces (cylinders, spheres, cones, ..)
    - no generic free-form objects

- 3D Generalized Hough Transform [Khoshelham 07]:
  - Normals in spite of gradient directions
  - 6D Hough space: sparsity, high memory requirements
  - \( O(MN^3) \): complexity of voting stage (M: 3D points, N: quant. intervals)
  - Can hardly be used in practice [Khoshelham 07]
PoV-independent vote casting

- Using the definition of a local RF, global-to-local and local-to-global transformations of 3D vectors can be defined:
  - $v_i^L = R_{GLi} \cdot v_i^G$
  - $v_i^G = R_{LiG} \cdot v_i^L$

![Diagram showing transformations between local and global frame vectors](attachment://diagram.png)
3D Hough Voting [Tombari 10]

- **Training stage**
  - A unique reference point (e.g. the centroid) is used to cast votes for each feature.
  - These votes are transformed in the local RF of each feature to be PoV-independent:
    \[ v_{i,m}^L = R_{GLi} \cdot (b_m - p_{i,m}) \]

- **Testing stage**:
  - For each correspondence \( c_i \), its scene point \( p_{i,s} \) casts a PoV-independent vote:
    \[ v_{i,s}^G = R_{LiG} \cdot v_{i,s}^L + p_{i,s} \]
Correspondence votes are accumulated in a 3D Hough space.

Coherent votes point at the same position of the reference point $b_s$.

Local maxima in the Hough space identify object instances (handles the presence of multiple instances of the same model).

The LRF allows reducing the voting from 6D to 3D (only translation).

Votes can be interpolated to handle the approximation introduced by voting space quantization.
3D Hough example in PCL

typedef pcl::ReferenceFrame RFType;
pcl::PointCloud<RFType>::Ptr model_rf; //fill with RFs
pcl::PointCloud<RFType>::Ptr scene_rf; //fill with RFs
pcl::CorrespondencesPtr m_s_corrs; //fill it
std::vector<pcl::Correspondences> clusters;
pcl::Hough3DGrouping<PT, PT, RFType, RFType> hc;
hc.setHoughBinSize (cg_size);
hc.setHoughThreshold (cg_thres);
hc.setUseInterpolation (true);
hc.setInputCloud (m_keypoints);
hc.setInputRf (model_rf);
hc.setSceneCloud (s_keypoints);
hc.setSceneRf (scene_rf);
hc.setModelSceneCorrespondences (m_s_corrs);
hc.cluster (clusters);
Absolute Orientation

- Given a set of "coherent" correspondences, determine the 6DOF transformation between the model and the scene (3x3 rotation matrix $R$ - or equivalently a quaternion - and 3D translation vector $T$).
- Under the assumption that no outlier is present (conversely to, e.g., ICP).
- Given this assumption, the problem can be solved in closed solution via Absolute Orientation [Horn 87] [Arun 87]:
  - given a set of $n$ exact correspondences $c_1 = \{p_{1,m}, p_{1,s}\}, \ldots, c_n = \{p_{n,m}, p_{n,s}\}$, $R$ and $T$ are obtained as
    \[
    \arg\min_{R,T} \sum_{i=1}^{n} \left\| p_{i,s} - R \cdot p_{i,m} - T \right\|^2
    \]
  - Simply a derivation of the least square estimation problem with 3D vectors.
Segmentation for global pipelines

- With cluttered scenes global descriptors require data pre-segmentation
- General approach: smooth region segmentation [Rabbani 06]
  - Region growing:
    - Starting from a seed point $p_s$, add to its segment all points in its neighborhood that satisfy:
      \[ \| p_s - p_i \|_2 < \tau_d \cap n_s \circ n_i > \tau_n \]
    - Iterate for all the newly added points, considered as seeds
  - Fails with high object density, non-smooth objects
MPS Segmentation

MPS (Multi-Plane Segmentation) [Trevor13]:

- Fast, general approach focused on RGB-D data (depth + color)
- Works on “organized” point clouds (neighboring pixels can be accessed in constant time)
- Computes connected components on the organized cloud (exploiting 3D distances, but also normals and differences in the RGB space)

- PCL class: `pcl::OrganizedMultiPlaneSegmentation`
- See PCL’s Organized Segmentation Demo for more: `pcl_organized_segmentation_demo`

- Once planes are detected, objects can be found via Euclidean clustering.
  - E.g., class `pcl::OrganizedConnectedComponentSegmentation` can follow using the extracted planes as a mask
Both the local and global pipelines provide a set of Object Hypotheses $H = \{h_1, h_2, \ldots, h_n\}$, where $h_i = \{M_i, R_i, t_i\}$.

Several hypotheses are false positives: how do you discard them without compromising the Recall?

Typical geometric cues being enforced [Papazov10] [Mian06] [Bariya10] [Johnson 99]

- % of (visible) model points being explained by the scene (i.e. having one close correspondent), aka inliers
- Number of unexplained model points, aka outliers

These cues are applied sequentially, one hypothesis at a time.

Two such methods in PCL:
- `pcl::GreedyVerification`
- `pcl::PapazovHV`
Global Hypothesis Verification

- [Aldoma12]
- **Simultaneous** geometric verification of all object hypotheses
- Consider the two possible states of a single hypothesis: $x_i = \{0, 1\}$ (inactive/active).
- By switching the state of an hypothesis, we can evaluate a global cost function that estimates how good the current solution $\chi = \{x_1, x_2, .., x_n\}$ is.
- A global cost function integrating four geometrical cues is minimized:
  $$\mathcal{I}(\chi): \mathbb{B}^n \rightarrow \mathbb{R}, \chi = \{x_1, \ldots, x_n\}, x_i \in \mathbb{B} = \{0, 1\}$$
- $\mathcal{I}(\chi)$ considers the whole set of hypotheses as a global scene model instead of considering each model hypothesis separately.
Global Hypothesis Verification (2)

\[ \mathcal{S}(\chi) = \sum_{p \in S} (\Lambda_{\chi}(p) + \Upsilon_{\chi}(p) - \Omega_{\chi}(p)) + \lambda \cdot \sum_{i=1}^{n} |\Phi_{h_i}| \cdot x_i \]

- \(\Omega_{\chi}\): scene inliers
- \(\Lambda_{\chi}\): multiple assignment
- \(\Upsilon_{\chi}\): clutter
- \(|\Phi_{h_i}|\): #outliers for \(h_i\)

- Maximize number of scene points explained (orange).
- Minimize number of model outliers (green).
- Minimize number of scene points multiple explained (black).
- Minimize number of unexplained scene points close to active hypotheses (yellow, purple).
- Optimization solved using Simulated Annealing or other metaheuristics (METSLib library).
Using GHV in PCL

```cpp
pcl::GlobalHypothesesVerification<pcl::PointXYZ, pcl::PointXYZ> goHv;

goHv.setSceneCloud (scene);

goHv.addModels (aligned_hypotheses, true);

goHv.setResolution (0.005f);

goHv.setInlierThreshold (0.005f);

goHv.setRadiusClutter (0.04f);

goHv.setRegularizer (3.f); // outliers' model weight

goHv.setClutterRegularizer (5.f); // clutter points weight

goHv.setDetectClutter (true);

goHv.verify ();

std::vector<bool> mask_hv;

goHv.getMask (mask_hv);
```
Multi-pipeline HV [Aldoma 13]

- Typical scenario: objects in clutter laying on a dominant plane, RGB-D data
- Exploiting multiple pipelines from RGB-D data to handle different object characteristics (low texture, non-distinctive 3D shape, occlusions, ...)
  - Hypothesis Verification stage based on GO [Aldoma ECCV12] fits well the scenario
- Injection of RGB information along different modules
  - 3D «global» description
  - Hypothesis Verification
Multi-pipeline HV (2)
THE END
(of theory part.. let’s try out the hands-on tutorial!)