3D Sensing and Point Clouds

November 8, 2013

Table of Contents

Table of Contents

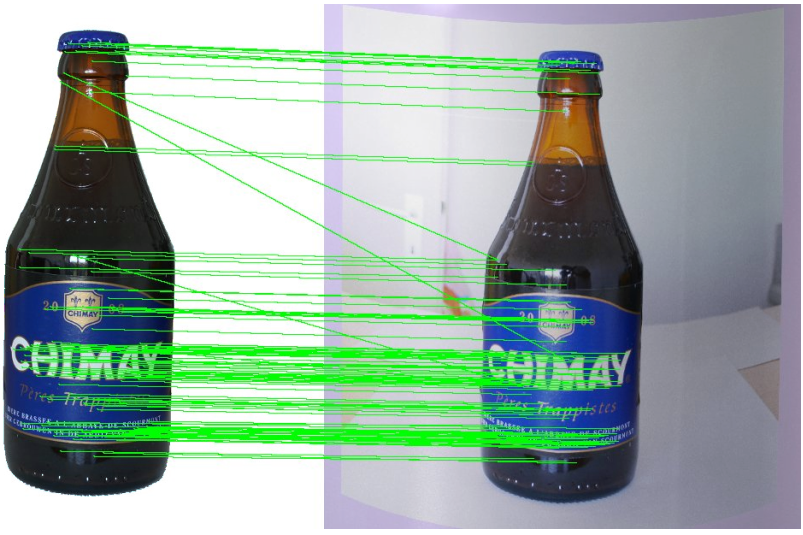
Camera vs. 3D sensor

- RGB camera
  1. Affected by light and artifacts.
  2. No reliable 3D info.

- Depth sensors
  1. Depth segmentation trivial.
  2. Can be combined with RGB.

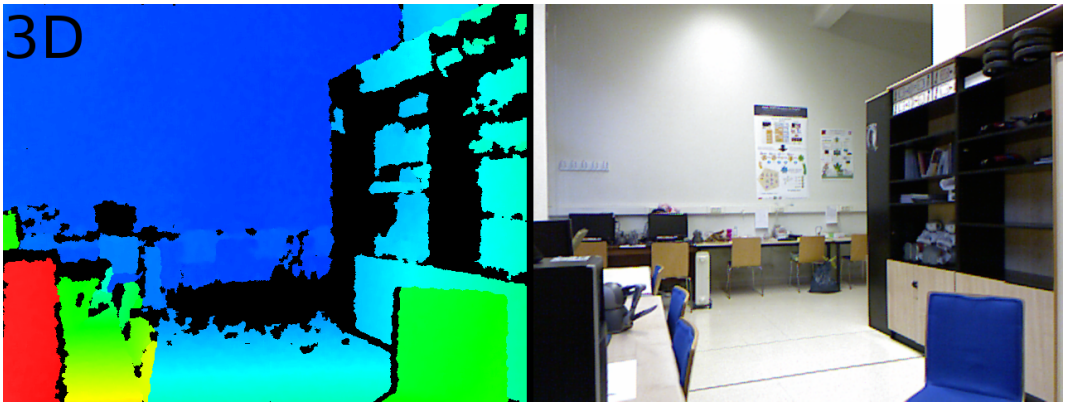
Camera vs. 3D sensor

Camera vs. 3D sensor

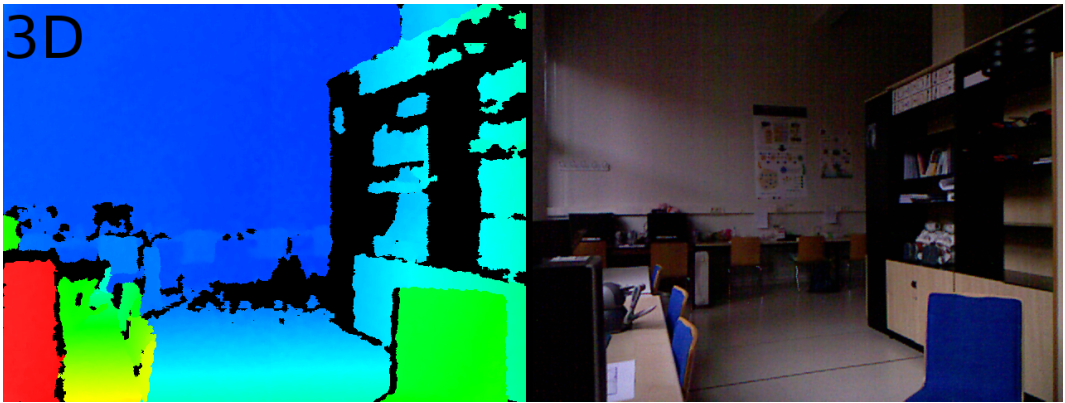


Camera vs. 3D sensor

Camera vs. 3D sensor



Camera vs. 3D sensor



Types of depth sensors

1. Triangulation
   - Stereo cameras
2. Time-of-Flight (ToF)
   - LIDAR
   - ToF camera
3. Structured light
   - Kinect

Stereo cameras

- Passive measurement.
- Requires calibrated pair of sensors. Perfect calibration is impossible.
- Quality depends on objects' textures, but is generally low.

Stereo cameras

Steps:

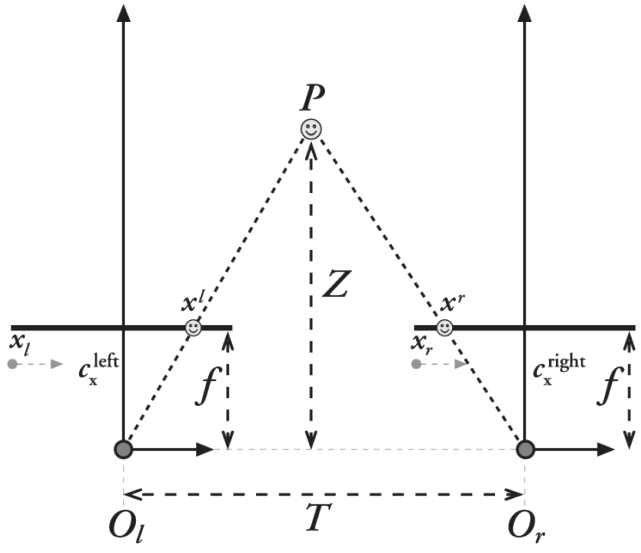
1. *Undistortion*: remove radial and tangential lens distortion.
2. *Rectification*: adjust for the angles and distances between cameras.
3. *Correspondence*: find the same features in the left and right images.
4. *Reprojection*: use triangulation to get a depth map from disparities.

Stereo cameras

$$d = \frac{fT}{||x_1 - x_2||}$$

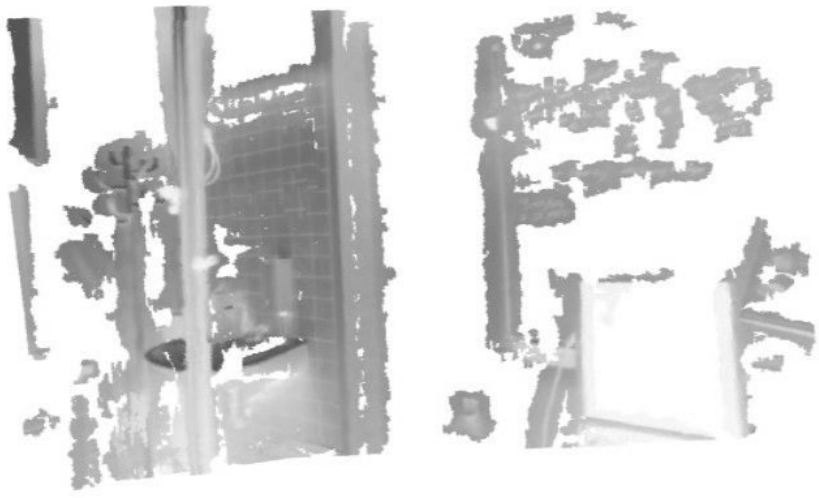
d: distance to be estimated
f: focal distance of both cameras
T: distance between the cameras
x_1, x_2: corresponding points in the two images

Stereo cameras

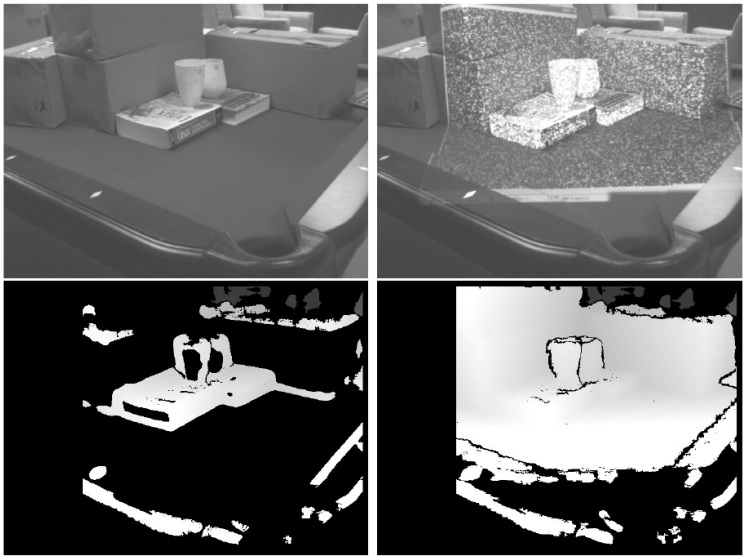


Stereo cameras

Stereo cameras



Stereo cameras



LIDAR

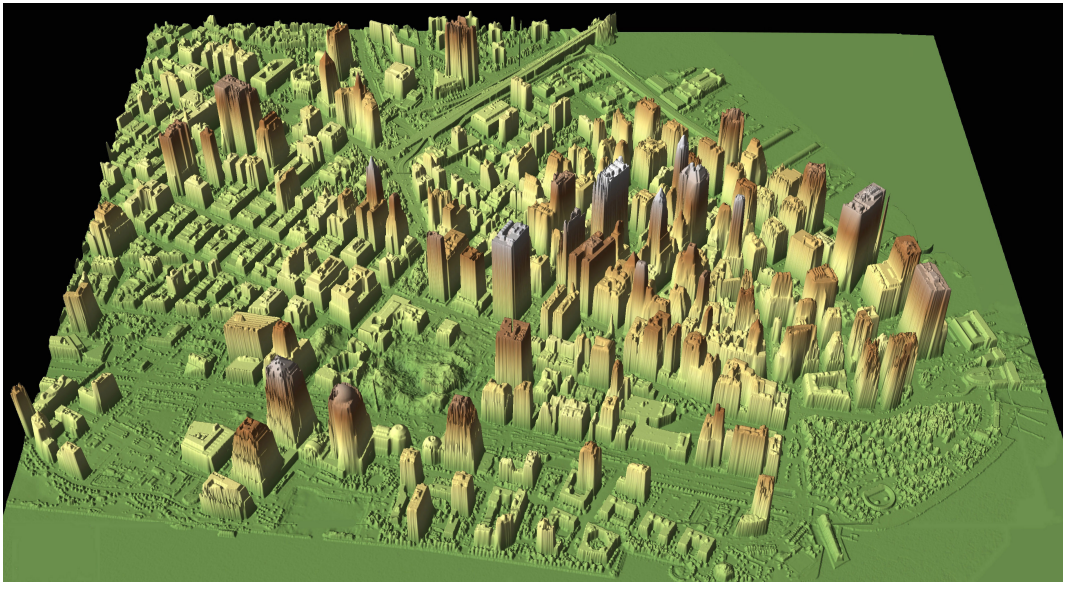
- LIght Detection And Ranging.
- Uses laser beam, with a lot of range.
- High-res and accurate; slow and expensive.

LIDAR

Distance:

$$d = \frac{ct}{2}$$

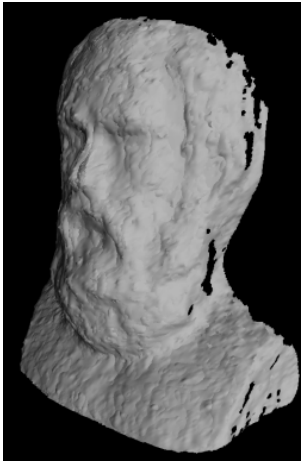
c: speed of light (or measuring signal)
t: time taken for signal to come back

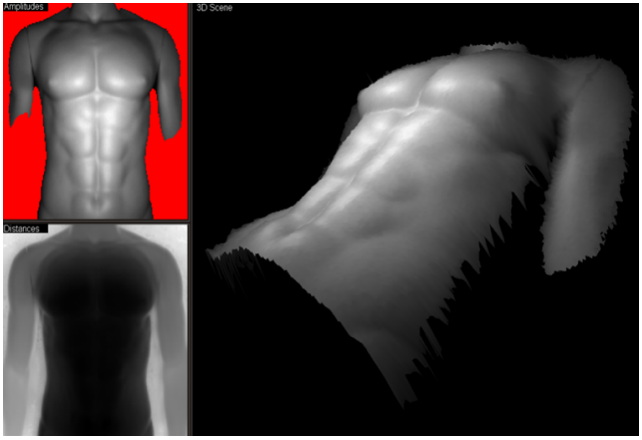


ToF cameras

- Each pulse captures the entire scene, not just a point.
- Very fast (30-100 Hz), low-res (1cm max), range $\sim$6m, cheap.
- Infrarred mostly.

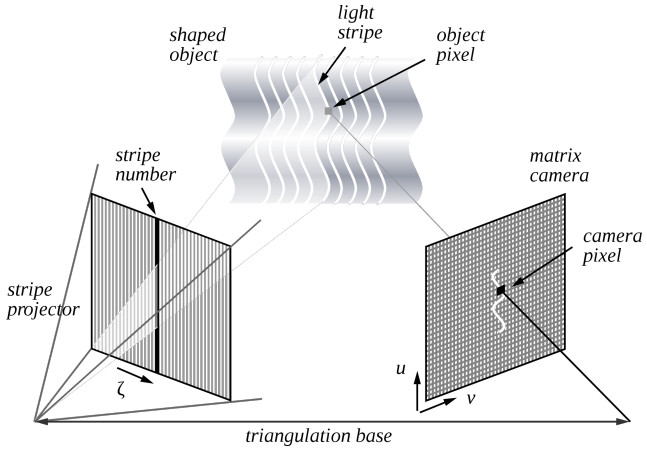
ToF cameras



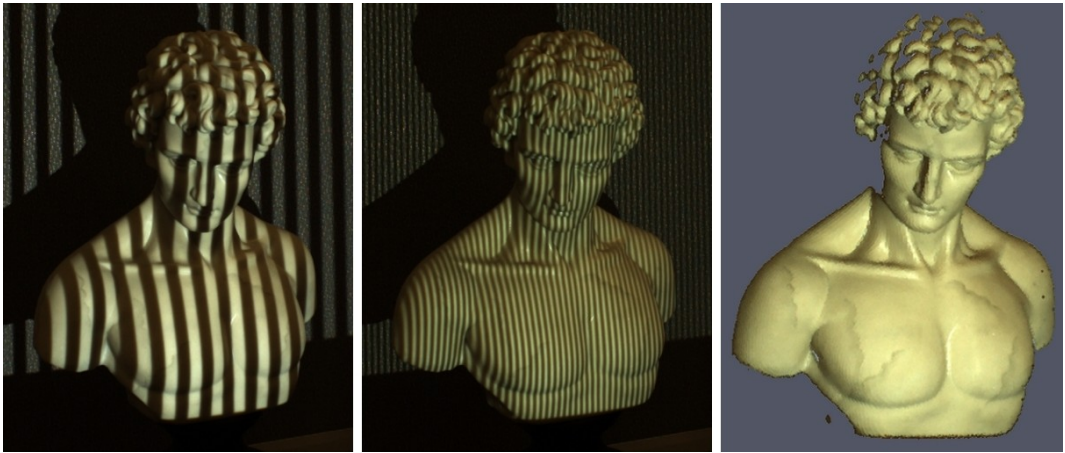


Structured light

Projects a known pattern of light and analyzes deformation.

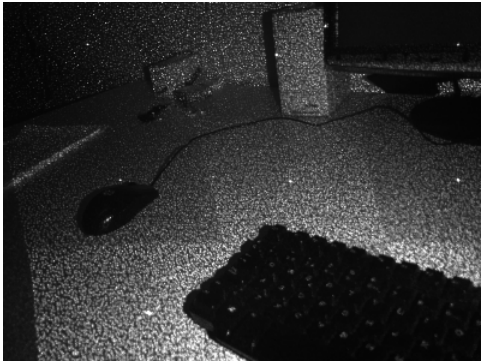


Structured light



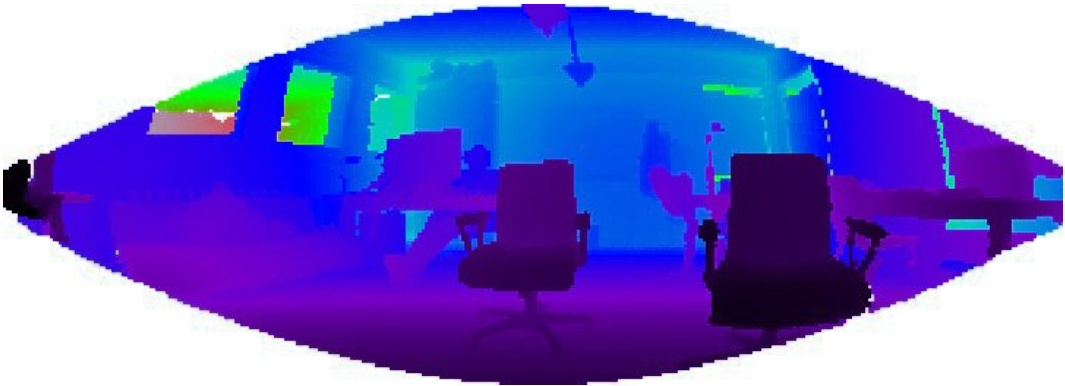
Structured light

Features similar to a ToF camera.



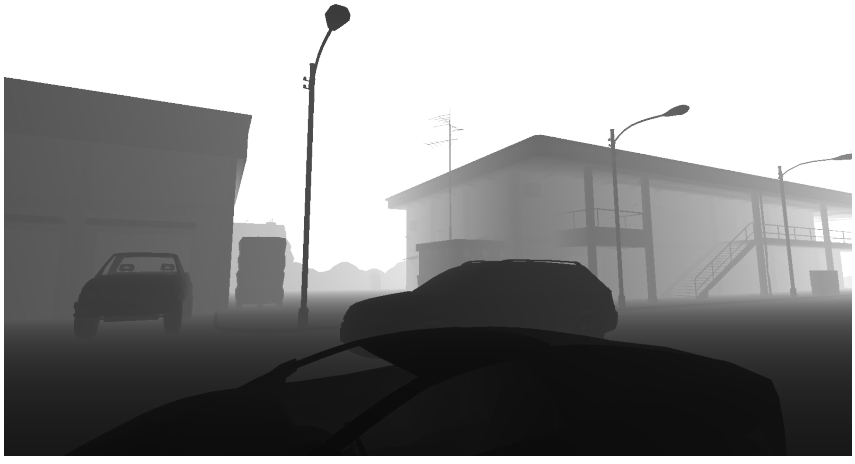
Storing depth data

2D matrix (range image, depth is color-coded).



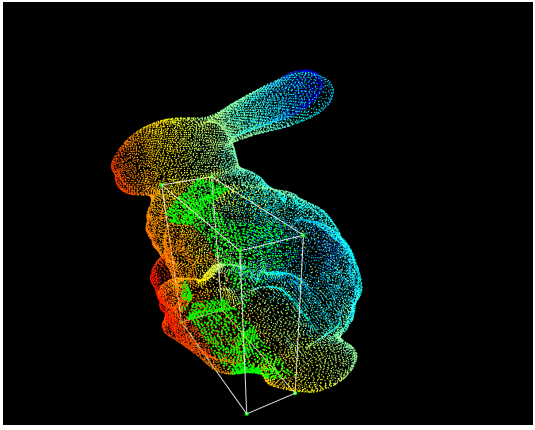
Storing depth data

2D matrix (Z buffer, grayscale)



Storing depth data

Raw 3D (point cloud).



Storing depth data

Raw 3D (textured point cloud).

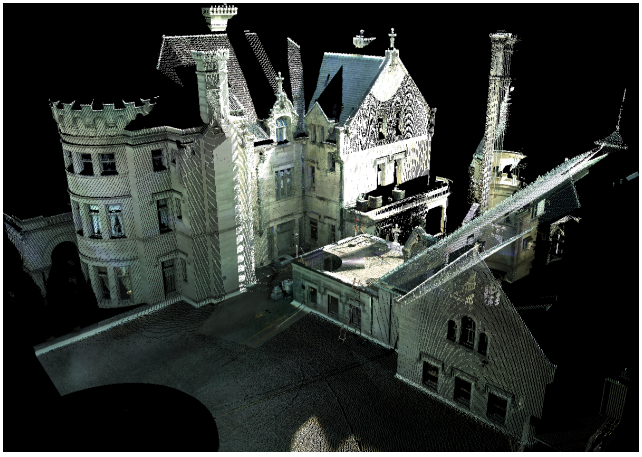
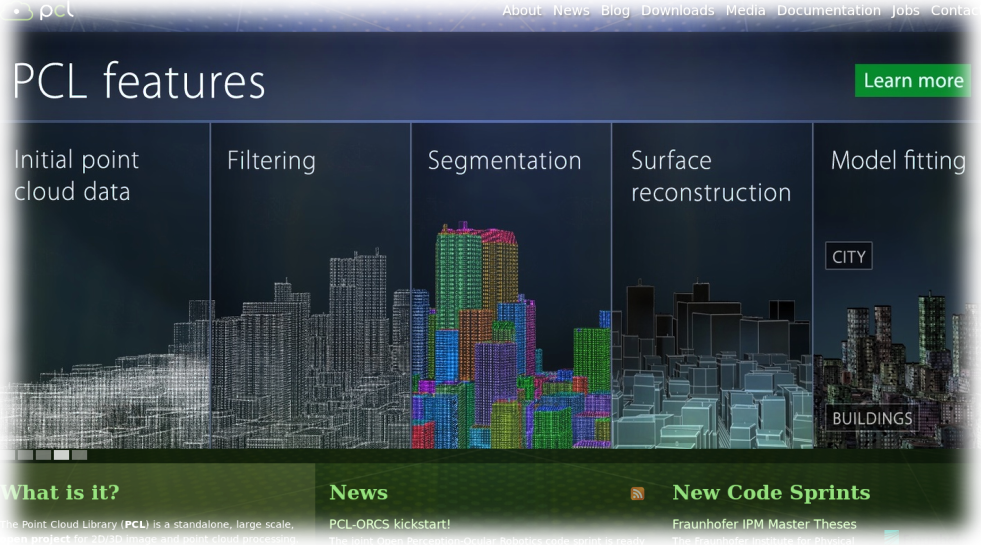


Table of Contents

PCL

- Project started March 2010 by Radu Bogdan Rusu and Steve Cousins.
- Version 1.0 released on May 2011.
- Open source, cross-platform framework.

PCL


pcl
About News Blog Downloads Media Documentation Jobs Contact
PCL features
Learn more
Initial point cloud data
Filtering
Segmentation
Surface reconstruction
Model fitting
CITY
BUILDINGS
What is it?
The Point Cloud Library (PCL) is a standalone, large scale,
open project for 2D/3D image and point cloud processing.
News
PCL-ORCS kickstart!
The joint Open Perception-Ocular Robotics code sprint is ready
New Code Sprints
Fraunhofer IPM Master Theses
The Fraunhofer Institute for Physical

PCL modules

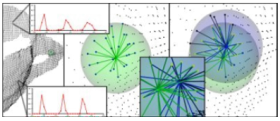
PCL is made by modular libraries:

- *Common*
- *Features*
- *Filters*
- *IO*
- *KdTree*
- *Keypoints*
- *Octree*
- *Range Image*
- *Registration*
- *Sample Consensus*
- *Search*
- *Segmentation*
- *Surface*
- *Visualization*

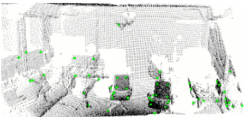
PCL modules

filters

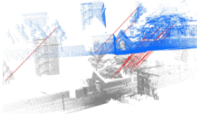
features



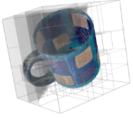
keypoints



registration



kdtree

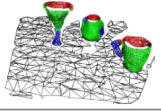


octree

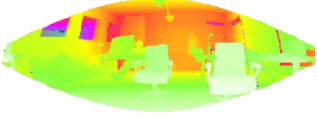
segmentation

sample_consensus

surface



range_image



io

visualization

<PointT>

```
struct PointXYZ
{
    float x;
    float y;
    float z;
    float padding;
};
```

Other types:

- *PointXYZI*
- *PointXYZRGBA*
- *PointXYZRGB*
- *InterestPoint*
- *Normal*
- *PointXYZRGBNormal*
- *PointXYZINormal*
- *PointWithRange*
- *PointWithViewpoint*
- ...

pcl::PointCloud<PointT>

```
// List of points
std::vector<PointT, Eigen::aligned_allocator <PointT>> points;

// Total number of points for unorganized clouds
uint32_t width;

// 1 for unorganized clouds
uint32_t height;

// True if there are no Inf or NaN
bool is_dense;

// Sensor position is known
Eigen::Vector4f sensor_origin_;
Eigen::Quaternionf sensor_orientation_;
```

PCL file

- .pcd files (PCD → Point Cloud Data).
- Readable plain text, or fast binary format.

Example

```
# .PCD v.7 - Point Cloud Data file format
VERSION .7
FIELDS x y z
SIZE 4 4 4
TYPE F F F
WIDTH 2
HEIGHT 1
VIEWPOINT 0 0 0 1 0 0 0
POINTS 2
DATA ascii
0.35222 -0.15188 -0.1064
-0.39741 -0.47311 0.2926
```

Table of Contents

Point cloud processing

Usually depth data taken from a sensor needs fixing or adjustments:

- Filtering
  - Outlier removal
  - Resampling (downsampling, upsampling)
- Reconstruction
  - Surface smoothing
  - Triangulation
  - Hole filling
- Registration

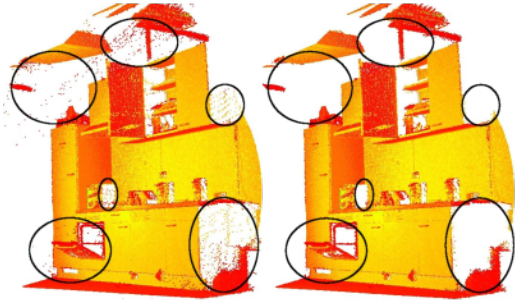
Point cloud processing

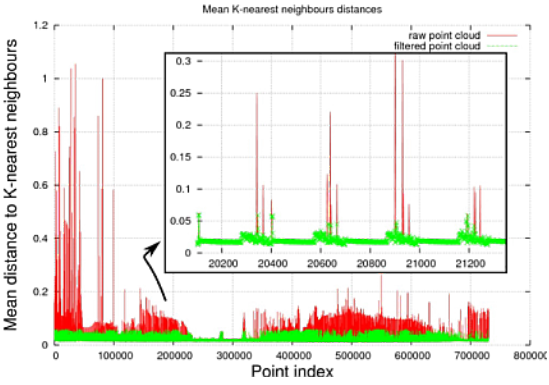
After that, we are ready to work with it:

- Decomposition
  - Kd-tree
  - Octree (voxelization)
  - Box decomposition tree (bd-tree)
- Feature estimation
  - Normals
  - Surface curvature
- Segmentation (clustering)
- Keypoint finding
- Object recognition...

Filtering

Noise removal: trim isolated (outliers) and incorrect points from the dataset:





Filtering

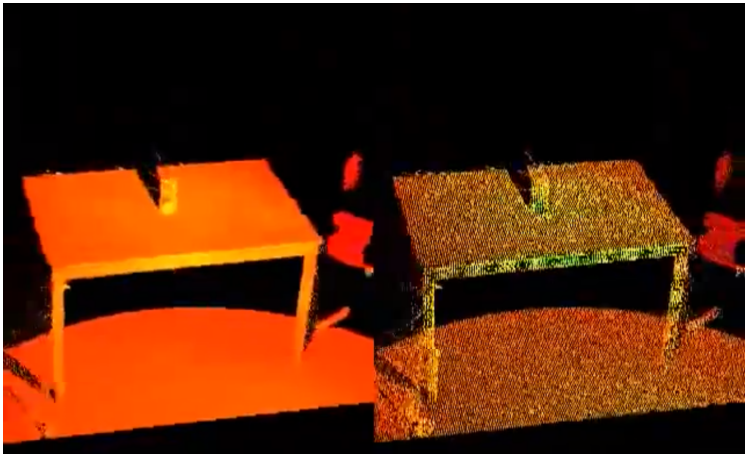
Gaussian distribution for outlier removal (k closest neighbors):

$$P^* = \left\{ p_q^* \in P | \ (\mu_k - \alpha \cdot \sigma_k) \leq \overline{d}^* \leq (\mu_k + \alpha \cdot \sigma_k) \right\}$$

P^*: remaining point cloud after filtering
p_q^*: point being computed
P: entire cloud
μ_k: mean of the distribution
σ_k: deviation of the distribution
$\overline{d}^*$: mean distance of the point to its neighbors
α: desired density restrictiveness factor

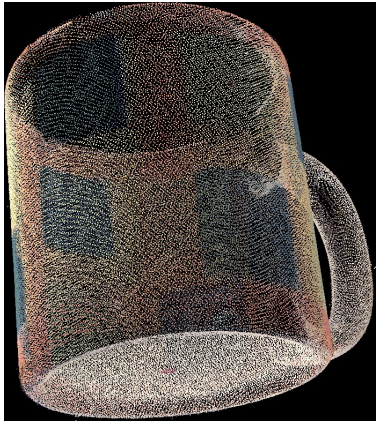
Downsampling

Reducing the number of points, for less complexity. Example: voxelized grid.



Upsampling

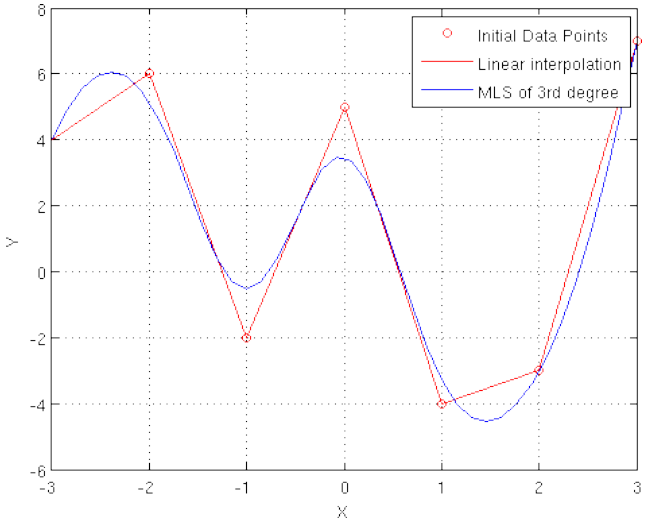
Adding more points to increase detail and resolution.



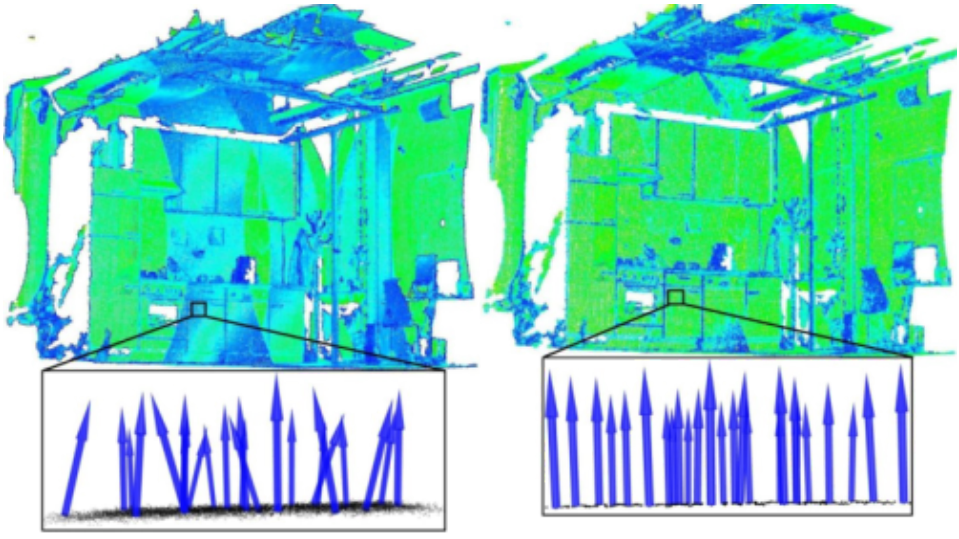
Surface smoothing

Fixes badly registered clouds, double walls, measurement errors...

Moving Least Squares (MLS): higher order polynomial interpolations between points.



Surface smoothing



Hole filling

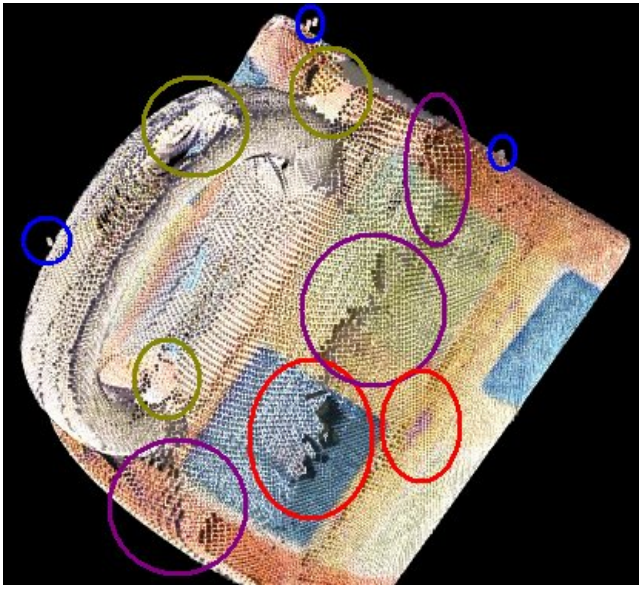
Extra points are added to areas where a hole is detected.

If $\Theta = \{\theta_1 \dots \theta_n\}$ is the list of sorted angles between the lines formed by two points p_{k_1} and p_{k_2} in the neighborhood P^k with the query point p_q, then p_q is a point located on the boundary of a hole, given:

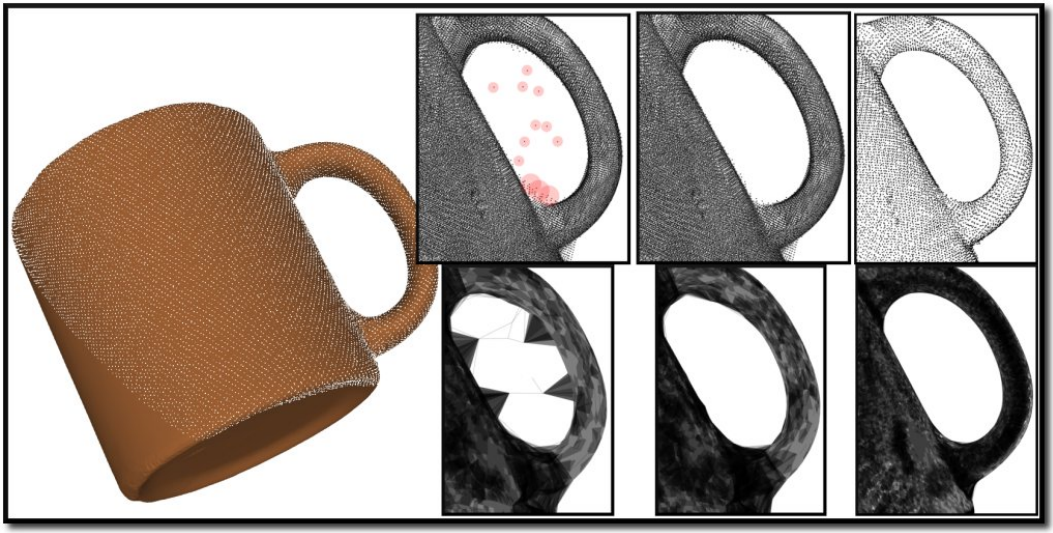
$$max(\alpha = \theta_{i+1} - \theta_i \geq \alpha_{th})$$

α_{th}: maximum given threshold angle

Hole filling



Reconstruction

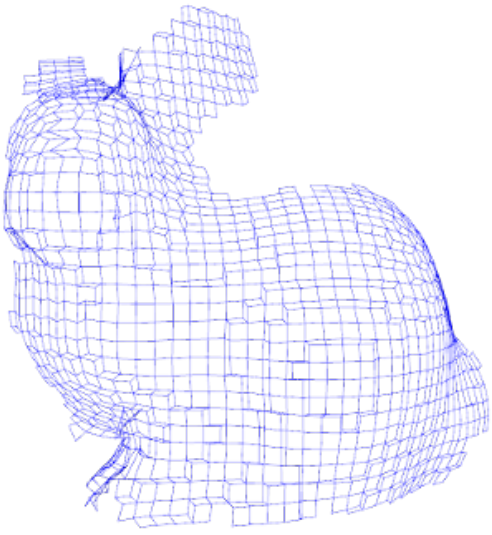


Triangulation

Turning point cloud into a polygon-made model.

Points are connected with points in their local neighborhood. This neighborhood is estimated from the point's normal plane.

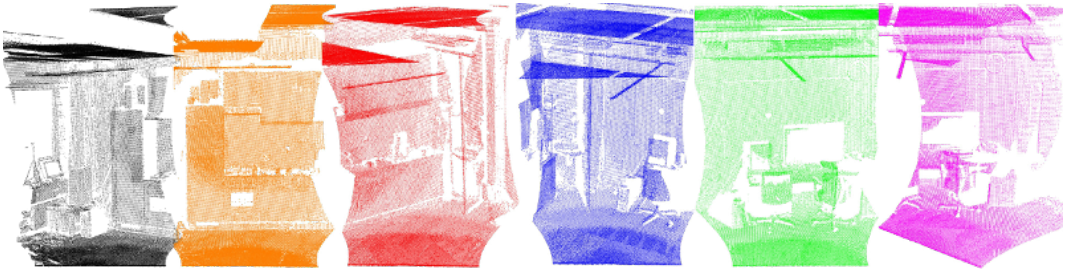
Triangulation



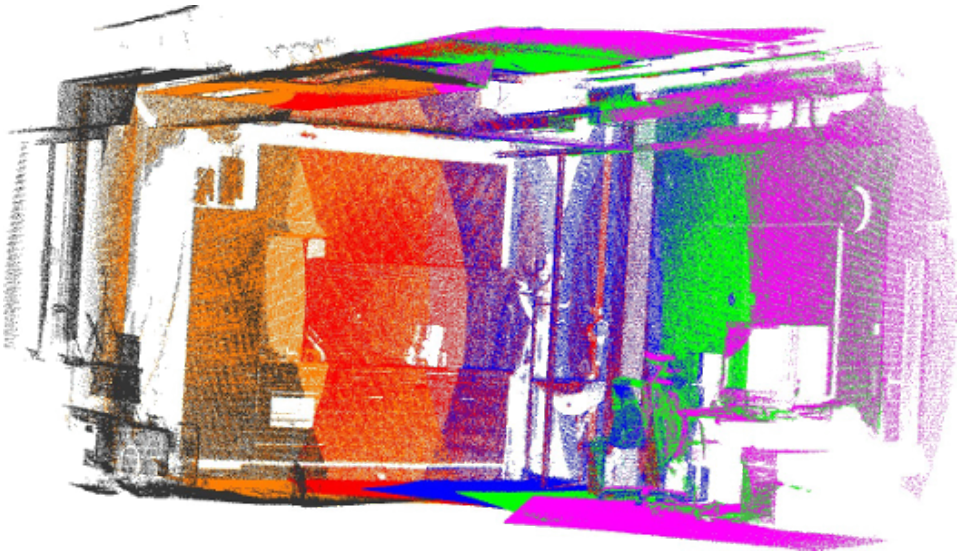
Registration

Consistently align various 3D point cloud data views into a complete model.

Solution: find point correspondences in the given input datasets, and estimate rigid transformations that can rotate and translate one into other.



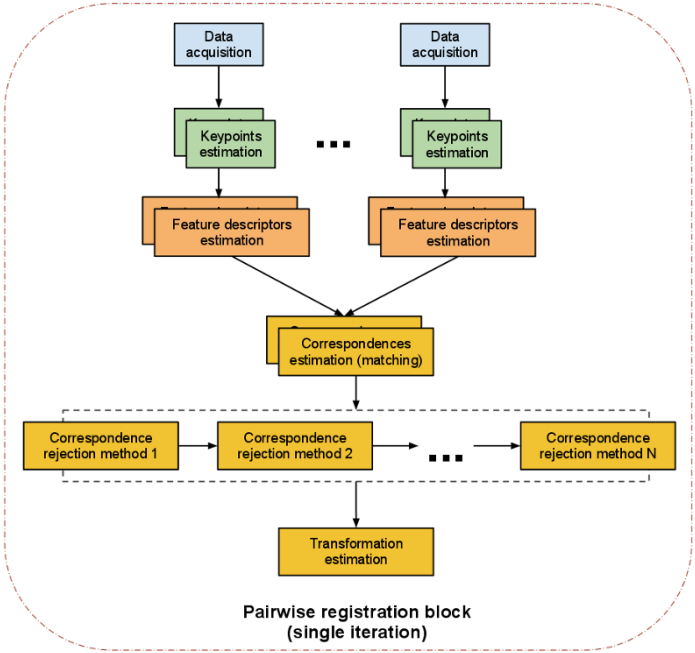
Registration



Registration

The problem is called *pairwise registration*.

It outputs a rigid transformation matrix (4x4) with the rotation and translation to be applied to one of the datasets (source), so it is perfectly aligned with the other (target).


Data acquisition
Data acquisition
Keypoints estimation
Keypoints estimation
Feature descriptors estimation
Feature descriptors estimation
Correspondences estimation (matching)
Correspondence rejection method 1
Correspondence rejection method 2
Correspondence rejection method N
Transformation estimation
Pairwise registration block (single iteration)

ICP

Iterative Closest Point (ICP) is an algorithm that minimizes the difference between two point clouds.

Steps:

1. Associate points by the nearest neighbor criteria.
2. Estimate transformation parameters using a mean square cost function.
3. Transform the points using the estimated parameters.
4. Iterate (re-associate the points and so on).

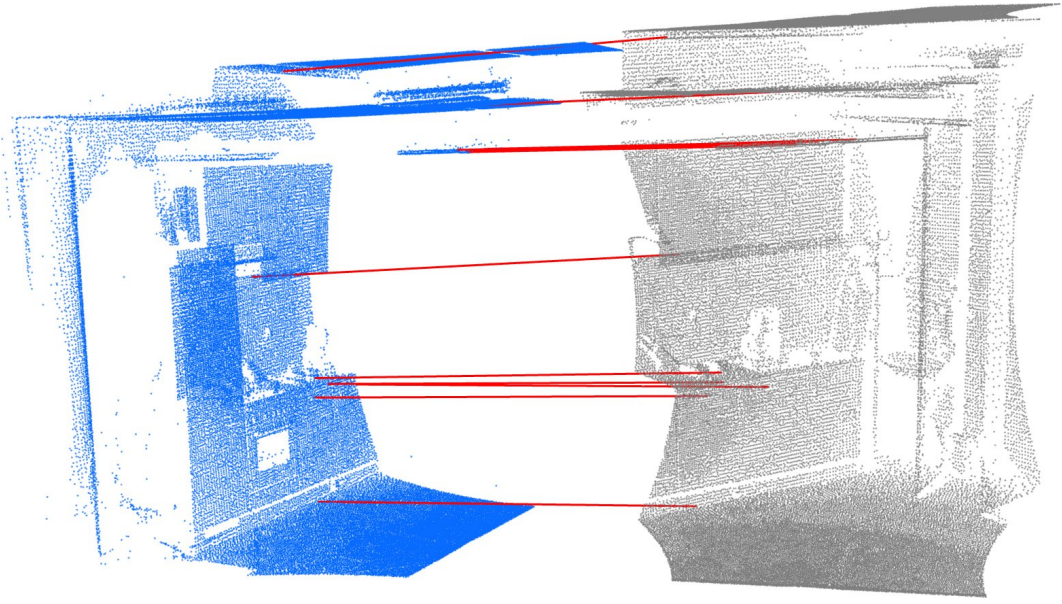
Problems: computational cost with many iterations.

Registration

Other options:

- PFH (Point Feature Histogram): seek keypoints and match them.
- NDT (Normal Distributions Transform): for large clouds, over 100,000 points. Uses standard optimization techniques applied to statistical models of 3D points.

PFH



Decomposition

Storing the cloud in a way that makes searching efficient: k nearest neighbors, approximate nearest neighbor, neighbors within radius...

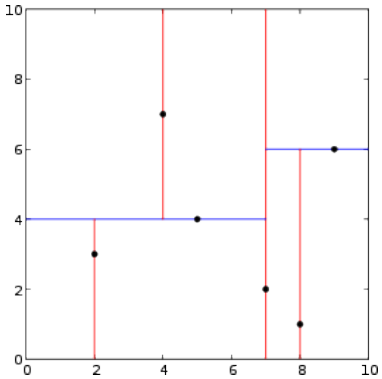
Methods:

1. K-dimensional tree (k-d tree)
2. Octree
3. Box decomposition tree (bd-tree)

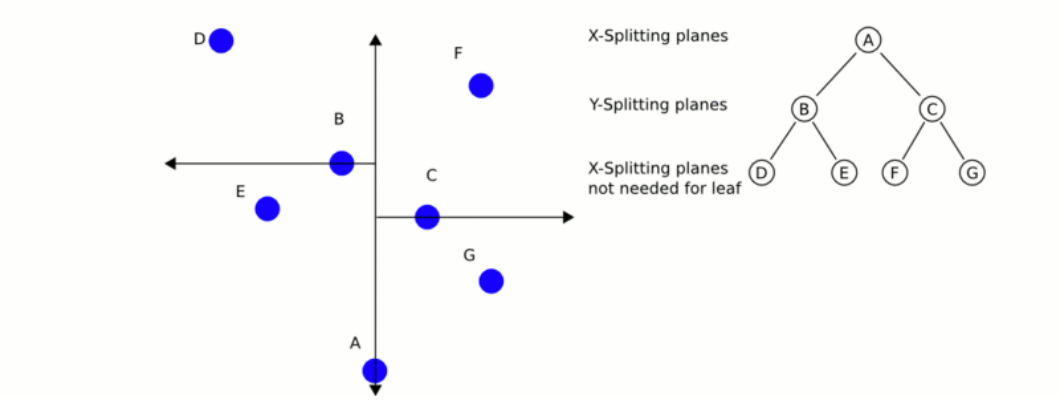
K-d tree

Binary tree that stores a set of k-dimensional points enabling efficient range searches.

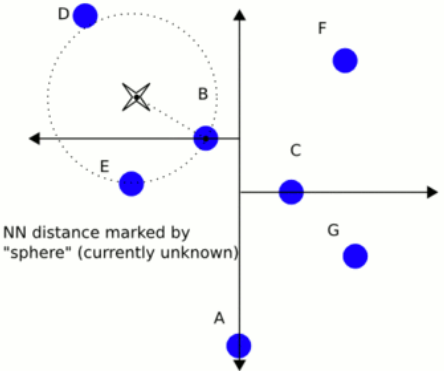
Each level splits all children on a specific dimension. At the root all children are split based on the first dimension (left subtree if lesser, right if greater). Each level down divides on the next dimension, returning to the first one when exhausted.

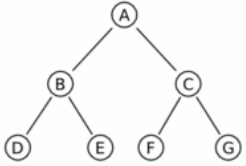


K-d tree

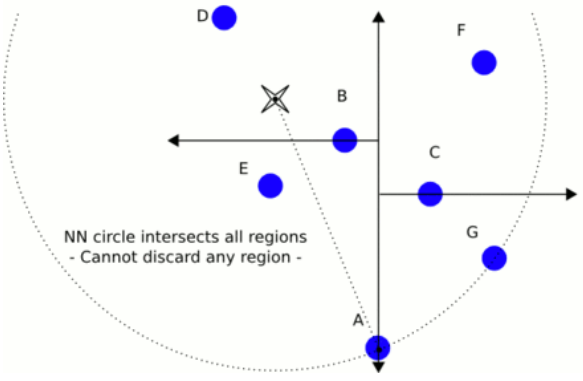

D
F
B
C
E
G
A
X-Splitting planes
Y-Splitting planes
X-Splitting planes
not needed for leaf
A
B
C
D
E
F
G

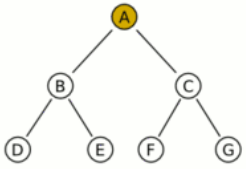
K-d tree


D
F
B
C
E
G
A
NN distance marked by
"sphere" (currently unknown)


A
B
C
D
E
F
G

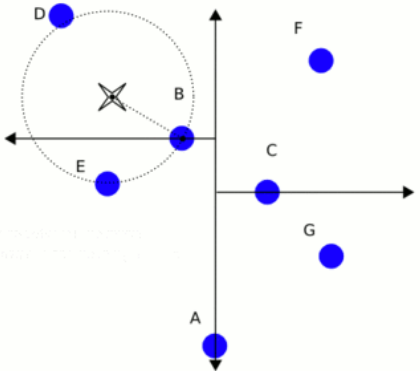
K-d tree

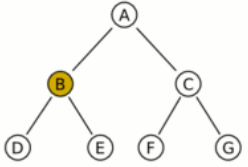




Start at A, then proceed in depth-first search (maintain a stack of parent-nodes if using a singly-linked tree). Set best estimate to A's distance Then examine left child node

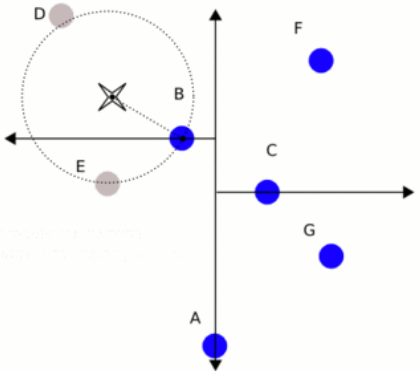
K-d tree

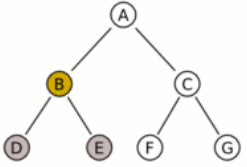




Calculate B's distance and
compare against best estimate
- It is smaller distance, so update
best estimate. Examine children (left then right)

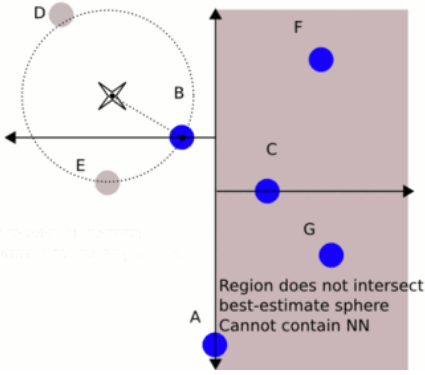
K-d tree

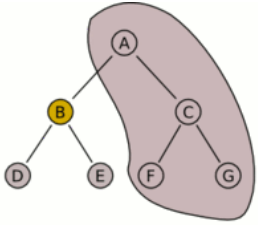




D & E Discarded as B
(already visited) is closer.
B is the best estimate for B's sub-branch
Proceed back to parent node

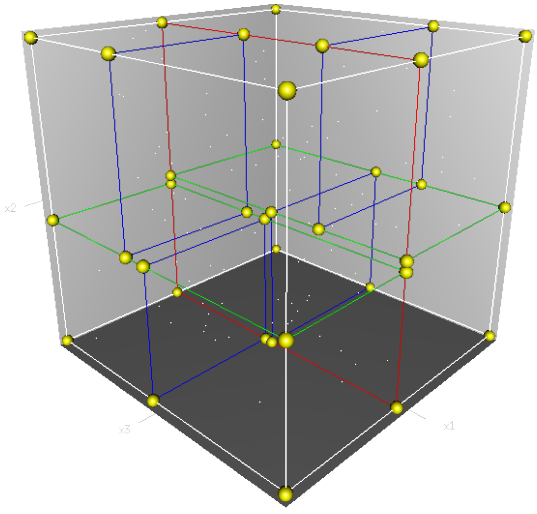
K-d tree

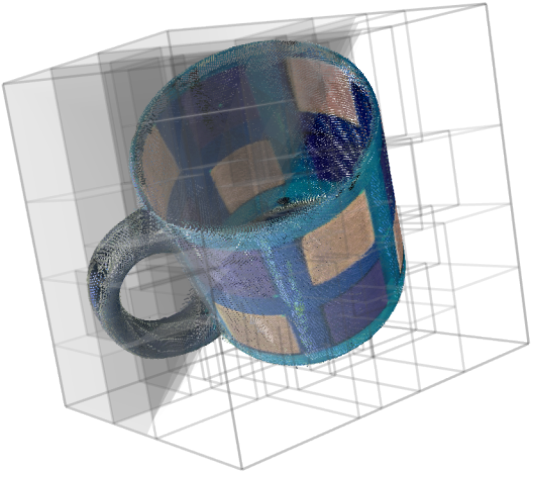




A's children have all been searched,
B is the best estimate for entire tree

K-d tree





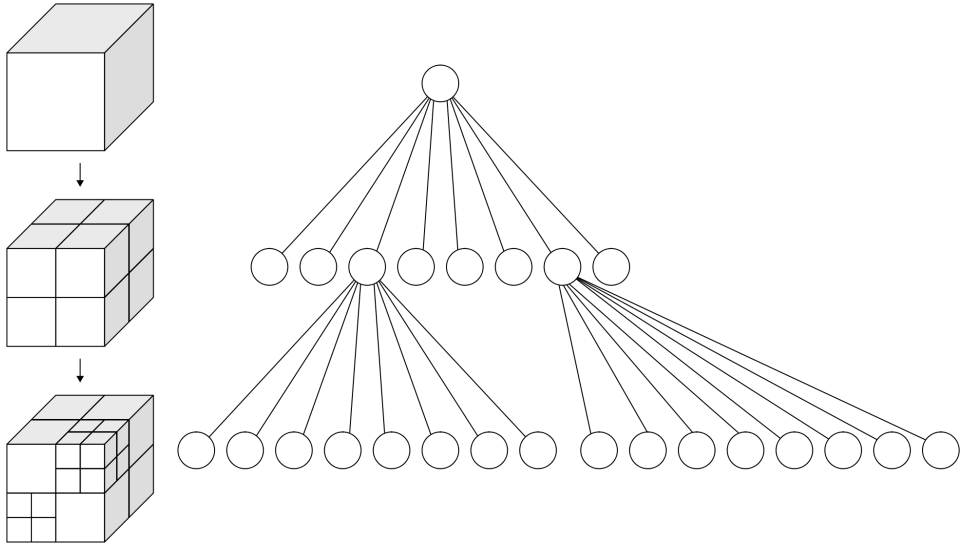
Octree

Hierarchical tree data structure useful for searches, compression, downsampling...

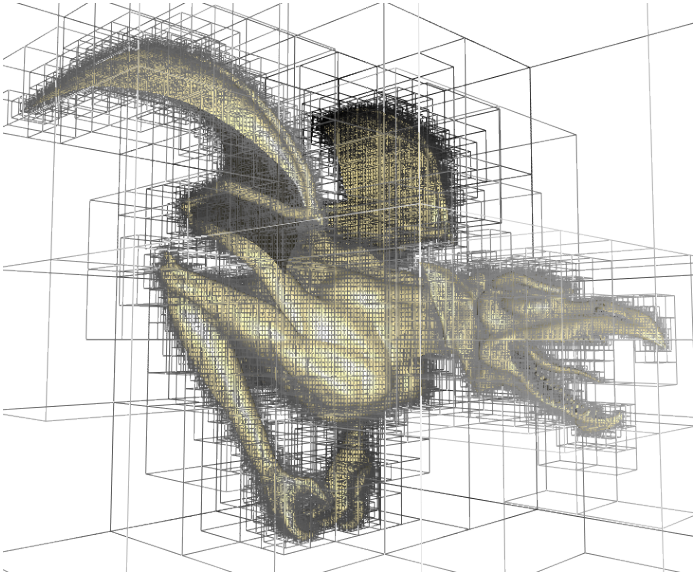
Each octree node (voxel) has either eight children or no children. The root node describes a cubic bounding box which encapsulates all points.

At every level, it becomes subdivided by a factor of 2 which results in an increased voxel resolution.

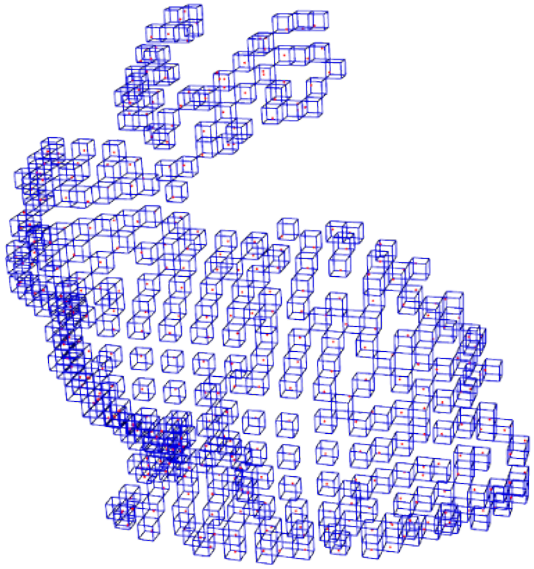
Octree



Octree



Octree

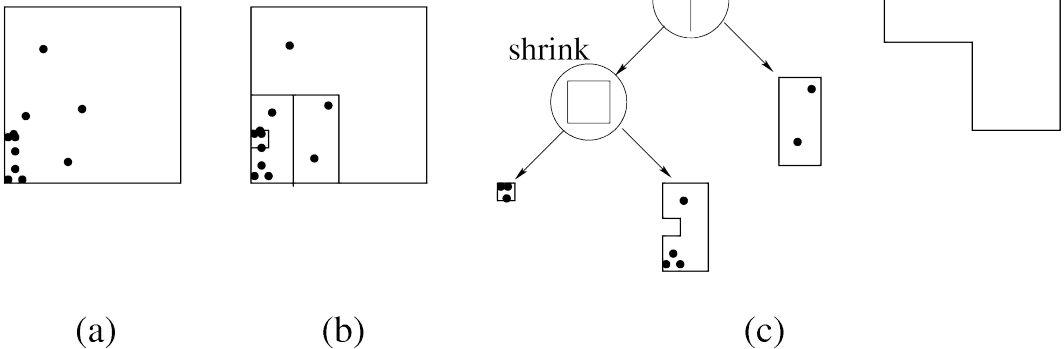


Bd-tree

Balanced box decomposition tree (bd-tree): similar to a kd-tree, but the aspect ratio of the tree edges is assured to be bounded by a constant.

They are made with *shrinks* and *splits*.

Bd-tree



Feature estimation

A *feature* describes geometrical patterns based on the information around a point.

The space selected around the query point p_q is referred to as the k-neighborhood.

Most used features:

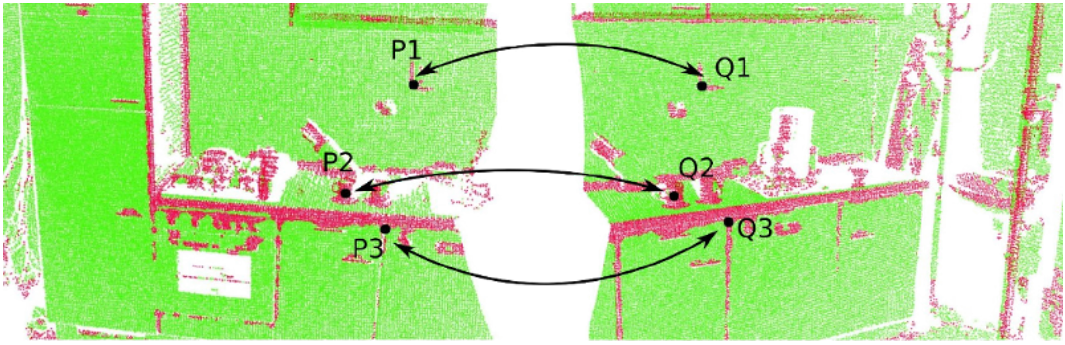
- Surface curvature at that point.
- Normal at that point.

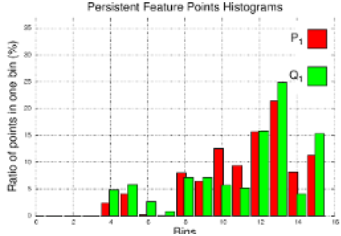
Feature estimation

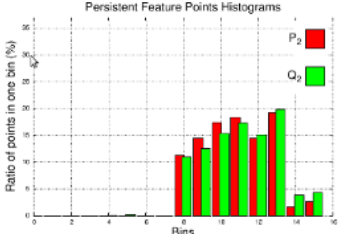
A good point feature is:

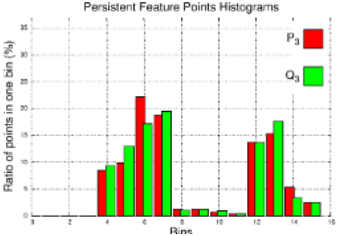
1. Similar for points residing on the same or similar surfaces.
2. Robust in case of:
   - Rigid transformations (rotation, translation)
   - Varying sampling density
   - Noise

Feature estimation



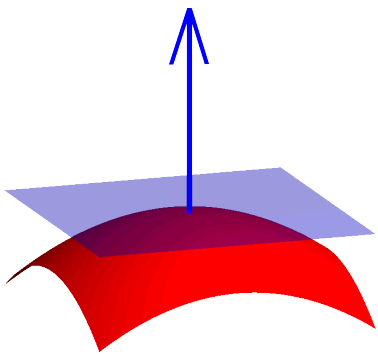


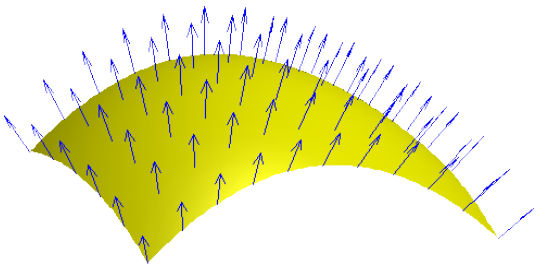




Surface normals

Normal: vector perpendicular to the surface at that point.





Surface normals

The normal to a point on the surface is approximated by the normal of a plane tangent to the surface: a least-square plane fitting estimation problem.

We use Principal Component Analysis (PCA): analysis of the eigenvectors and eigenvalues of a covariance matrix created from the nearest neighbors of the point.

Surface normals

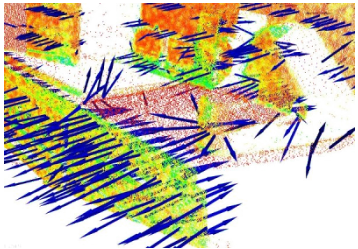
For each point p_i we assemble the covariance matrix C as:

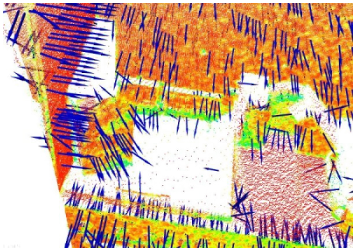
$$C = \frac{1}{k} \sum_{i=1}^{k} \cdot (p_i - \overline{p}) \cdot (p_i - \overline{p})^T, \; C \cdot \vec{v_j} = \lambda_j \cdot \vec{v_j}, \; j \in \{0, 1, 2\}$$

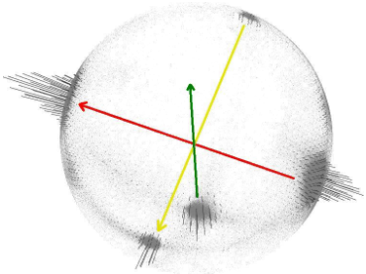
k: number of points in the neighborhood of p_i
$\overline{p}$: 3D centroid of the nearest neighbors
λ_j: j-th eigenvalue if the covariance matrix
$\vec{v_j}$: j-th eigenvector

Surface normals

The Extended Gaussian Image (EGI), a.k.a. *normal sphere*, describes the orientation of all normals from the point cloud.

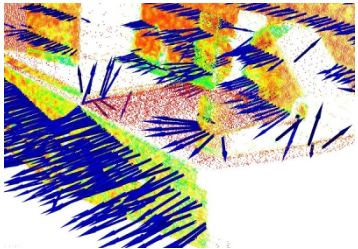


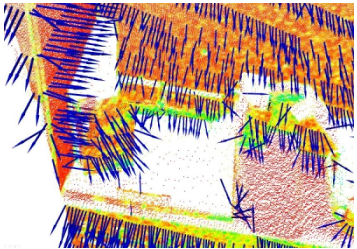


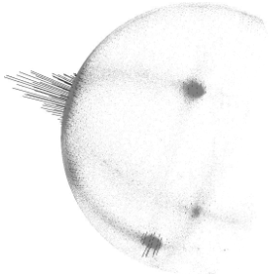


Surface normals

The Extended Gaussian Image (EGI), a.k.a. *normal sphere*, describes the orientation of all normals from the point cloud.







Descriptors

Surface normals and curvature estimates are extremely fast and easy to compute, but capture little detail of the geometry of a point's k-neighborhood.

We need better 3D feature *descriptors* that more uniquely identify each point:

- Point Feature Histogram (PFH)
  - Fast Point Feature Histogram (FPFH)
- Viewpoint Feature Histogram (VFH)
- Normal Aligned Radial Features (NARF)

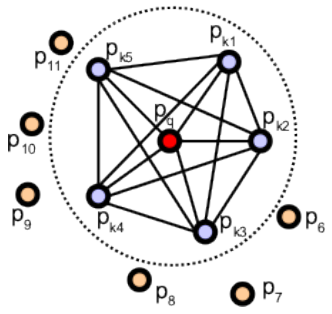
PFH

The Point Feature Histogram encodes the mean curvature around the point using a multi-dimensional histogram of values. It is invariant to the 6D pose and robust with different sampling densities or noise levels.

The PFH samples the interactions between the directions of the estimated normals. The better the normal estimation, the better PFH is.

PFH

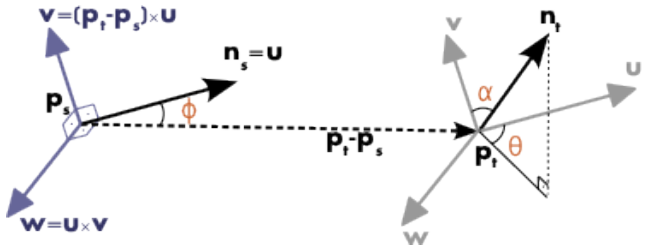
For every query point p_q we have an influence region of radius r. All k neighbors inside are interconnected. The relationship between each pair will be computed.



Hence, the computational complexity is $O(k^2)$ for each point.

PFH

For every pair p_i, p_j we compute the difference of the normals n_i, n_j. We define a fixed coordinate frame (uvw) at one of the points:



$$u = n_i \qquad v = u \times \frac{(p_j - p_i)}{||p_j - p_i||_2} \qquad w = u \times v$$

Using the *uvw* frame, the difference between the normals n_i, n_j can be expressed as a set of 3 angular features:

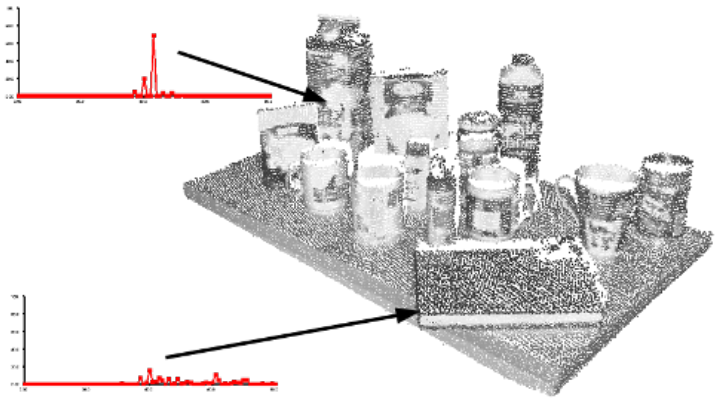
$$\alpha = v \cdot n_j \qquad \text{ø} = u \cdot \frac{(p_j - p_i)}{d} \qquad \theta = \arctan\left(w \cdot n_j, u \cdot n_j\right)$$

d: Euclidean distance between p_i and p_j ($d = ||p_j - p_i||_2$)

The quadruplet $(\alpha, \text{ø}, \theta, d)$ is computed for each pair of points in the k-neighborhood, reducing 12 values (x,y,z and normal for 2 points) to these 4.

PFH

Finally, the set of all quadruplets is binned into a histogram. The binning process divides each features's value range into b subdivisions, and counts the number of occurrences in each subinterval.



FPFH

The computational complexity of PFH for a cloud with n points is $O(nk^2)$. To run real time we need something simpler: the Fast Point Feature Histogram (FPFH).

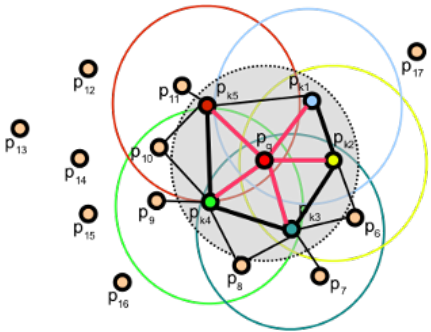
Differences:

1. FPFH only pairs the neighbors with the point, not with each other.
2. FPFH can include points outside the r radius (but mostly $2r$ away).
3. A separate histogram is created for each dimension. They are then concatenated.

FPFH

The algorithm has two runs:

1. For each point a set (α, ϕ, θ) is calculated with its k neighbors. This is called the Simplified Point Feature Histogram (SPFH).



2. The FPFH of each point is determined based on the SPFH of its neighbors.

$$FPFH(p_q) = SPFH(P_q) + \frac{1}{k}\sum_{i=1}^{k}\frac{1}{\omega_k} \cdot SPFH(p_k)$$

ω_k: weight representing the distance between p_q and p_k in some given metric space

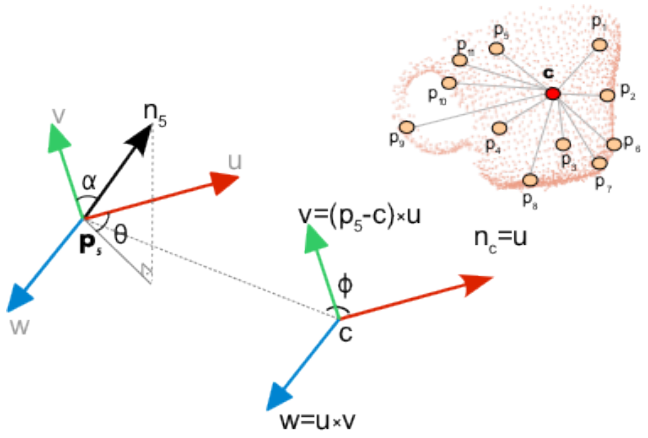
The weighting scheme allows to include more detail about the surrounding surface.

VFH

The Viewpoint Feature Histogram (VFH) is a technique based on FPFH, adding viewpoint variance while retaining invariance to scale. Designed for object recognition and 6DOF pose estimation.

VFH

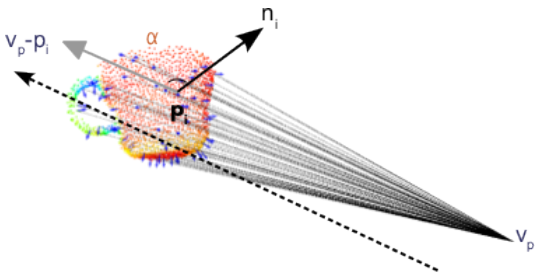
The FPFH is now estimated for the entire object cluster. Additional statistics are computed between the viewpoint direction and the normals estimated at each point. We mix the viewpoint direction into the relative normal angle calculation in the FPFH.



VFH

The first *component* is computed with a histogram of angles between the central viewpoint direction translated to each normal.

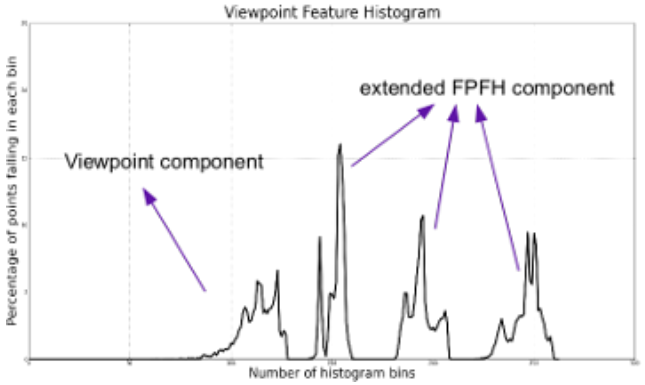
The second measures relative pan, tilt and yaw angles like in FPFH, but between the viewpoint direction at the central point and each of the normals on the surface.



VFH

The result is the VFH feature, with two parts:

1. A viewpoint direction component.
2. A surface shape component comprised of an extended FPFH.



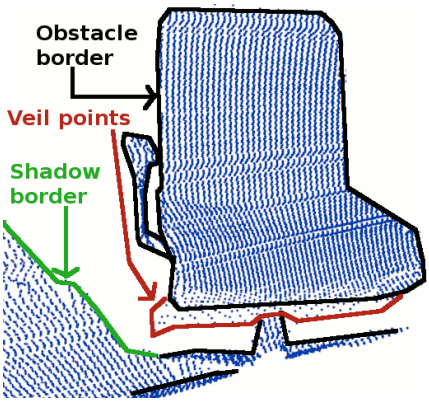
NARF

A *keypoint* is a point that is stable, distinctive, and can be identified using a well-defined detection criterion.

NARF (Normal Aligned Radial Feature) is a method for finding and storing such keypoints in a *descriptor* for each object.

NARF

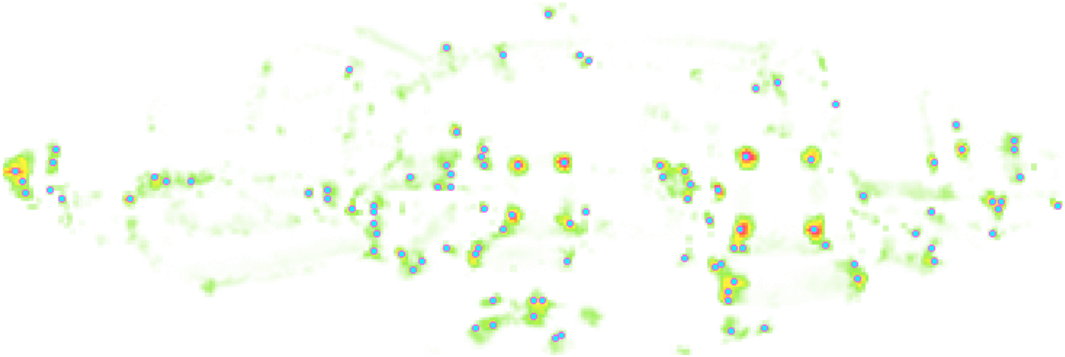
NARF considers the borders of the objects, identified by transitions from foreground to background. Something trivial with depth data.



NARF

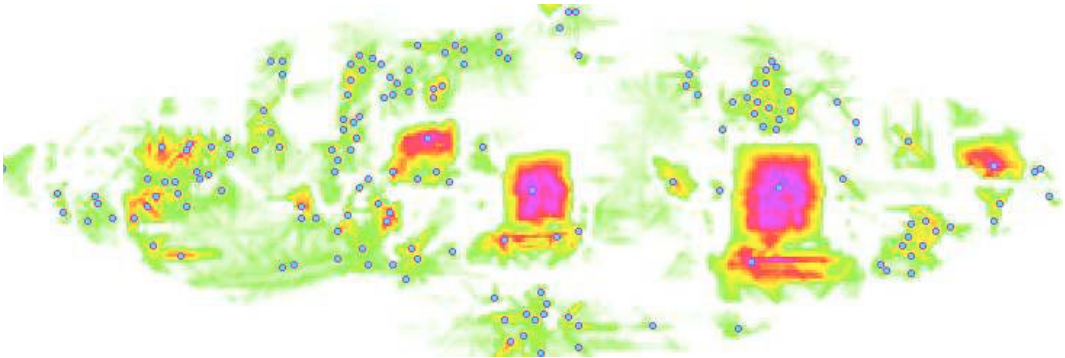
NARF

Support size of 20cm.

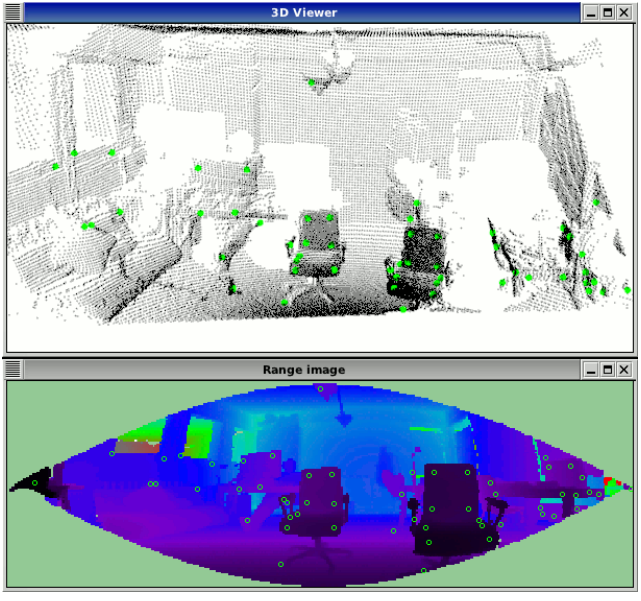


NARF

Support size of 1m.



NARF



Segmentation

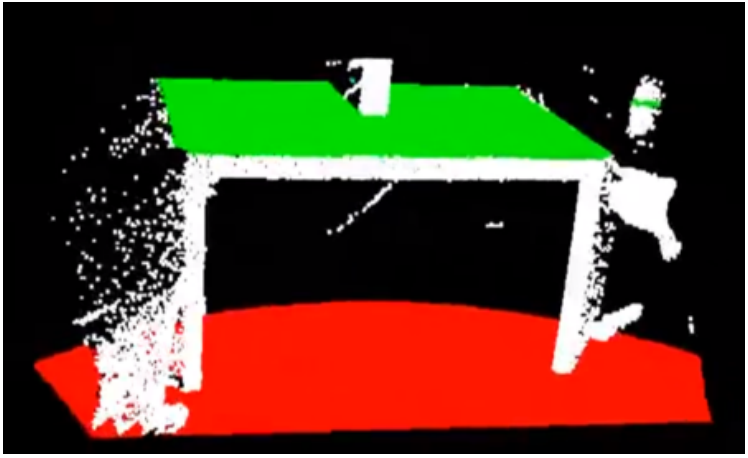
The process of dividing a point cloud into *clusters* (groups of points considered individual objects or regions).

Methods:

- Plane model segmentation
- Cylinder model segmentation
- Euclidean Cluster Extraction
  - Conditional Euclidean Clustering
- Region growing segmentation
  - Color-based region growing segmentation
- Min-Cut based segmentation
- Difference of Normals based segmentation

Plane model segmentation

Find all the points within a point cloud that support a plane model.



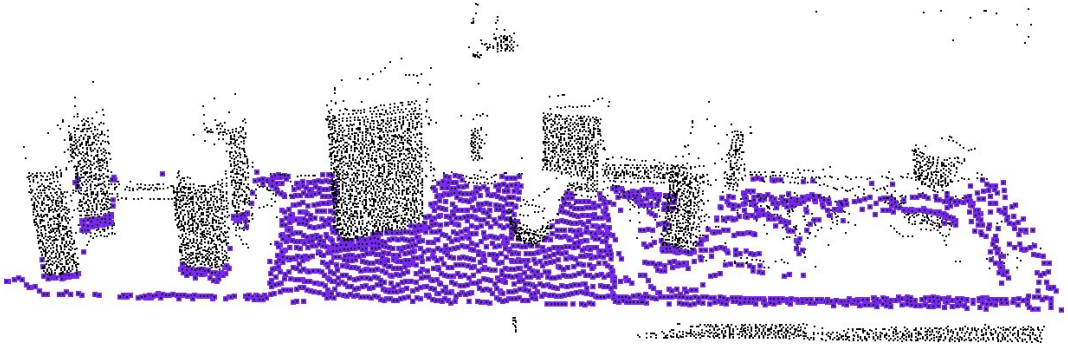
Plane model segmentation

Steps:

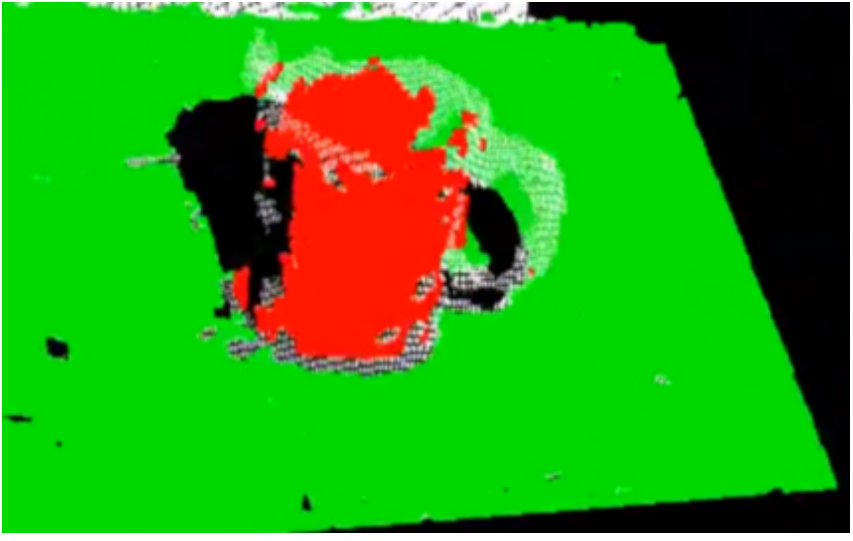
1. Randomly select three non-collinear points $\{p_i, p_j, p_k\}$ from P.
2. Compute the model coefficients from the points $(ax + by + cz + d = 0)$
3. Compute the distances from all $p \in P$ to the plane model (a, b, c, d)
4. Count the number of points whose distance d checks $0 \leq |d| \leq |d_t|$
   d_t: distance threshold

   At the end, the set with most points is selected as the support for the best planar model found.

Plane model segmentation



Cylinder model segmentation

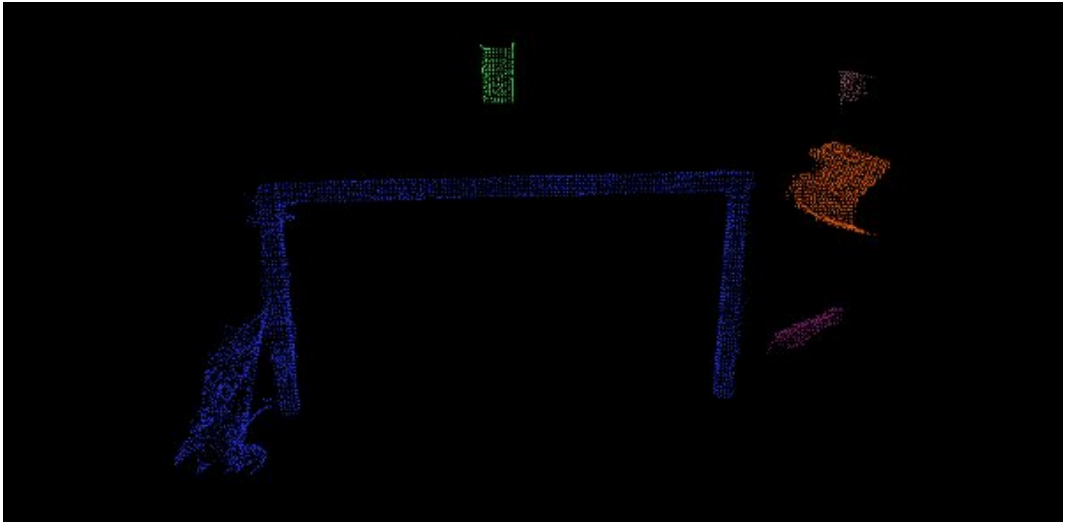


Euclidean Cluster Extraction

Uses a flood fill algorithm, with a kd-tree to make nearest neighbor searches:

1. Initialize an empty list of clusters, C, and a queue of points to be checked, Q.
2. For every point $p_i \in P$ do:
    1. Add p_i to Q.
    2. For every $p_i \in Q$:
        1. Search for all neighbors of p_i, P_k^i, in a sphere with radius $r < d_{th}$.
        2. For every neighbor $p_k^i \in P_k^i$, add it to Q if not already processed.
    3. Add Q to the list of clusters C, reset Q to empty.

Euclidean Cluster Extraction

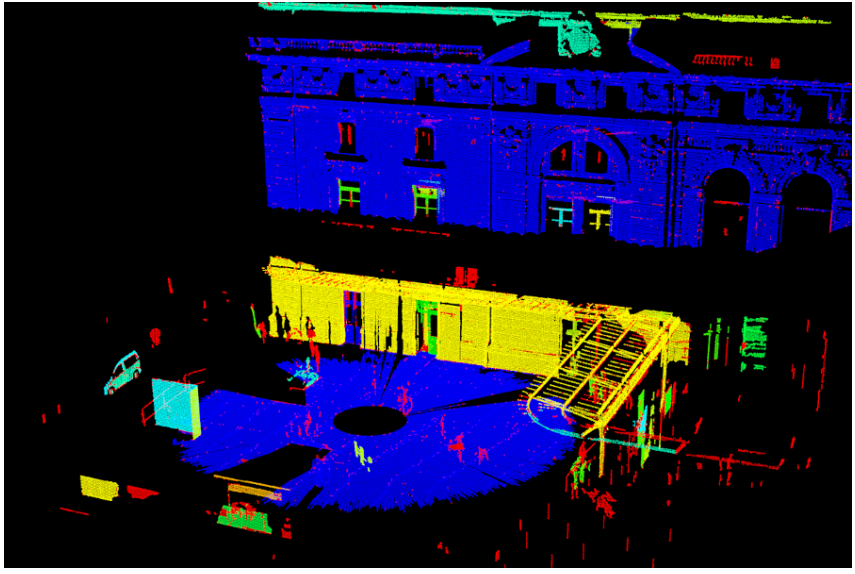


Conditional Euclidean Clustering

Based on the previous, but with a custom condition that all points need to meet to be included in the current cluster. Can also impose size constraints on clusters.

Requires more computational power.

Conditional Euclidean Clustering



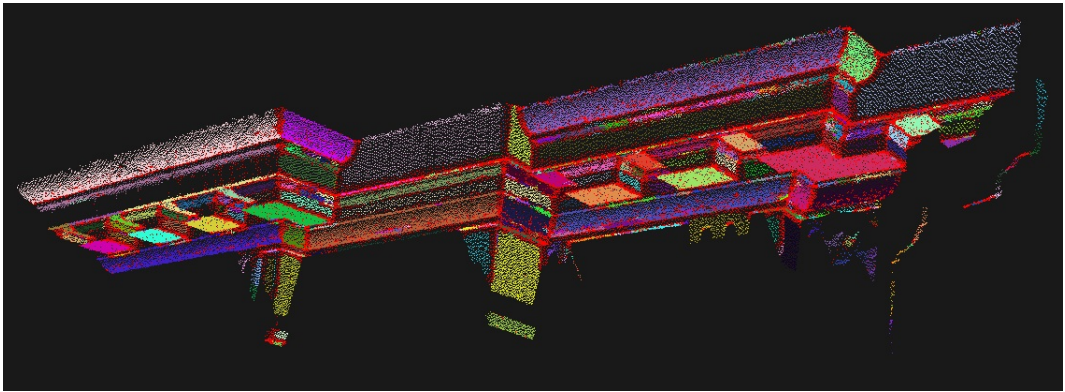
Region growing segmentation

A greedy-like, region-growing, flood-filling algorithm like Euclidean segmentation.

Merges points that not only are close, but also share a similar smoothness constraint (part of the same smooth surface). It compares the angles between the points normals:

1. While there are points left in P, pick the one with the lesser curvature.
2. Add that point p_i to the list of seeds, S.
3. For every neighbor p_k of p_i:
    1. Test the angles between the normals. If $< n_{th}$, add to region.
    2. Test the curvature. If $< c_{th}$, add to seeds.
4. Remove current seed from S. If S empties, region is finished.

Region growing segmentation

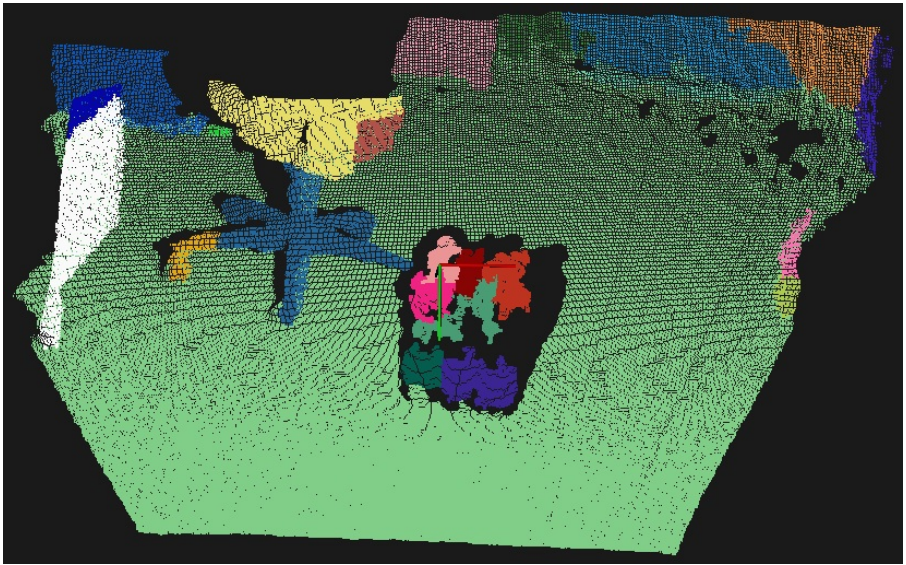


Color-based region growing segmentation

Perform a normal region growing. After that, merge clouds with similar colors together.

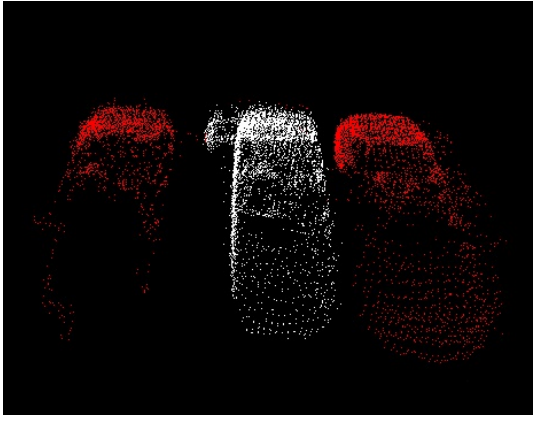
Optionally, replace normal testing of the original algorithm with color testing.

Color-based region growing segmentation



Min-Cut based segmentation

Binary segmentation. From an object's center and radius, divides the cloud in two parts: points that belong to the object and points in the background.



Min-Cut based segmentation

The segmentation is performed constructing a graph:

1. Every point will be a vertex, plus two more vertices called *source* and *sink*.
2. Every point vertex has edges connecting it to its neighbors in the cloud. Also, every point vertex is connected with source and the sink.
3. Assign weights to each edge:
    1. A *smooth cost* weight for edges between cloud points.
    2. A *data weight*, consisting of foreground and background penalties.
        1. Foreground for edges connecting points to the source. User defined.
        2. Background for edges connecting points to the sink.
4. Search a minimum cut. Based on it, segment cloud.

Min-Cut based segmentation

$$smoothCost = e^{-\left(\frac{dist}{\sigma}\right)^2}$$

dist: distance between points

$$backgroundPenalty = \left(\frac{distanceToCenter}{radius}\right)$$

radius: radius of the object, input parameter for algorithm

$$distanceToCenter = \sqrt{(x - centerX)^2 + (y - centerY)^2}$$

Difference of Normals based segmentation

Scale-based segmentation of unorganized point clouds. Finds points that belong within the scale parameters given:

1. Estimate the normal for every point, using a large support radius (r_l).
2. Estimate the normal for every point, using a small support radius (r_s).
3. For every point, apply the Difference of Normals operator.
4. Filter the resulting vector field to isolate points belonging to scale of interest.

Difference of Normals based segmentation

Two unit point normals, $\hat{n}(p, r_l)$ and $\hat{n}(p, r_s)$, are estimated with different radii.

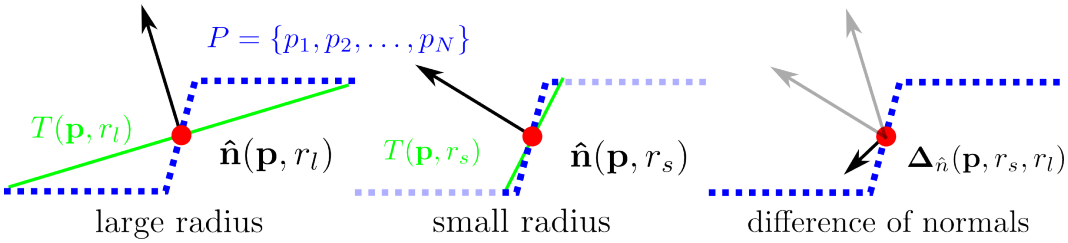
Difference of Normals operator:

$$\Delta\hat{n}(p, r_s, r_l) = \frac{\hat{n}(p, r_s) - \hat{n}(p, r_l)}{2}$$

where $r_s, r_l \in \mathbb{R}$, $r_s < r_l$, and $\hat{n}(p, r)$ is the surface normal estimate at point p, given the support radius r.

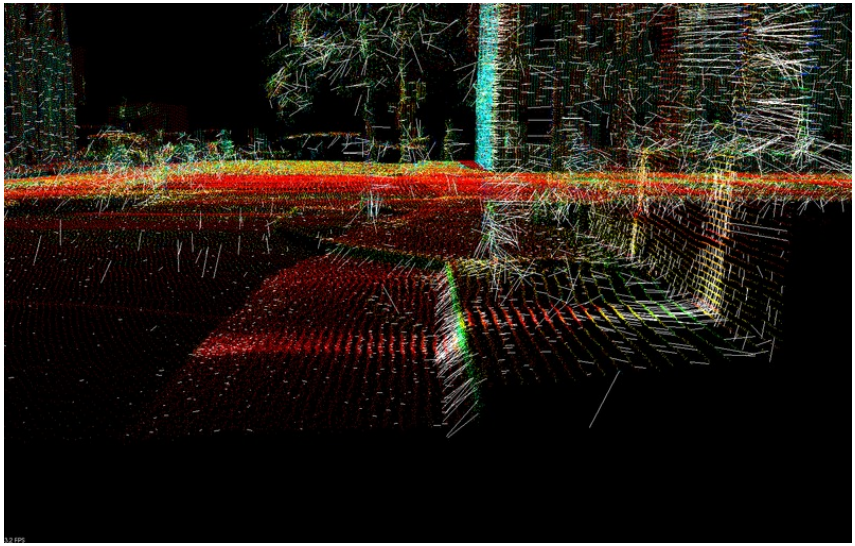
The result of the operator is a normalized vector field.

Difference of Normals based segmentation

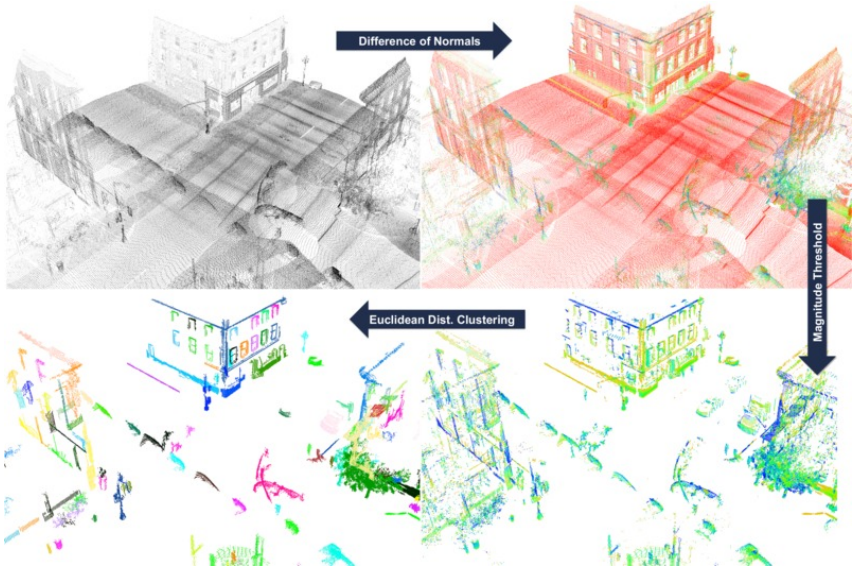


Normals estimated with a small support radius r_s are affected by small-scale surface changes, and noise.

Difference of Normals based segmentation



Difference of Normals based segmentation



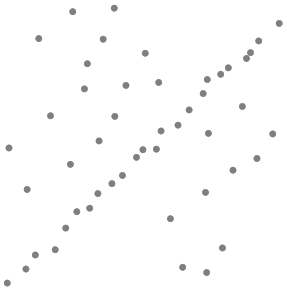
Difference of Normals based segmentation

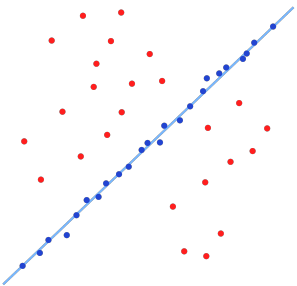
Table of Contents

RANSAC

RANdom SAmple Consensus (RANSAC) is an iterative method to estimate parameters of a mathematical model from a set of observed data which contains outliers.

It is non-deterministic: the more iterations, the higher the probability of having a good result.





RANSAC

The input to RANSAC is:

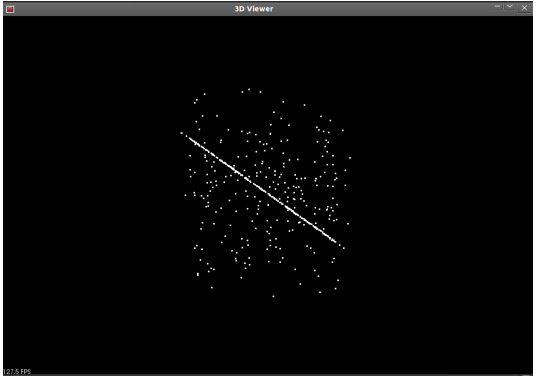
- A set of observed values.
- A parameterized model which can be fitted to them.
- Some confidence parameters.

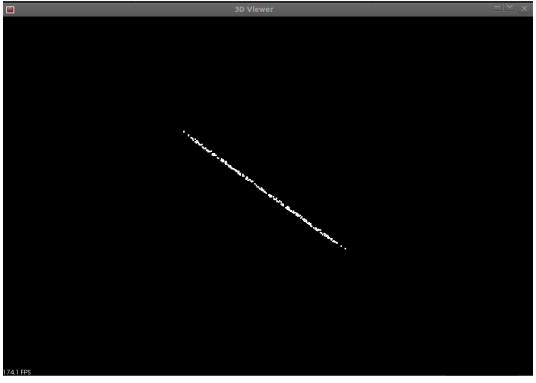
Every iteration, RANSAC selects a random subset of data, and makes the hyphothesis that they are inliers. Then it tests it:

1. The parameters of a model are fitted to these "inliers".
2. All other data is tested against this new model, and added as inlier if it fits.
3. The model is considered good if many points are considered inliers.
4. The model is fitted again for all current inliers, not just the initial ones.
5. The model is evaluated with the one we are seeking.

RANSAC

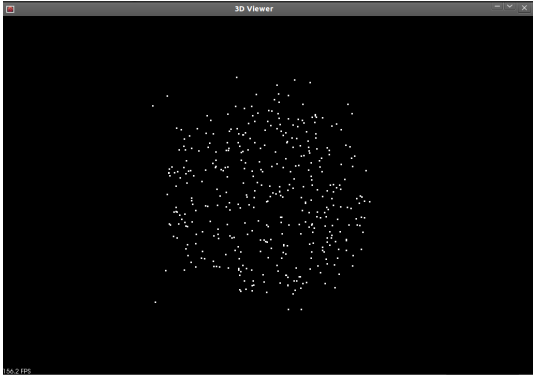
Fitting the model of a line with a point cloud:

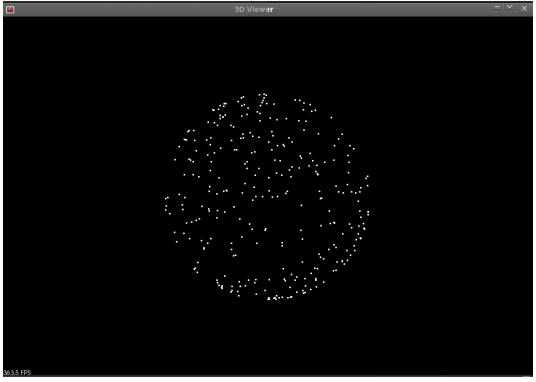




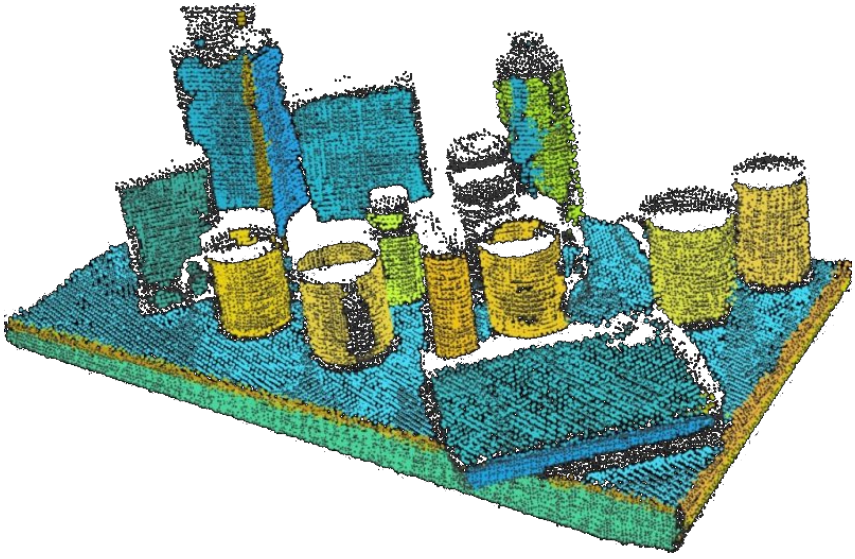
RANSAC

Fitting the model of a sphere with a point cloud:





RANSAC



Implicit Shape Model

This algorithm combines a generalized Hough transform and the Bag-of-words approach.

It has two steps, training and recognition:

1. Training: a model is computed from a training set of clouds.
2. Recognition: the model is used to predict the center of an object, in a different cloud.

Implicit Shape Model

Training:

1. Keypoint detection is made. Normally, just a downsampling with a voxel grid.
2. Features are estimated for every keypoint (i.e., FPFH).
3. All features are clustered with a k-means algorithm to construct a dictionary of visual "words". One visual word per cluster, one instance of the word per feature.
4. For every instance we get the direction from the keypoint to the center of mass.
5. For each word, we calculate the statistical weight.
6. For every keypoint, we calculate the learned weight.

Implicit Shape Model

Statistical weight that weights all the votes cast by visual word v_j for class c_i:

$$W_{st}(c_i, v_j) = \frac{1}{n_{vw}(c_i)} \frac{1}{n_{vot}(v_j)} \frac{\frac{n_{vot}(c_i, v_j)}{n_{ftr}(c_i)}}{\sum_{c_k \in C} \frac{n_{vot}(c_k, v_j)}{n_{ftr}(c_k)}}$$

$n_{vot}(v_j)$: total number of votes from visual word v_j
$n_{vot}(c_i, v_j)$: number of votes for class c_i from v_j
$n_{vw}(c_i)$: number of visual words that vote for class c_i
$n_{ftr(c_i)}$: number of features from which c_i was learned.
C: set of all classes

Implicit Shape Model

Learned weight:

$$W_{lrn}(\lambda_{ij}) = f\left(\left\{ e^{-\frac{d_a\left(\lambda_{ij}^2\right)}{\sigma^2}} \mid a \in A \right\}\right)$$

λ_{ij}: vote cast by a particular instance of word v_j on a particular training shape of class c_i (distance of the particular instance of word v_j to the center of the training shape on which it was found)
A: set of all features associated with word v_j on a shape of class c_i
σ: variable recommended to be 10% of the shape size
f: median function
$d_a(\lambda_{ij})$: Euclidean distance between voted and actual center

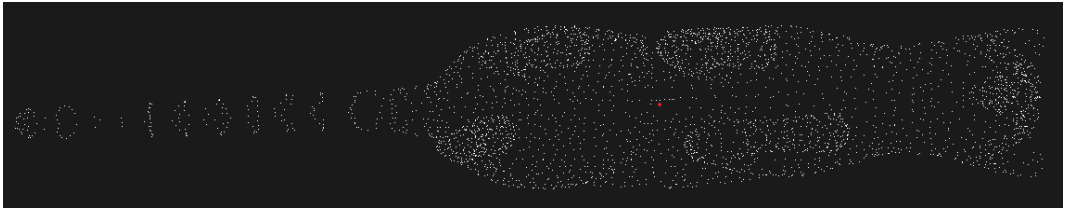
Implicit Shape Model

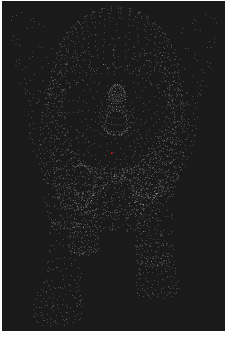
Recognition:

1. Keypoint detection.
2. Feature estimation for every keypoint of the cloud.
3. For each feature, search nearest visual word (cluster) in dictionary.
4. For every feature:
   - For every instance which casts a vote for the class of interest, of every visual word from the trained model:
     - Add vote with the corresponding direction and power.
5. Analyze all votes with non maxima suppression to get the single point marking the center.

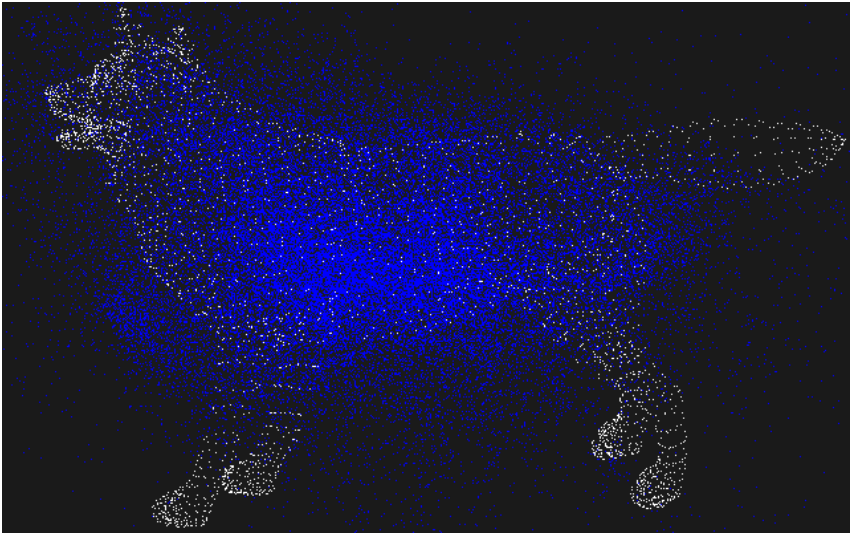
Vote power formula:

$$W(\lambda_{ij}) = W_{st}(v_j, c_i) * W_{lrn}(\lambda_{ij})$$





Implicit Shape Model



Template alignment

Use SAC to fit previously captured templates to a new model.

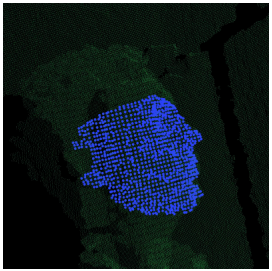
It returns:

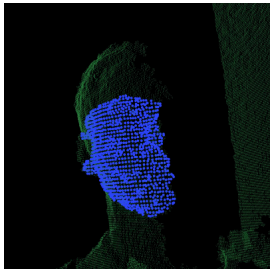
- The *fitness* level of the alignment (the lower, the better).
- A transformation matrix telling how the template should be rotated and translated in order to best align with the cloud:

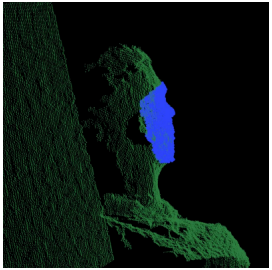
$$T = \begin{bmatrix} & & & t_x \\ & R & & t_y \\ & & & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

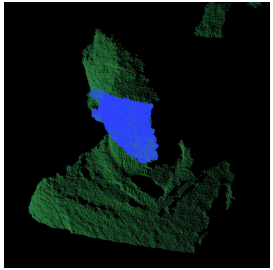
R: 3x3 rotation matrix
(t_x, t_y, t_z): translation vector

Template alignment







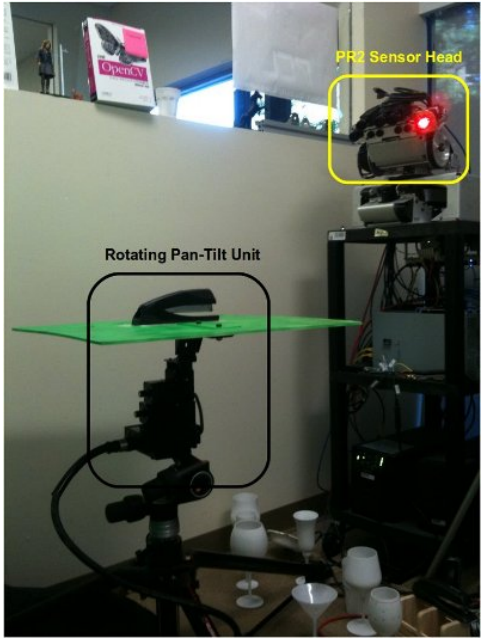


VFH cluster recognition

Viewpoint Feature Histograms (VFH) are meta-local descriptors, for cluster recognition and 6DOF pose estimation. This method returns a set of potential candidates.

Stages:

- Training
  1. Set up a scene with 1 object easily separable as cluster.
  2. Use a ground-truth system to obtain the pose.
  3. Rotate and conpute a VFH descriptor for each view.
  4. Save the views and build a kd-tree.
- Testing
  1. Extract the clusters of the new scene.
  2. Compute the VFH descriptor for each one.
  3. Search for candidates in the trained kd-tree.

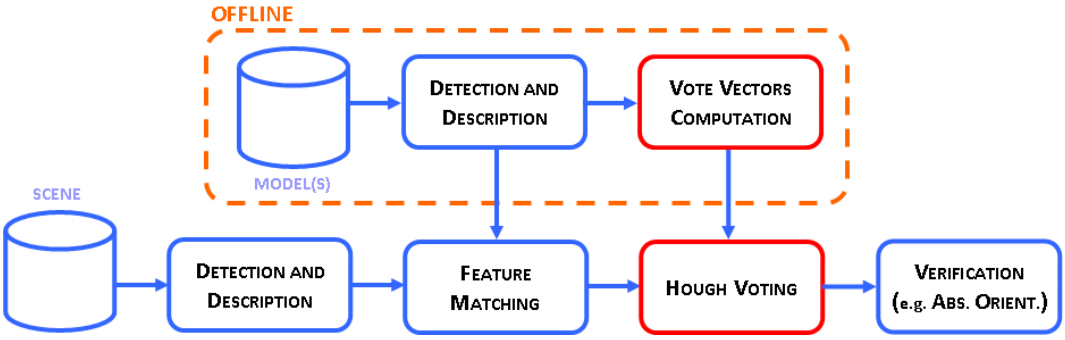

PR2 Sensor Head
Rotating Pan-Tilt Unit

Correspondence grouping

Uses point-to-point correspondences to match model instances in a scene, using 3D descriptors and a 3D Hough voting scheme. Also returns the transformation matrix.

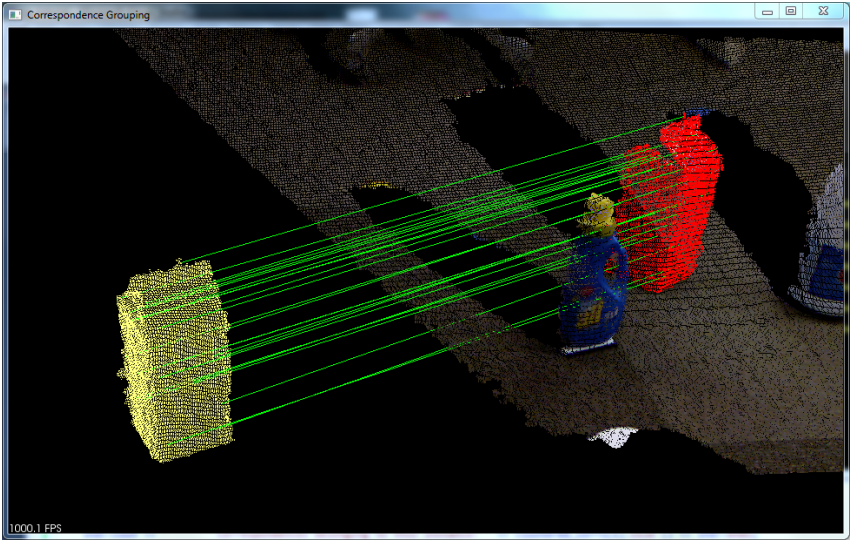
This algorithm needs to associate a Local Reference Frame (LRF) for each keypoint. The voting seeks for evidence of the object being sought in a given location.

Correspondence grouping

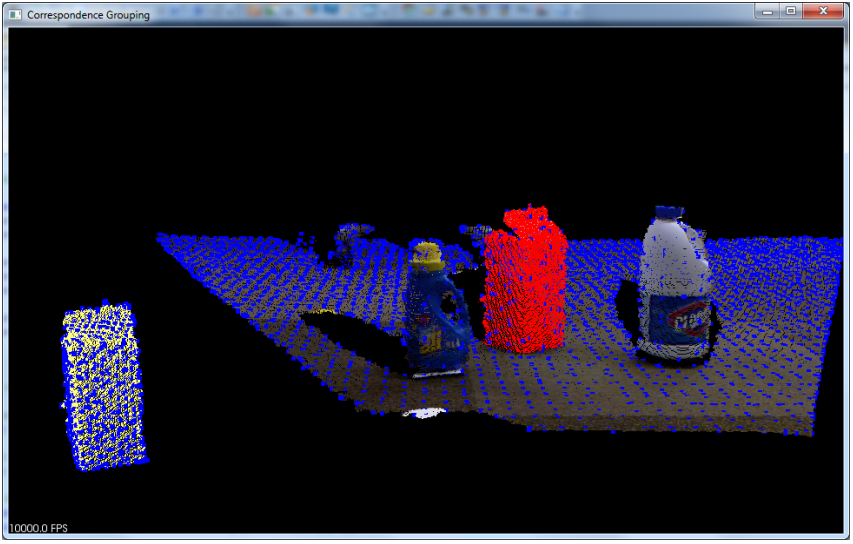


Correspondence grouping

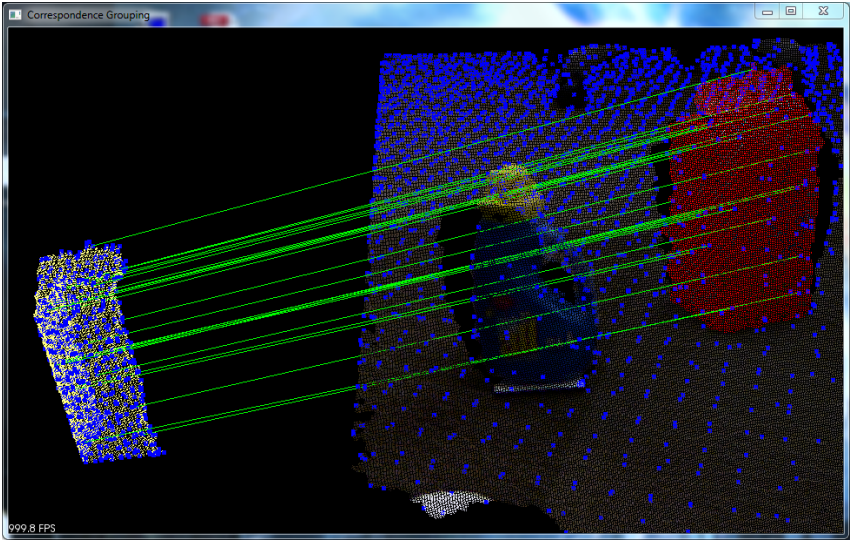
Correspondence grouping



Correspondence grouping



Correspondence grouping



NARF

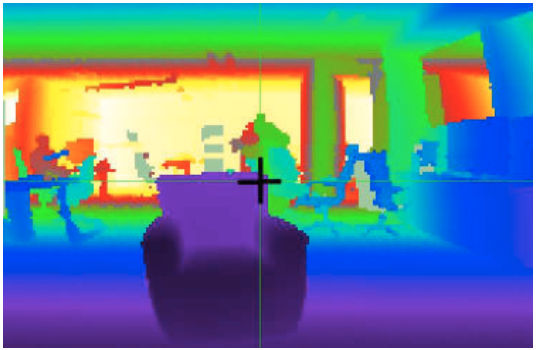
Normal Aligned Radial Feature (NARF) descriptors have certain advantages:

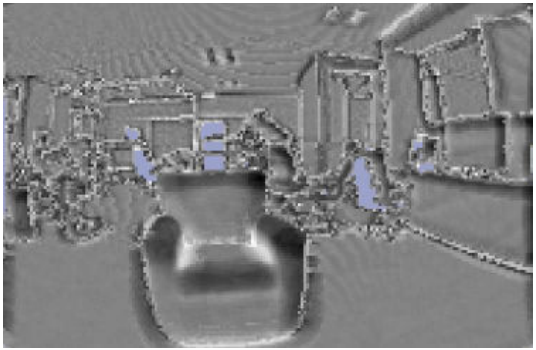
- They capture data about occupied and free space arond a keypoint.
- They are robust against noise.
- Optionally, they are rotation invariant.

NARF

To get the descriptor for a keypoint:

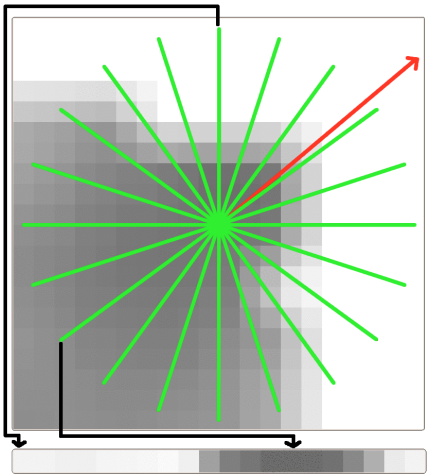
1. A normal aligned range value patch is calculated on the point (a small range image with the observer looking at the point along the normal).





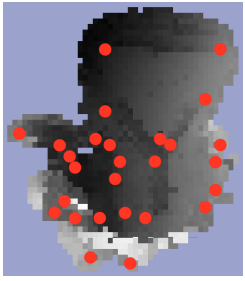
NARF

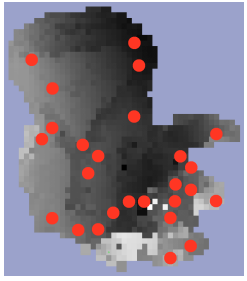
2. A star pattern is overlayed onto the patch. Each beam corresponds to a value in the final descriptor. It captures how much the pixels under it change.

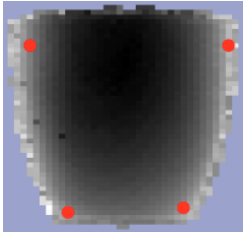


3. The unique orientation of the descriptor is extracted. The 360° are discretized into a number of bins and an histogram is created. The maximum histogram bin will be the dominant orientation.
4. The descriptor is shifted according to this value to achieve rotational invariance.

NARF







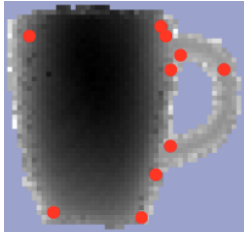
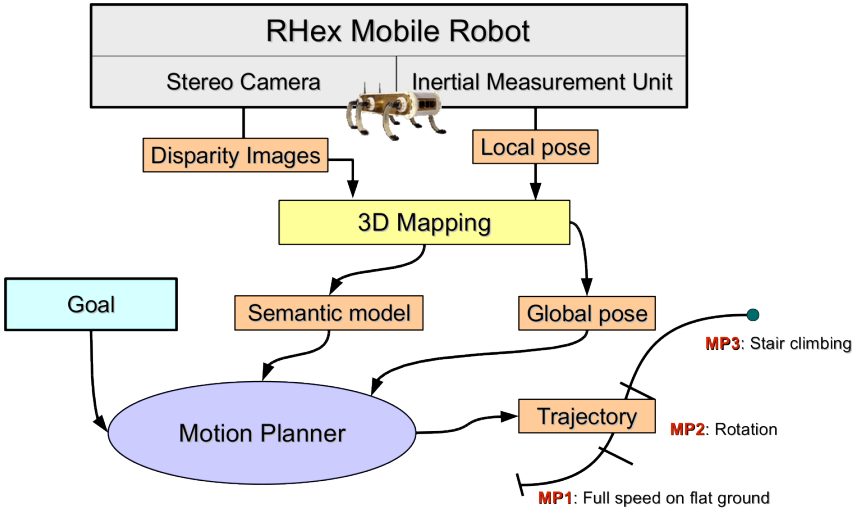


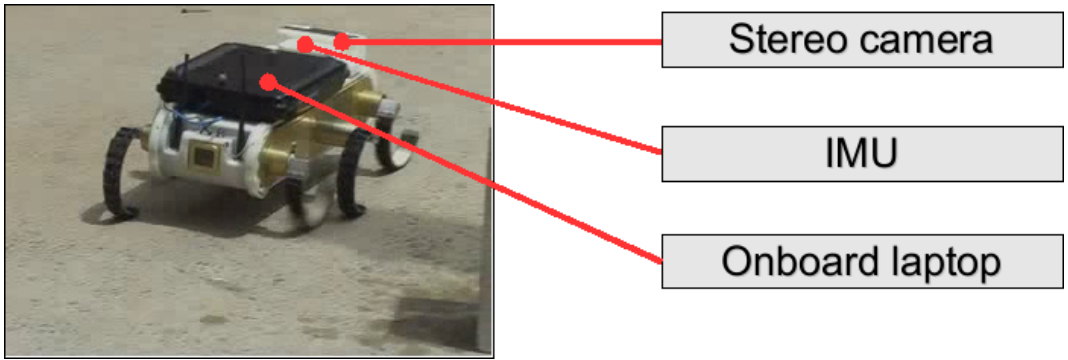
Table of Contents

Navigation

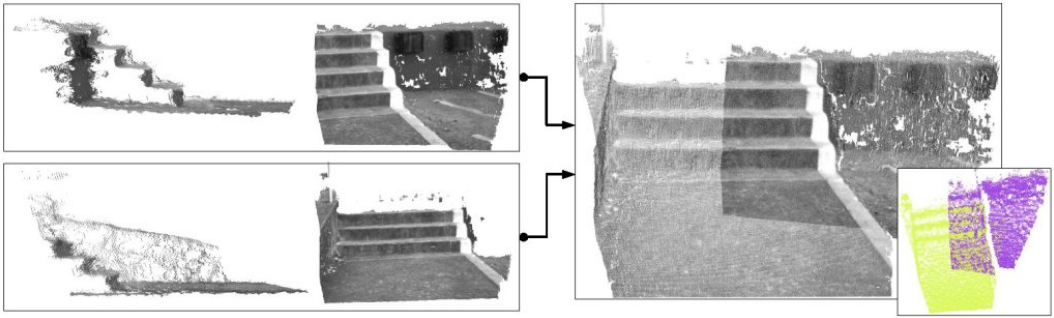
Depth sensors make spatial navigation easier for robots.



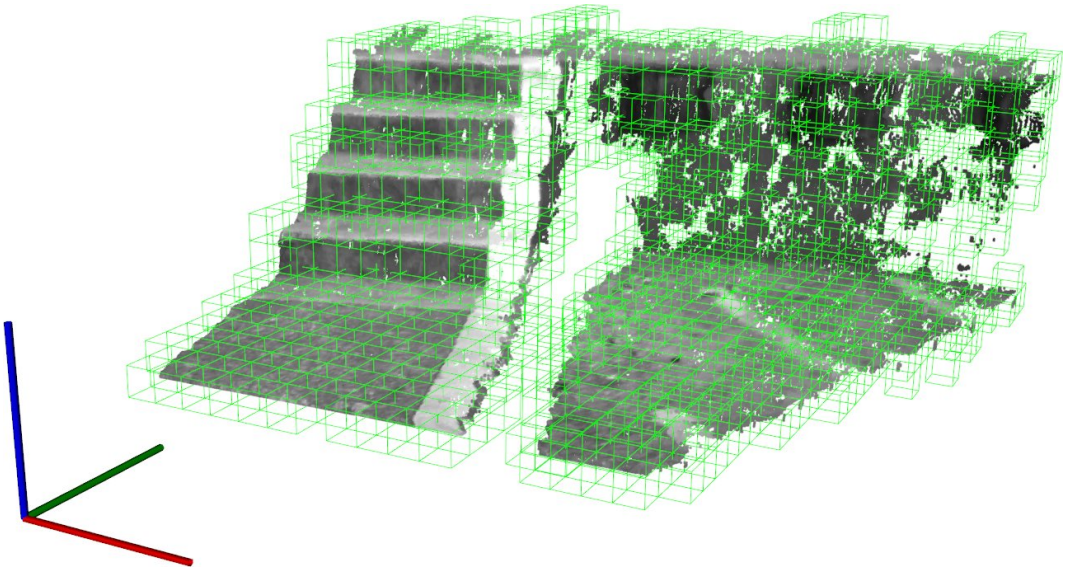
Navigation



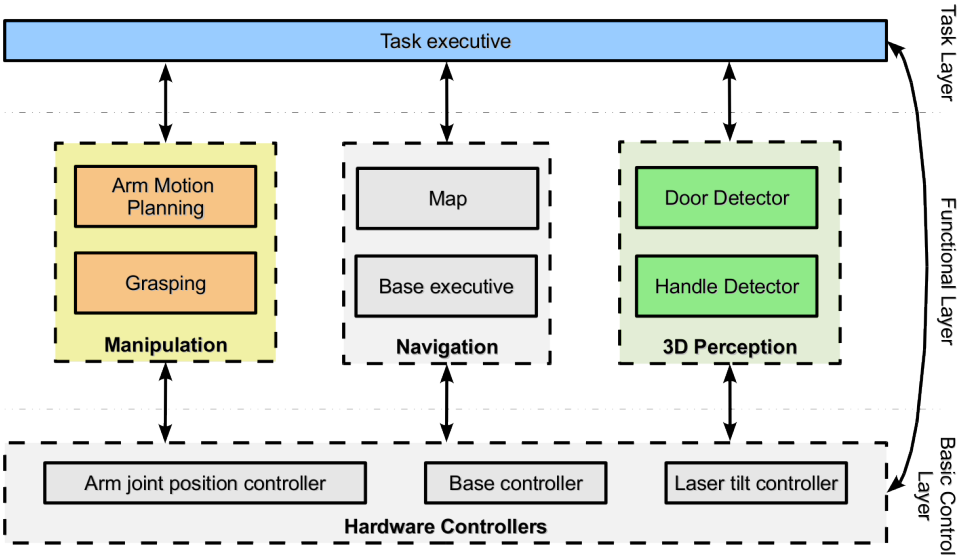
Navigation



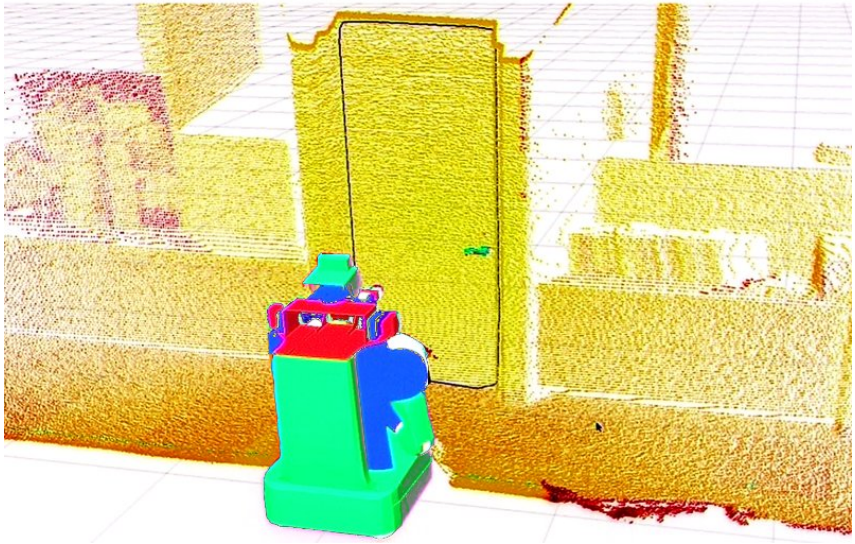
Navigation



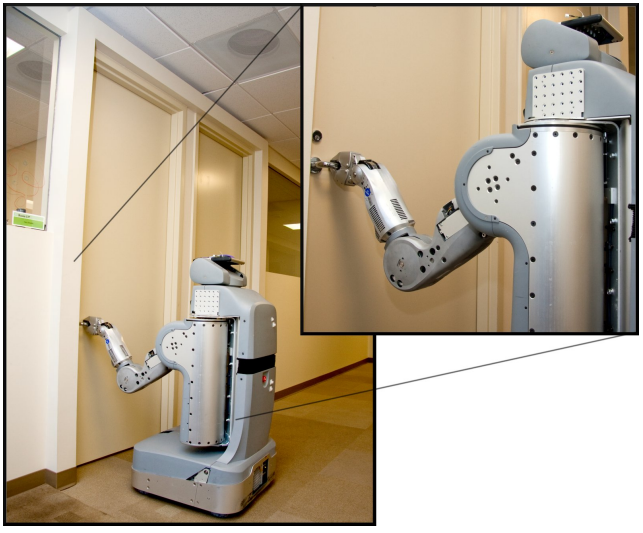
Opening doors



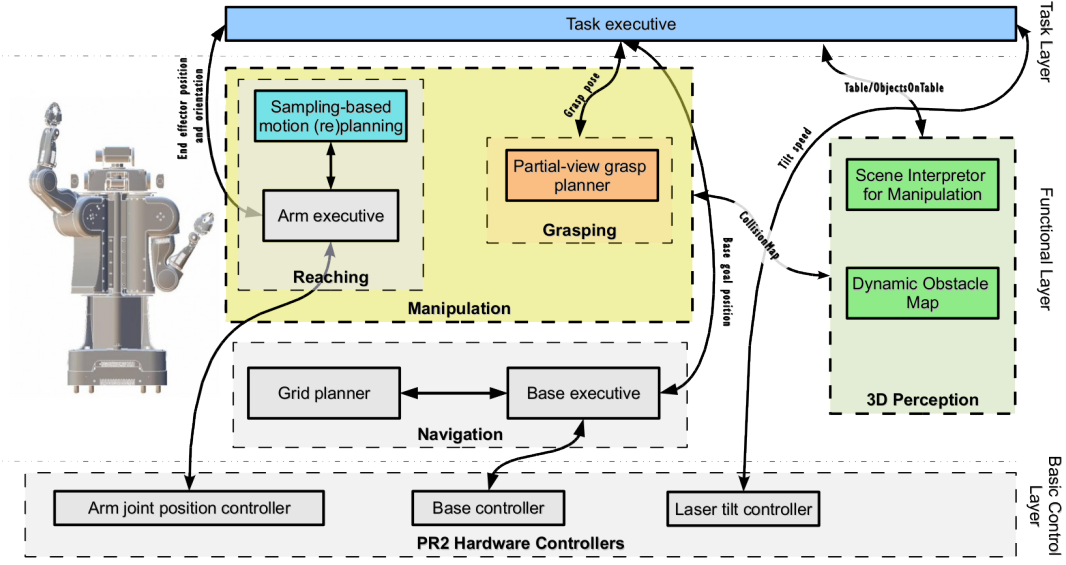
Opening doors



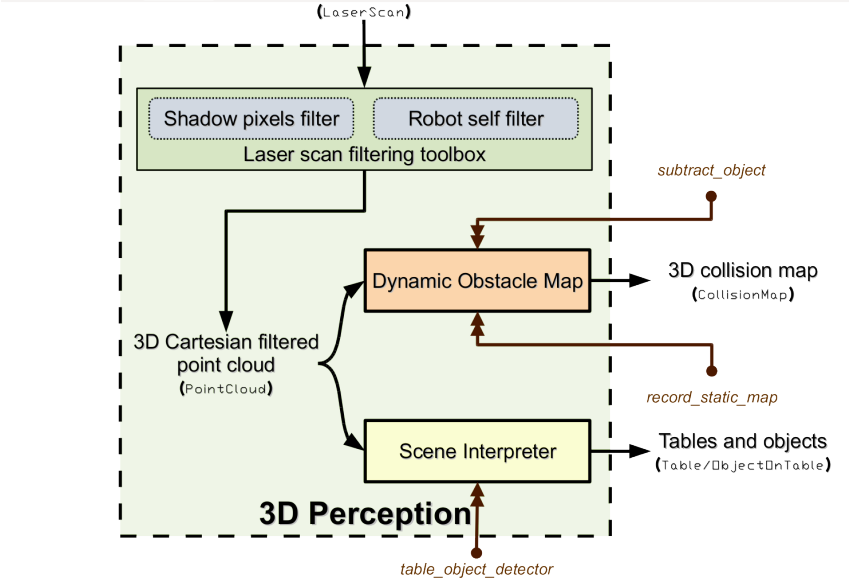
Opening doors



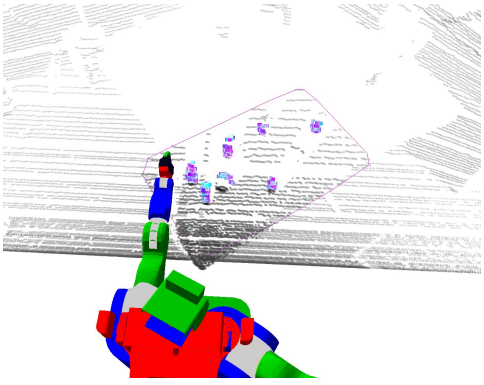
Object grasping

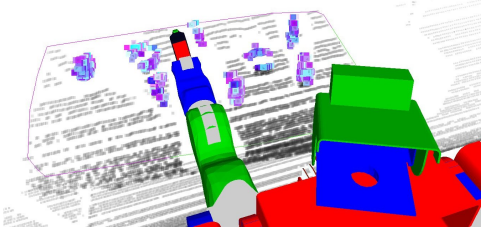


Object grasping



Object grasping





Object grasping

Pose estimation

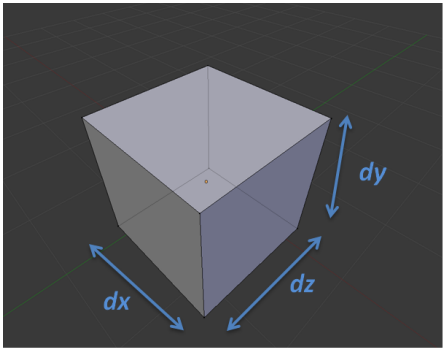
Real time 3D scanning

ICP is used to register each new frame into the existing world model.

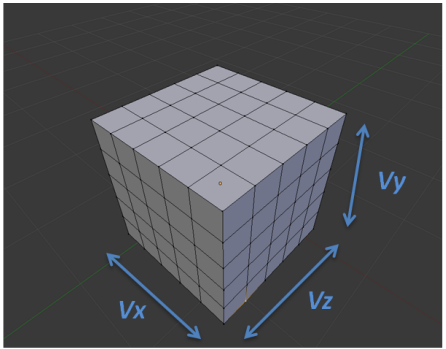
To do it real-time, parallel processing on the GPU is mandatory.

Real time 3D scanning

To get advantage of how the data is stored in the GPU at runtime, a new type of point cloud is needed: the Truncated Surface Distance Function (TSDF) cloud.



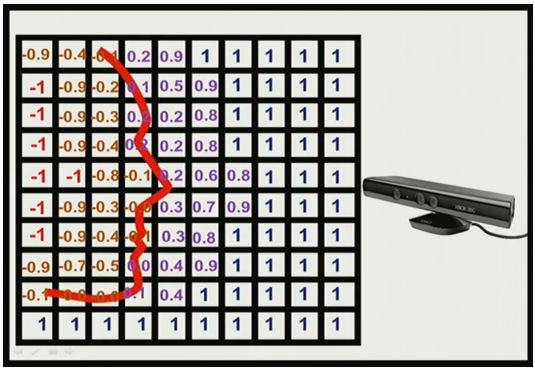
dx = dy = dz = 3 [meters]



Vx = Vy = Vz = {32, 64, 128, 256, 512} [voxels]

Real time 3D scanning

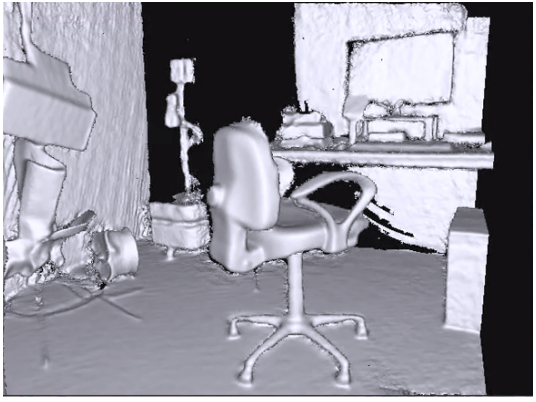
The TSDF cloud is a 3x3m cube subdivided 512 times per axis (512^3 voxels).

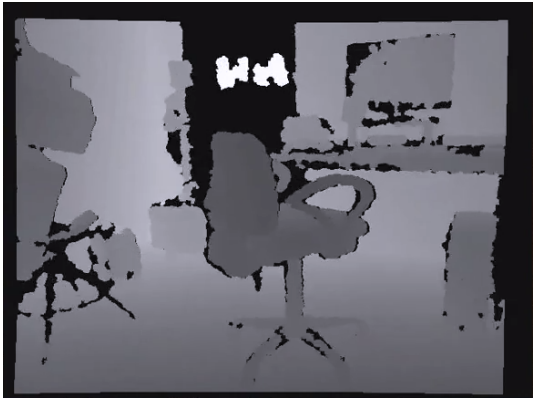


The TSDF value of every voxel is the distance to the nearest isosurface. It ranges from 1 to -1, positive when "in front" of a surface, negative when inside. Voxels with a value of 1 are not extracted to save memory, since they represent empty space.

Real time 3D scanning

A camera tracking solution is running on the background. When it detects that the sensor has left the 3x3m cube, all data is sent from the GPU to the CPU, and it starts again with a new TSDF cloud.





Real time 3D scanning

Thank you for viewing...