Tutorial / Einführung

Für das Modul

Objekterkennung und Geodatenfusion

Point Cloud Library

Fragen an :

danilo.schneider@tu-dresden.de

stephan.maes@tu-dresden.de

bernd.grafe@tu-dresden.de

Inhalt:

1. Was ist PCL?

PCL ist ein umfangreiches und plattformunabhängiges Standalone Open-Source-Framework zur 3D/4DPunktwolkenverarbeitung und Geometrieprozessierung geschrieben in C++.
PCL wird unter der BSD Lizenz geführt und ist daher für den kommerziellen und wissenschaftlichen Gebrauch frei. Die Bibliothek beinhaltet eine sehr umfangreiche Anzahl an Modulen (bzw. modular aufgebauten Bibliotheken) mit den wichtigsten Algorithmen für diverse Anwendungsmöglichkeiten und untersteht der ständigen Weiterentwicklung.

2. Was kann PCL?

- Filterung
- Oberflächenkonstruktion
- Schätzverfahren
- Model fitting
- Erkennung von Ausreißern
- Rauschminderung
- Segmentierung
- Objekterkennung
- Visualisierung
- Punktwolkenregistrierung
- Usw.

3. Wie funktioniert PCL?

Der Aufbau erfolgt in Form von Klassen/Funktionen in Modulen mit allgemeingehaltenen Prozessen für hohe Wiederverwendbarkeit und Allgemeingültigkeit unter Verwendung gängiger APIs und Bibliotheken:

- Eigen (Matrizen, Vektoren, Lineare Algebra etc.)
- FLANN (Fast Library for Approximate Nearest Neighbors)
- Qt (UI Framework)
- VTK (Visualization Toolkit)
- Qhull (konvexe Hülle, Delaunay Triangulation, Voronoi etc)
- OpenNI (Sensorerfassung)
- sowie OpenMP (Thread Multiprocessing), TBB (Threading Building Blocks), Boost shared Pointers

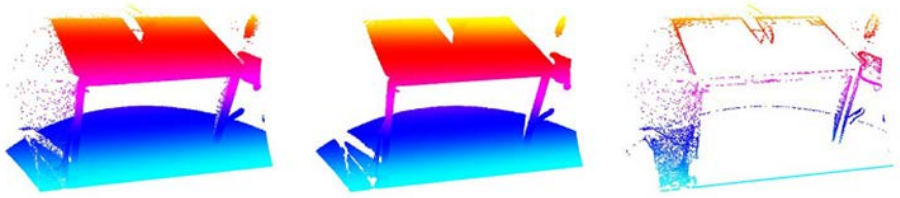
Einfacher Programmablauf als Piplineschema:

1. Prozessobjekt erstellen
2. Input setzen
3. Parameter setzen
4. Ausführen

!

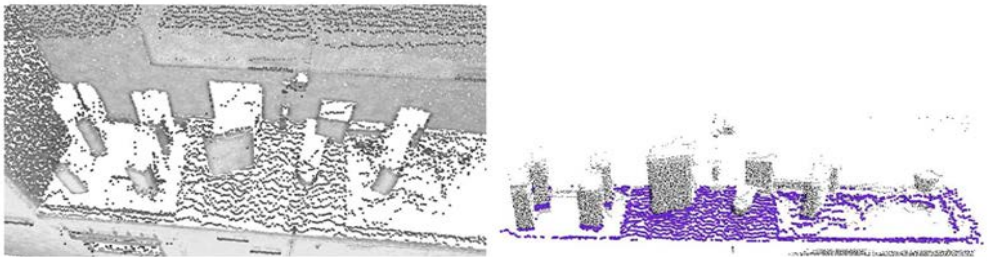
Beispiele für den Ablauf:

```
pcl::StatisticalOutlierRemoval<pcl::PointXYZ> f;        1
f.setInputCloud (input_cloud);                          2
f.setMeanK (50);                                        3
f.setStddevMulThresh (1.0);                             3
f.filter (output_cloud);                                4
```



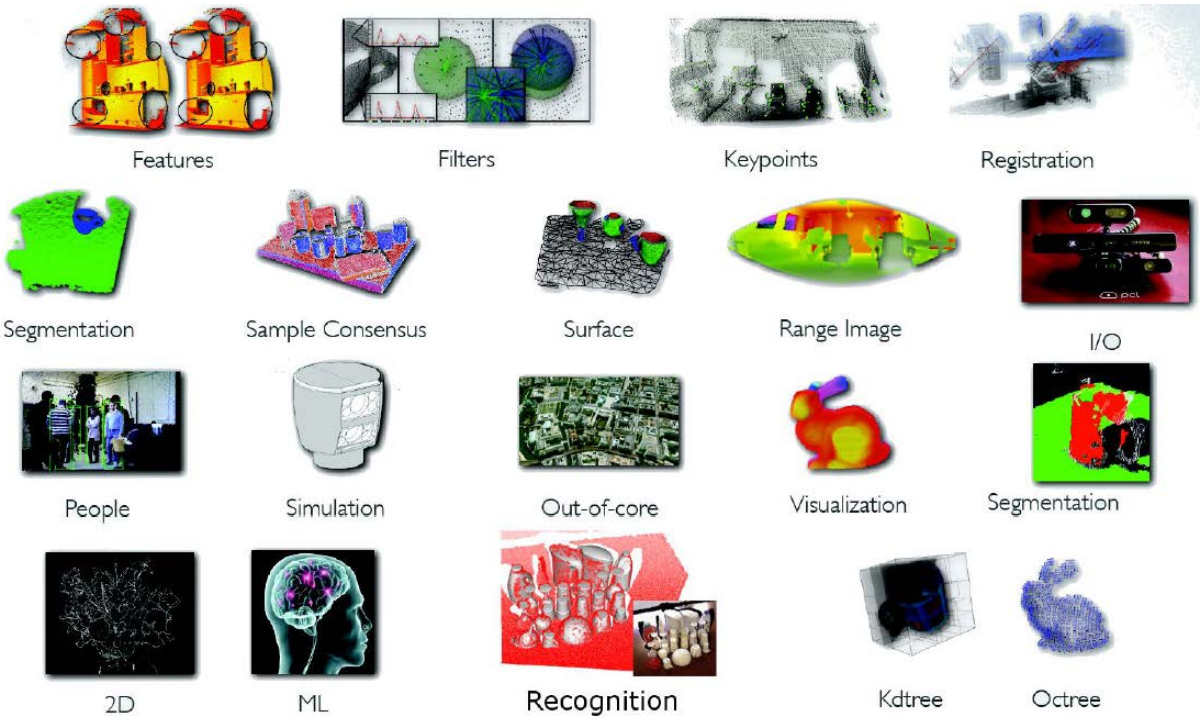
Beispiel für Ausreißerentfernung mit k=50 Nachbarschaften

```
pcl::SACSegmentation<pcl::PointXYZ> s;                  1
f.setInputCloud (input_cloud);                          2
f.setModelType (pcl::SACMODEL_PLANE);                   3
f.setMethodType (pcl::SAC_RANSAC);                      3
f.setDistanceThreshold (0.01);                          3
f.segment (output_cloud);                               4
```

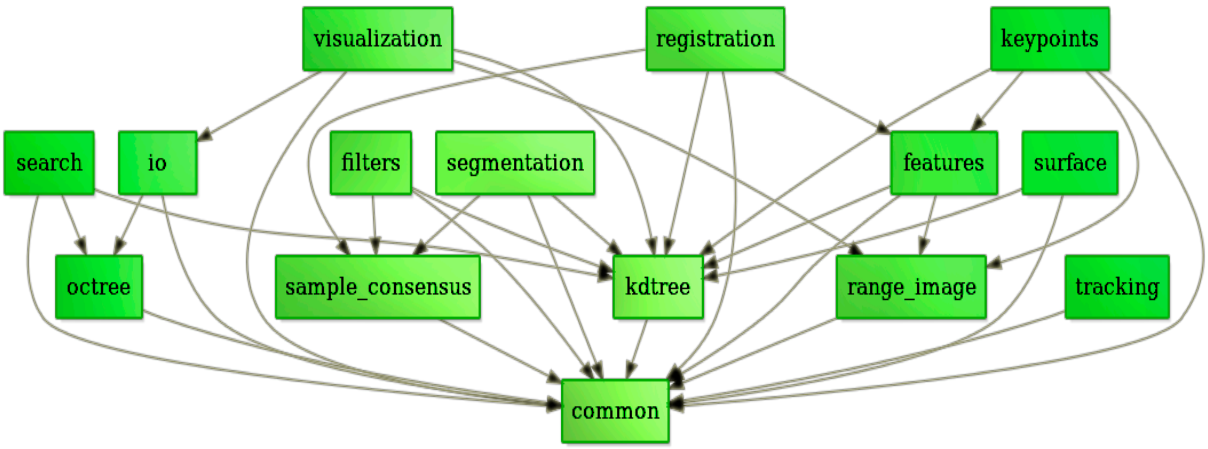


Segmentierung/Ebenenerkennung mit Ransac (1cm Toleranz)

4. Module



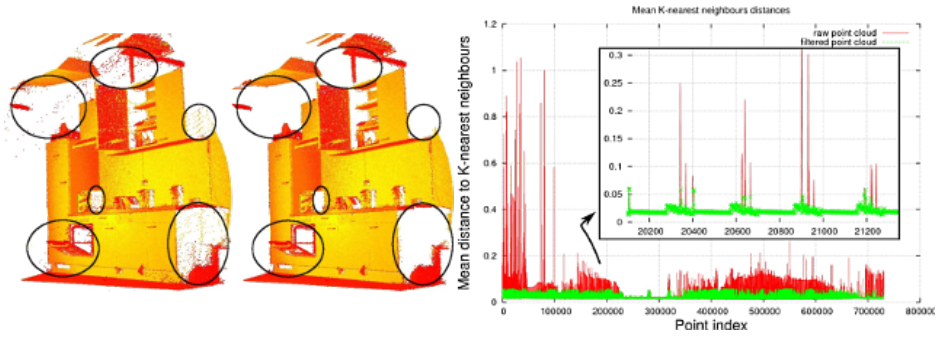
Modulübersicht von PCL 1.7 (All-In-One Installer nur für 1.6.0 verfügbar**)**



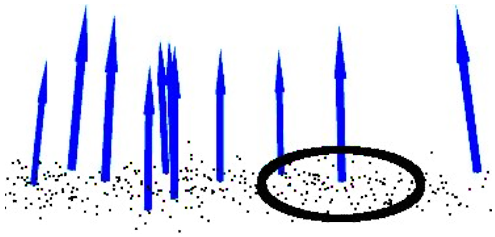
Modulbeziehungen / Abhängigkeiten

kurze Übersicht ausgewählter Module:

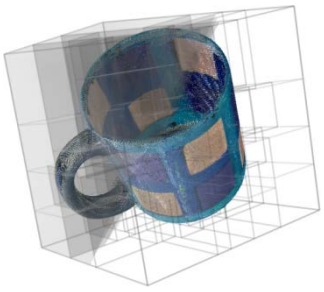
- **Common**:
  - Grundlage
  - Gemeinsam genutzte Datenstruktur und Methoden
  - (PointCloudClass, Punkttypen, Repräsentationen, Distanzberechnung, Transformationen, Mittelwertberechnung etc.)
- **Filter**:
  - Statistische Ausreißerverfahren (Ausreißer, Rauschen eliminieren)



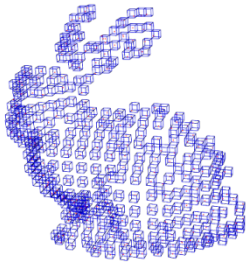
- **Features:**
  - Schätzfunktionen
  - Normalen-, Krümmungsberechnung
  - K-Nachbarschaften



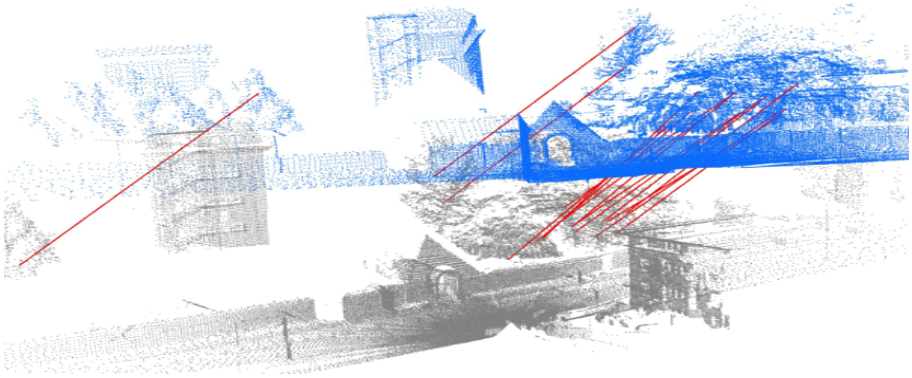
- **IO:**
  - Input/Output – Lesen/Schreiben der Punktwolke in Point Cloud Data
- **KDTree**:
  - Nachbarschaftssuche
  - K-dimensionaler Baum zur Raumpartitionierung
  - Splittet entlang der Dimension
  - Binärbaum



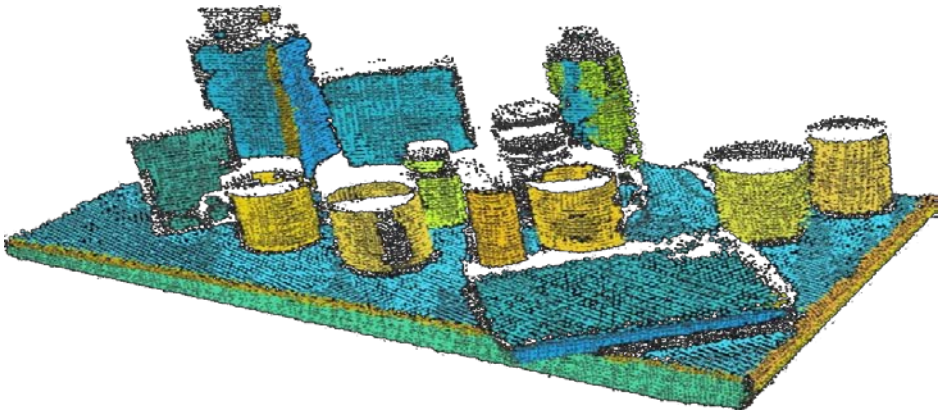
- **OCTree**
  - 8er Baum (nicht binär)
  - Splittet entlang des Punktes
  - Ebenso Raumteilung



- **Keypoints:**
  - POI
  - Kleiner stabiler Subset
  - Einzigartige Eigenschaften
  - Repräsentation der Cloud
  - Schnellere Berechnungen
- **Registration**:
  - Punktwolken verbinden
  - Transformation

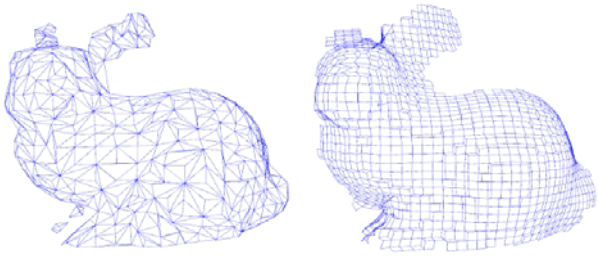


- **Sample Consensus:**
  - Template Objekte für Abgleich
  - Auffinden geometrischer Formen (Zylinder, Kugel etc.)
  - Z.B. mit Hilfe von RANSAC
  - Modeldetektion + Parameter
  - (Plane Fitting: Türen, Wände)



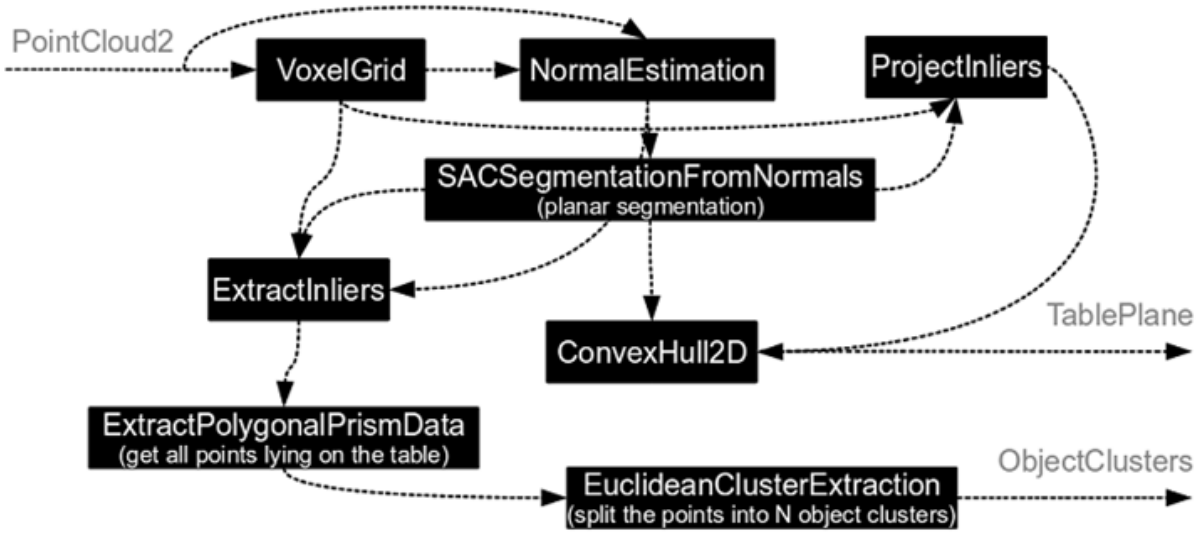
- **Segmentation:**
  - Clusterbildung
  - Auffinden isolierter Regionen
  - Geeignet zur Vorprozessierung

- **Surface:**
  - Rekonstruierung
  - Rauschreduzierung
  - Smoothing
  - Meshing



- **Visualization**:
  - Darstellung von Punkten, diverser Formen, Normalen, Histogramme etc.

Module bzw. deren Klassen/Funktionen können im Prozessablauf gekoppelt werden. Folgendes Bild veranschaulicht die Verknüpfung der Funktionen zur Identifizierung einer Ebene. Dabei wird jede einzelne Klasse, wie gezeigt, erstellt (Objekt erstellen, Input/Parameter setzen, Ausführen). Der Output der einen Klasse wird somit Input der nächsten.

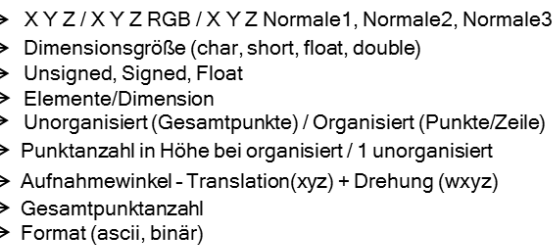


Nähere Beschreibungen zu den Funktionen folgen im Abschnitt 6.

5. Das Dateiformat PCD

Die Point Cloud Data (PCD) ist das eigene Format für PCL, da andere bereits existierende Formate nur teilweise den notwendigen Kriterien für PCL entsprechen.

```
# .PCD v.7 - Point Cloud Data file format
VERSION .7
FIELDS x y z rgb
SIZE 4 4 4 4
TYPE F F F F
COUNT 1 1 1 1
WIDTH 213
HEIGHT 1
VIEWPOINT 0 0 0 1 0 0 0
POINTS 213
DATA ascii
0.93773 0.33763 0 4.2108e+06
0.90805 0.35641 0 4.2108e+06
0.81915 0.32 0 4.2108e+06
0.97192 0.278 0 4.2108e+06
0.944 0.29474 0 4.2108e+06
0.98111 0.24247 0 4.2108e+06
0.93655 0.26143 0 4.2108e+06
0.91631 0.27442 0 4.2108e+06
```



Die Datei besteht aus einem Header und Body. Im Header werden alle Metadaten bezüglich der Art der Punktwolke gespeichert (siehe oben). Der Body beinhaltet folglich die einzelnen Koordinaten der Punkte.

Vom Format CSV zu PCD muss selbständig umgewandelt werden – hierbei kann ein Tool bereitgestellt werden.

6. Wichtige Funktionen/Klassen am Beispiel eines Workflows für Flächenerkennung

Der Workflow zur Bearbeitung und Analyse einer Punktewolke lässt sich gut anhand der Reihenfolge der genutzten PCL-Klassen darstellen. Die hier aufgelisteten Informationen zu Klassen und Funktionen stammen aus der Dokumentation zu PCL.

VoxelGrid

Die VoxelGrid Klasse erstellt ein 3D Voxel Grid über einer gegebenen Punktwolke. Danach werden in jedem Voxel alle darin enthaltenen Punkte zusammengerechnet und durch den gemeinsamen Schwerpunkt ersetzt. Diese Herangehensweise ist ein wenig langsamer als die Punkte durch den Mittelpunkt des Voxels zu repräsentieren, aber sie repräsentiert die darunterliegende Oberfläche besser.

Parameter

- Leaf Size: Gibt die Größe der Blätter des VoxelGrids an.
    - float lx (0.5)
    - float ly (0.5)
    - float lz (0.5)

OutlierRemovel

Dieser Algorithmus iteriert zweimal über den gesamten Input: Während des ersten Durchlaufs berechnet der Algorithmus den durchschnittlichen Abstand eines jeden Punktes zu seinen nächsten k Nachbarn. Danach werden der Mittelwert und die Standardabweichung aller dieser Abstände berechnet, um einen Schwellenwert zu erhalten. Der Schwellenwert ergibt sich aus: Mittelwert (mean) + Standardabweichung (stddev) * Multiplikator (stddev_mult). Während des zweiten Durchlaufs werden die Punkte als Inlier oder Outlier klassifiziert, je nachdem ob deren durchschnittlicher Abstand zu deren Nachbarn über oder unter dem berechneten Schwellenwert liegt.

Parameter

- MeanK: Gibt die Anzahl der Nachbarn an, die genutzt werden, um den mittleren Abstand zu berechnen.
    - int nr_k (10)
- StddevMulThresh: Ist der Multiplikator in der Berechnung des Schwellenwerts, welcher genutzt wird, um Punkte anhand ihres mittleren Abstands zu k Nachbarn als Inlier oder Outlier zu klassifizieren. Der Schwellenwert ergibt sich aus: Mittelwert (mean) + Standardabweichung (stddev) * Multiplikator (stddev_mult)
    - double stddev_mult (0.2)

SACSegmentation

Die SACSegmentaion Klasse repräsentiert die Nodelet Segmentierung für Sample Consesus Methoden und Modelle. In diesem Zusammenhang erstellt sie einen Nodelet wrapper für generische Zwecke für Segmentierungen, die auf SAC aufbauen.

Parameter

- ModelType: Gibt an welches Modell als Grundlage genutzt werden soll. Die Enumeration *enum pcl::SacModel* listet alle möglichen Typen auf.
    - int model (pcl::SACMODEL_PLANE)

- MethodType: Gibt an welche Art von Sample Consensus genutzt werden soll. Die möglichen Arten sind als namespaces hinterlegt.
    - int method (pcl:: SAC_RANSAC)
- DistanceThreshhold: Gibt den Abstand an, welcher für das gewählte Model als Schwellenwert dienen soll.
    - double threshhold (0.5)
- MaxIterations: Gibt die maximale Anzahl an Iterationen an bevor der Algorithmus abbricht.
    - int max_iterations
- Probability: Gibt an wie hoch die Wahrscheinlichkeit sein soll, dass wenigstens eine Lösung keine Outliers mehr enthält.
    - double probability
- OptimizeCoefficients: Dieser Parameter wird auf true gesetzt, sollte eine Verbesserung der Koeffizienten nötig sein.
    - bool optimize
- RadiusLimits: Gibt den maximal und minimal erlaubten Radius für das gewählte Model an (anwendbar für Modelle, die einen Radius schätzen)
    - const double &min_radius,
    - const double &max_radius
- SamplesMaxDist: Gibt die maximal erlaubte Entfernung für das Zeichnen der zufälligen Objekte an.
    - const double &radius
    - SearchPtr search
- Axis: Ist eine Achse, welche senkrecht zum gesuchten Objekt verlaufen soll.
    - const Eigen::Vector3f &ax
- EpsAngle: Gibt den maximal erlaubten Winkel zwischen der gesetzten Achse und der Normalen des Models an.
    - double ea

EuclideanClusterExtraction

Dies ist eine Klasse zur Segmentierung von Clustern im euklidischen Sinne.

Parameter

- SearchMethod: Die Funktion benötigt einen Pointer auf das zu durchsuchende Objekt.
    - const KdTreePtr &tree (pcl::search::KdTree<pcl::PointXYZ>::Ptr)
- ClusterTolerance: Räumliche Toleranz des Clusters als ein Maß im L2 euklidischen Raum.
    - double tolerance (0.5)
- MinClusterSize: Beschreibt die Anzahl der Punkte, die mindestens in einem Cluster enthalten sein müssen, damit es als gültig angesehen wird.
    - int min_cluster_size (100)
- MaxClusterSize: Beschreibt die Anzahl der Punkte, die maximal in einem Cluster enthalten sein dürfen, damit es als gültig angesehen wird.
- int max_cluster_size (10000)

BoundaryEstimation

Diese Klasse berechnet, ob eine Reihe von Punkten auf einer Oberfläche liegt in dem es ein Winkelkriterium nutzt.

Der Code nutzt berechnete Normalen der Oberfläche in jedem Punkt der Inputdaten.

Parameter

- AngleThreshold: Der Schwellenwert auf dessen Grundlage Punkte als normal oder zur Oberfläche gehörend eingestuft werden. (default $\frac{\pi}{2.0}$)
    - Float angle
- InputNormals: Ein Pointer auf die Inputdaten, welche die Normalen in den Punkten eines XYZ Datensatzes beinhalten.
    - const PointCloudNConstPtr &normal
- SearchMethod: Ein Pointer zu dem gewünschten Suchobjekt.
    - const KdTreePtr &tree
- KSearch: Gibt die Anzahl der k nächsten Nachbarn an, die für die Berechnung genutzt warden.
    - Int k
- RadiusSearch: Gibt den Radius an, der genutzt wird, um die nächsten Nachbarn zu finden.
    - double radius
- InputCloud: Ein Pointer zu den Inputdaten.
    - const PointCloudConstPtr &cloud
- Indices: Es wird ein Pointer zum Vektor von Indizes, die die Inputdaten repräsentieren benötigt.
    - const PointIndicesConstPtr &indices

ConcaveHull

Parameter

- Alpha: Dieser Wert gibt an, wie groß die resultierenden Hüllensegmente maximal sein dürfen (umso kleiner Alpha, um so detaillierter die Hülle).
    - double alpha (5)

PlaneWithPlaneIntersection

Findet die Gerade, welche den Schnitt zweier nicht paralleler Ebenen darstellt mit Hilfe von Lagrange Multiplikatoren.

Parameter

- const Eigen::Vector4f &plane_a
    - Koeffizienten der ersten Ebene in der Form: ax + by + cz + d = 0
- const Eigen::Vector4f &fplane_b
    - Koeffizienten der zweiten Ebene in der Form: ax + by + cz + d = 0
- Eigen::VectorXf &line
    - Koeffizienten der Gerade, wobei line.tail<3>() = Richtung und line.head<3>() = Punkt auf der Gerade (am Nächsten zu (0,0,0)) gilt
- Double angular_tolerance = 0.1

LineWithLineIntersection

Findet den 3D Punkt, der den Schnitt zweier Geraden im Raum darstellt.

Parameter

- const pcl::ModelCoefficients &line_a
  - Koeffizienten der ersten Gerade (Punkt, Richtung)
- const pcl::ModelCoefficients &line_b
  - Koeffizienten der zweiten Gerade (Punkte, Richtung)
- Eigen::Vector4f &point
  - Platzhalter für gefundenen 3D-Punkt
- Double sqr_eps = 1e-4
  - Maximal erlaubter Quadratabstand zur richtigen Lösung

7. Möglicher Programmablauf für einen Workflow

Einfacher beispielhafter Programmablauf (die in Klammer stehenden Dateinamen dienen als Hinweise für die Input/Output Benutzung):

1. Cloud einlesen
2. VoxelGrid für downsampling / Reduzierung der Punktdichte (file=name+downsampled.pcd)
3. OutlierRemoval (file=name+filtered.pcd)
4. Loop zur Flächenerkennung
    a. Segmentation (Finden zusammengehöriger Punkte auf Ebene -> Ebenenkoeffizienten + Pointcloud ; Schleife, da jeweils einmalig Ebenenpunkte gefunden werden, jene speichern, aus Urwolke löschen, weitersuchen. (file=name+plane_#.pcd)
    b. EuclideanClusterExtraction (KD Tree) große Mehrecke in einzelne Vierecke zerlegen (file=name_plane_#_cluster_#.pcd)
    c. BoundaryEstimation (Randpunkte auswählen) (file=name+plane+cluster+boundary.pcd)
    d. ConvexeHull|| LineSegmentaion/mergeLines/sortPoints|| Segmentation+LineIntersect+sort||PlaneIntersect bzw linewithlineintersect

Alternative

1. Wie 1-3 mit Loop und Segmentation
    a. Koeffizienten verwenden für planewithPlaneIntersection (Ansatz, um Koeffizienten zu erhalten für Schnittlinie)
    b. Koeffizienten verwenden für linewithLineIntersection
2. Punktwolken mit RGB Wert
3. Viele andere mögliche Methoden und Prozessabläufe denkbar

Eine bessere Veranschaulichung kann der PP-Präsentation entnommen werden

Folglich mögliche Aufgaben/Ideen:

Fläche durch Koeffizienten aus Segmentierung rekonstruieren
Liniensegmentierung aus bereits gefundenen Flächen
RGB Punktwolke verwenden zusätzlich anderer Methoden
auf dem Beispiel der gefunden Flächencluster aufbauen

Es gibt eine große Menge an verschiedenen Möglichkeiten von Methoden und denkbaren Methodenkombinationen. Dieser Abschnitt gilt nur Denkanstoß.

Ebenso interessant zu betrachten: ***Implicit Shape Model*** *(könnte für wiederkehrende Objekte, wie Fenster, verwendet werde) und* ***SiftKeyPoint*** *(gibt signifikante Punkte unter Beachtung von Intensitäten wieder, also mit RGB Werten)*

8. Codebeispiele für einzelne Funktionen

Einlesen einer Punktwolke:

```
//Headerfiles/Module laden
#include <pcl/io/pcd_io.h>
#include <pcl/point_cloud.h>

//Punktwolke erstellen
pcl::PointCloud<pcl::PointXYZRGBA>::Ptr cloud (new cl::PointCloud<pcl::PointXYZRGBA> ());
//laden der Punktwolke
pcl::io::loadPCDFile ("yourPointCloud.pcd", *cloud);

/// hier können weitere Funktionalitäten auftauchen…

//Speichern der Punktwolke in 3 verschiedenen Möglichkeiten
pcl::io::savePCDFileASCII ("cloud_ascii.pcd", *cloud);
pcl::io::savePCDFileBinary ("cloud_binary.pcd", *cloud);
pcl::io::savePCDFileBinaryCompressed ("cloud_binary_compressed.pcd", cloud);
```

Darstellen der Punktwolke (einfach):

```
//Punktwolke erstellen
pcl::PointCloud<pcl::PointXYZRGB>::Ptr cloud;
//Punktwolke laden (siehe oben) oder neue erstellen

//CloudViewer zur Visualisierung erstellen
pcl::visualization::CloudViewer viewer ("Simple Cloud Viewer");

//Aufruf zur Darstellung der Punktwolke
viewer.showCloud (cloud);

while (!viewer.wasStopped ()){
//mögliche Prozessierungsanweisungen möglich in einem extra Thread
}
```

Darstellen der Punktwolke (erweitert):

```
main ()
{
pcl::PointCloud<pcl::PointXYZRGBA>::Ptr cloud (new
pcl::PointCloud<pcl::PointXYZRGBA>);pcl::io::loadPCDFile ("my_point_cloud.pcd", *cloud);
pcl::visualization::CloudViewer viewer("Cloud Viewer");
viewer.showCloud(cloud);

// da CloudViewer eingeschränkte Funktionen besitzt, wird hier der bessere PCLVisualizer aufgerufen
viewer.runOnVisualizationThreadOnce (viewerA); //wird nur beim ersten Aufruf ausgeführt
//alternativ
viewer.runOnVisualizationThread (viewerAlt); //wird bei jeder Vis.iteration ausgeführt

while (!viewer.wasStopped ()){}

return 0; //Funktionsende von main()
}

viewerA (pcl::visualization::PCLVisualizer& viewer)
{
//dies ist die eigentliche PCLVisualizer Klasse, hier können Objekte/Geometrien erstellt und
//dargestellt, Hintergrundfarben geändert, Prozessierungen durchgeführt werden usw.
}
```

VoxelGrid (Typisches Beispiel des Funktionsablaufs):

```
//1. VoxelGrid Objekt erstellen, welches die Punktwolke downsampled(Punktdichte verringert) und filtert
pcl::VoxelGrid<sensor_msgs::PointCloud2> sor;
//2. Input setzen
sor.setInputCloud (cloud_input);
//3. Parameter setzen (das ist nur ein Parameterbeispiel für die Blattgröße/Gridgröße)
sor.setLeafSize (0.25f, 0.25f, 0.25f);
//4. Ausführen -> filter, gespeichert in Outputwolke
sor.filter (*cloud_output);
```

Ausreißer entfernen:

```
//1. Objekt erstellen
pcl::StatisticalOutlierRemoval<pcl::PointXYZ> sor;
//2. Input setzen
sor.setInputCloud (cloud);
//3.Parameter setzen, Nachbaranzahl + Schwellenwert
sor.setMeanK (50);
sor.setStddevMulThresh (1.0);
//4.Ausführung
sor.filter (*cloud_filtered);
```

Fläche segmentieren:

```
//1. Objekt erstellen
pcl::SACSegmentation<pcl::PointXYZ> seg;
//Input wird später hinzugefügt
//3.Parameter setzen, Modelltyp, Methode, Schwellenwert
seg.setModelType (pcl::SACMODEL_PLANE);
seg.setMethodType (pcl::SAC_RANSAC
seg.setDistanceThreshold (0.5);
//eigentliche Durchführung erfolgt in einer Schleife, da durch Segmentierung jeweils nur eine Fläche
//gefunden wird – folglich gefundene Fläche speichern und nächste suchen

//Hilfsvariablen für Gesamtgröße der Ursprungswolke
nr_points = (int) cloud ->points.size ();

//Durchlaufe Schleife solange, wie mindestens 10% aus originaler Wolke enthalten sind
while (cloud ->points.size () > 0.01 * nr_points)
// je kleiner, desto mehr kleine Flächen können noch erkannt werden
         {

        //2.Input setzen
        seg.setInputCloud (cloud);
        //4.Ausführen, hierbei erhält man die Punktwolke und die Koeffizienten für die
        //Beschreibung der Ebene
        seg.segment (*inliers, *coefficients);

        //darauffolgend können die gefunden Flächenpunkte als eine neue Punktwolke gespeichert,
        //sowie aus der Urprungswolke cloud gelöscht werden, damit die Schleife sinnvoll durchlaufen
        //kann

        }
```

9. Einrichten von PCL und C++

Visual Studio 2008 oder 2010 – 32/64bit (empfohlen, da hier PCL 1.6 Installer existiert) installieren
→ kostenlose Version via **Dreamspark** für Windows (siehe Links im Abschnitt 10)

PCL mit entsprechendem All-In-One Installer downloaden und ausführen. Dabei wird **OpenNI** nicht installiert (befindet sich im installierten PCL Ordner und muss extra ausgeführt werden), ebenso muss **Qt** extra installiert werden (Link auf Downloadseite)

(Alternativ kann PCL auch von der Source her erstellt werden und selbstverständlich auch für alle weiteren gängigen Betriebssysteme)

Entscheiden zwischen Debug und Release Modus – für Anfäner ist der Release Modus zu empfehlen – da schneller – Debug muss beherrscht und weiter konfiguriert werden (Rechtsklick auf das Projekt->Properties und im Kopf „Configuration“ und „Platform“ auswählen – Weiterhin den Hinweis zum Debug und dem PCL-All-In-One Installer auf der Downloadseite beachten)

Setzen der Headerfiles, Libraries und Dependencies:
→Rechtsklick Projekt-> Properties-> C/C++-> General -> Additional Include Directories
→ Rechtsklick Projekt-> Properties-> Linker-> General -> Additional Library Directories
→ Rechtsklick Projekt-> Properties-> Linker-> Input -> Additional Dependencies
(im Anhang/Abschnitt 11 befinden sich Hinweise zum Konfigurieren dieser 3 Parameter)

Durch Fehleranzeige im Programmcode kann auf fehlende Headerfiles und Libraries geschlossen werden. Für die Dependencies ist es weniger ersichtlich (dabei sollen die gegebenen Dependencies aus dem Anhang/Abschnitt 11 helfen). Leider erkennt man Dependencies auch nicht in den von PCL gegebenen Tutorials.

Nach der vollständigen Konfiguration, kann mit dem Programmieren begonnen werden.

Viel Spaß!

10. Wichtige Links und Dokumentationen

C++

VisualStudio (Trial): http://www.microsoft.com/visualstudio/eng/downloads
VisualStudio (Full) https://www.dreamspark.com

C++ Beispiele: http://www.functionx.com/cpp/examples/index.htm
Allg. über C++: http://www.willemer.de/informatik/cpp/aufteil.htm

PCL

PCL: http://pointclouds.org/
PCL Win Installer: http://www.pointclouds.org/downloads/windows.html
(OpenNI und Qt müssen trotz AIO Installer installiert werden)
PCL Unix Installation: http://www.pointclouds.org/downloads/linux.html
PCL Mac OS: http://www.pointclouds.org/downloads/macosx.html

PCL Doku: http://pointclouds.org/documentation/
PCL Tutorials: http://pointclouds.org/documentation/tutorials
PCL Mailinglist: http://www.pcl-users.org/
IRC: #pcl on irc.oftc.net

Als Grundlage (Lesen und Schreiben von PCD):
http://pointclouds.org/documentation/tutorials/writing_pcd.php#writing-pcd
http://pointclouds.org/documentation/tutorials/reading_pcd.php#reading-pcd

11. Anhang: Konfigurationsparameter

Eingabe für Include Directories:

<path_to_PCL>\PCL1.6.0\3rdParty\Qt4.8.4\include;
<path_to_PCL>\PCL1.6.0\3rdParty\VTK\include\vtk-5.8;
<path_to_PCL>\PCL1.6.0\3rdParty\VTK\include;
<path_to_PCL>\PCL1.6.0\3rdParty\Qhull\include;
<path_to_PCL>\PCL1.6.0\3rdParty\FLANN\include;
<path_to_PCL>\PCL1.6.0\3rdParty\Eigen\include;
<path_to_PCL>\PCL1.6.0\3rdParty\Boost\include;
<path_to_PCL>\PCL1.6.0\include\pcl-1.6;
<path_to_PCL>\OpenNI\Include;

Eingabe für Library Directories:

<path_to_PCL>\PCL1.6.0\3rdParty\Qt4.8.4\lib
<path_to_PCL>\PCL1.6.0\3rdParty\VTK\lib\vtk-5.8
<path_to_PCL>\PCL1.6.0\3rdParty\VTK\lib
<path_to_PCL>\PCL1.6.0\3rdParty\Qhull\lib
<path_to_PCL>\PCL1.6.0\3rdParty\OpenNI
<path_to_PCL>\PCL1.6.0\3rdParty\FLANN\lib
<path_to_PCL>\PCL1.6.0\3rdParty\Boost\lib
<path_to_PCL>\PCL1.6.0\lib

Eingabe für Dependencies:

-------PCL----------
pcl_apps_release.lib
pcl_common_release.lib
pcl_features_release.lib
pcl_filters_release.lib
pcl_io_ply_release.lib
pcl_io_release.lib
pcl_kdtree_release.lib
pcl_keypoints_release.lib
pcl_octree_release.lib
pcl_registration_release.lib
pcl_sample_consensus_release.lib
pcl_search_release.lib
pcl_segmentation_release.lib
pcl_surface_release.lib
pcl_tracking_release.lib
pcl_visualization_release.lib

-------VTK----------
MapReduceMPI.lib
mpistubs.lib
QVTK.lib
vtkalglib.lib
vtkCharts.lib
vtkCommon.lib
vtkDICOMParser.lib
vtkexoIIc.lib
vtkexpat.lib
vtkFiltering.lib
vtkfreetype.lib
vtkftgl.lib
vtkGenericFiltering.lib
vtkGeovis.lib
vtkGraphics.lib
vtkhdf5.lib
vtkHybrid.lib
vtkImaging.lib
vtkInfovis.lib
vtkIO.lib
vtkjpeg.lib
vtklibxml2.lib
vtkmetaio.lib
vtkNetCDF.lib
vtkNetCDF_cxx.lib
vtkNetCDF-gd.lib
vtkpng.lib
vtkproj4.lib
vtkRendering.lib
vtksqlite.lib
vtksys.lib
vtktiff.lib
vtkverdict.lib
vtkViews.lib
vtkVolumeRendering.lib
vtkWidgets.lib
vtkzlib.lib

-------QT----------
phonon4.lib
QAxContainer.lib
QAxServer.lib

Qt3Support4.lib	QtHelp4.lib	QtSql4.lib
QtCLucene4.lib	qtmain.lib	QtSvg4.lib
QtCore4.lib	QtMultimedia4.lib	QtTest4.lib
QtDeclarative4.lib	QtNetwork4.lib	QtUiTools.lib
QtDesigner4.lib	QtOpenGL4.lib	QtWebKit4.lib
QtDesignerComponents4.lib	QtScript4.lib	QtXml4.lib
QtGui4.lib	QtScriptTools4.lib	QtXmlPatterns4.lib

Bzw. mit Pfandangabe (Der Pfad muss entsprechend des eigenen Pfades angepasst werden!!!)**:**

kernel32.lib;user32.lib;gdi32.lib;winspool.lib;shell32.lib;ole32.lib;oleaut32.lib;uuid.lib;comdlg32.lib;advapi32.lib;C:\Program Files\PCL 1.6.0\3rdParty\Boost\lib\libboost_system-vc100-mt-gd-1_49.lib;C:\Program Files\PCL 1.6.0\3rdParty\Boost\lib\libboost_filesystem-vc100-mt-gd-1_49.lib;C:\Program Files\PCL 1.6.0\3rdParty\Boost\lib\libboost_thread-vc100-mt-gd-1_49.lib;C:\Program Files\PCL 1.6.0\3rdParty\Boost\lib\libboost_date_time-vc100-mt-gd-1_49.lib;C:\Program Files\PCL 1.6.0\3rdParty\Boost\lib\libboost_iostreams-vc100-mt-gd-1_49.lib;C:\Program Files\PCL 1.6.0\lib\pcl_common_debug.lib;C:\Program Files\PCL 1.6.0\lib\pcl_octree_debug.lib;C:\Program Files\OpenNI\Lib64\openNI64.lib;C:\Program Files\PCL 1.6.0\3rdParty\VTK\lib\vtk-5.8\vtkCommon-gd.lib;C:\Program Files\PCL 1.6.0\3rdParty\VTK\lib\vtk-5.8\vtkRendering-gd.lib;C:\Program Files\PCL 1.6.0\3rdParty\VTK\lib\vtk-5.8\vtkHybrid-gd.lib;C:\Program Files\PCL 1.6.0\lib\pcl_io_debug.lib;C:\Program Files\PCL 1.6.0\3rdParty\FLANN\lib\flann_cpp_s-gd.lib;C:\Program Files\PCL 1.6.0\lib\pcl_kdtree_debug.lib;C:\Program Files\PCL 1.6.0\lib\pcl_search_debug.lib;C:\Program Files\PCL 1.6.0\lib\pcl_sample_consensus_debug.lib;C:\Program Files\PCL 1.6.0\lib\pcl_filters_debug.lib;C:\Program Files\PCL 1.6.0\lib\pcl_segmentation_debug.lib;C:\Program Files\PCL 1.6.0\lib\pcl_visualization_debug.lib;C:\Program Files\PCL 1.6.0\lib\pcl_features_debug.lib;C:\Program Files\PCL 1.6.0\3rdParty\Qhull\lib\qhullstatic_d.lib;C:\Program Files\PCL 1.6.0\lib\pcl_surface_debug.lib;C:\Program Files\PCL 1.6.0\lib\pcl_registration_debug.lib;C:\Program Files\PCL 1.6.0\lib\pcl_keypoints_debug.lib;C:\Program Files\PCL 1.6.0\lib\pcl_tracking_debug.lib;C:\Program Files\PCL 1.6.0\lib\pcl_apps_debug.lib;C:\Program Files\PCL 1.6.0\3rdParty\VTK\lib\vtk-5.8\vtkRendering-gd.lib;C:\Program Files\PCL 1.6.0\3rdParty\VTK\lib\vtk-5.8\vtkGraphics-gd.lib;C:\Program Files\PCL 1.6.0\3rdParty\VTK\lib\vtk-5.8\vtkverdict-gd.lib;C:\Program Files\PCL 1.6.0\3rdParty\VTK\lib\vtk-5.8\vtkImaging-gd.lib;C:\Qt\4.8.0\lib\QtGuid4.lib;C:\Qt\4.8.0\lib\QtSqld4.lib;C:\Qt\4.8.0\lib\QtCored4.lib;C:\Program Files\PCL 1.6.0\3rdParty\VTK\lib\vtk-5.8\vtkIO-gd.lib;C:\Program Files\PCL 1.6.0\3rdParty\VTK\lib\vtk-5.8\vtkFiltering-gd.lib;C:\Program Files\PCL 1.6.0\3rdParty\VTK\lib\vtk-5.8\vtkCommon-gd.lib;C:\Program Files\PCL 1.6.0\3rdParty\VTK\lib\vtk-5.8\vtkDICOMParser-gd.lib;C:\Program Files\PCL 1.6.0\3rdParty\VTK\lib\vtk-5.8\vtkNetCDF_cxx-gd.lib;C:\Program Files\PCL 1.6.0\3rdParty\VTK\lib\vtk-5.8\vtkmetaio-gd.lib;C:\Program Files\PCL 1.6.0\3rdParty\VTK\lib\vtk-5.8\vtksys-gd.lib;ws2_32.lib;comctl32.lib;wsock32.lib;C:\Program Files\PCL 1.6.0\3rdParty\VTK\lib\vtk-5.8\vtksqlite-gd.lib;C:\Program Files\PCL 1.6.0\3rdParty\VTK\lib\vtk-5.8\vtkpng-gd.lib;C:\Program Files\PCL 1.6.0\3rdParty\VTK\lib\vtk-5.8\vtktiff-gd.lib;C:\Program Files\PCL 1.6.0\3rdParty\VTK\lib\vtk-5.8\vtkzlib-gd.lib;C:\Program Files\PCL 1.6.0\3rdParty\VTK\lib\vtk-5.8\vtkjpeg-gd.lib;C:\Program Files\PCL 1.6.0\3rdParty\VTK\lib\vtk-5.8\vtkexpat-gd.lib;C:\Program Files\PCL 1.6.0\3rdParty\VTK\lib\vtk-5.8\vtkftgl-gd.lib;C:\Program Files\PCL 1.6.0\3rdParty\VTK\lib\vtk-5.8\vtkfreetype-

gd.lib;opengl32.lib;C:\Program Files\PCL 1.6.0\3rdParty\VTK\lib\vtk-5.8\vtkexoIIc-gd.lib;C:\Program Files\PCL 1.6.0\3rdParty\VTK\lib\vtk-5.8\vtkNetCDF-gd.lib;vfw32.lib

Fragen, Anregungen etc. bezüglich der PCL Konfiguration o.ä. an bernd.grafe@tu-dresden.de