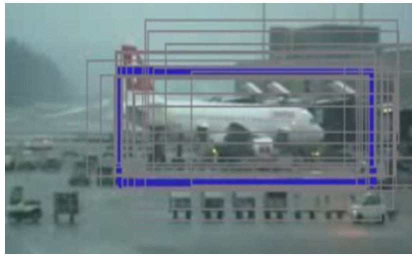
UNIVERSITÀ DEGLI STUDI DELLA BASILICATA

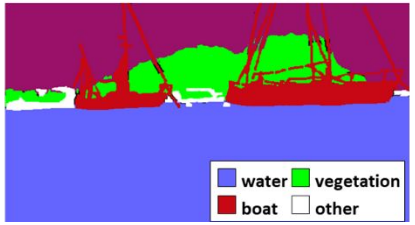


Corso di Sistemi Informativi
A.A. 2018/19

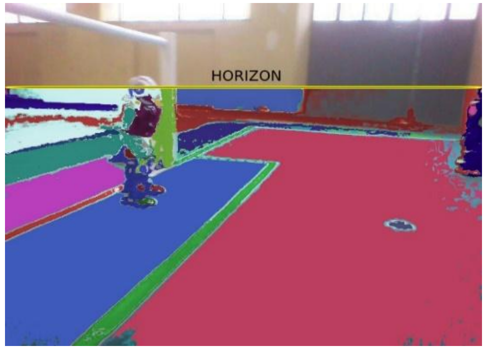
Docente
Domenico Daniele Bloisi

Visualizzazione dati 3D pcl





Giugno 2019

References and Credits

Questo materiale deriva da:

Alberto Pretto – Sapienza Università di Roma
Introduction to PCL: The Point Cloud Library
Basic topics

http://www.dis.uniroma1.it/~pretto/download/pcl_intro.pdf

Gestione dati 2D

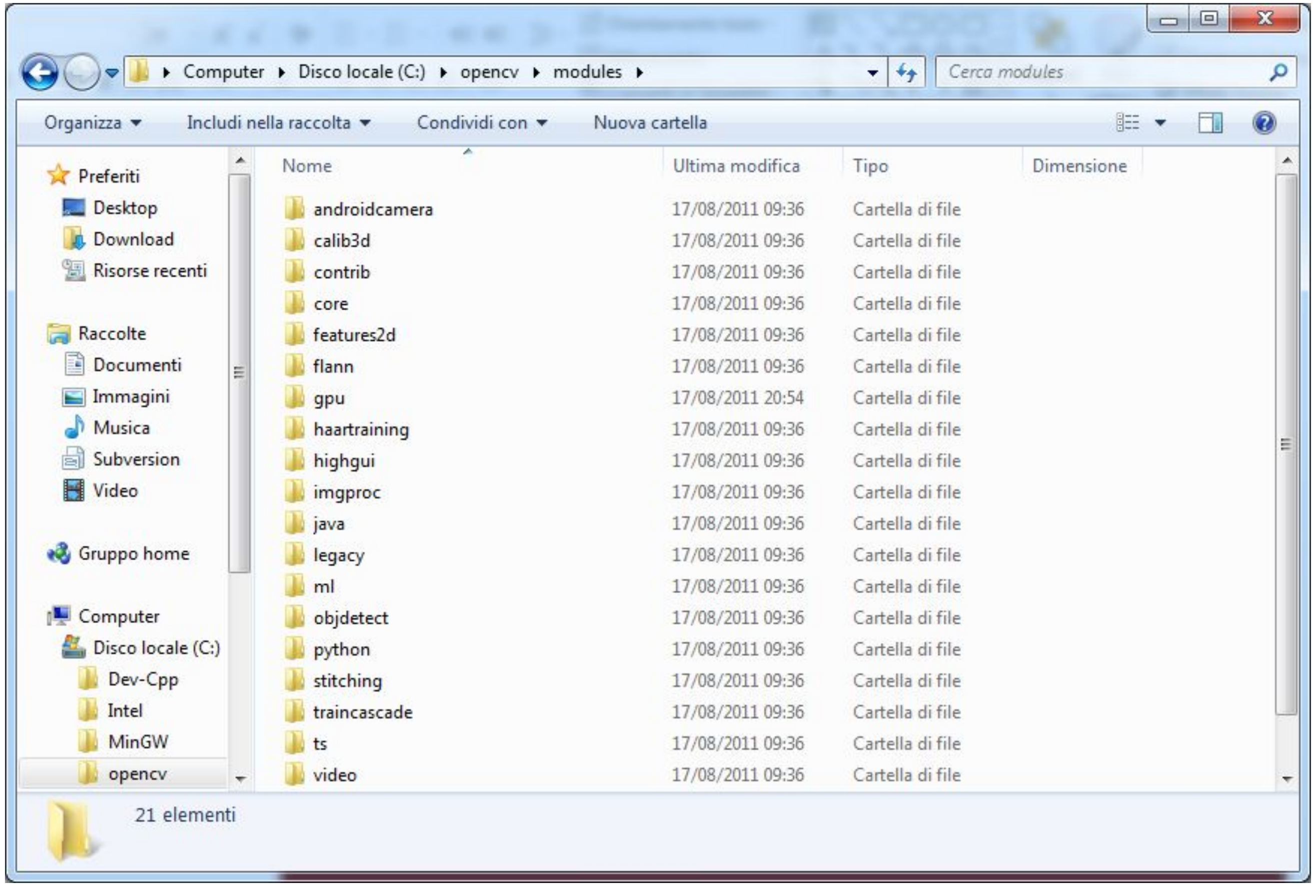
OpenCV (Open Source Computer Vision) is a library of programming functions for realtime computer vision

- BSD Licensed - free for commercial use
- C++, C, Python and Java (Android)
- interfaces
- Supports Windows, Linux, Android, iOS
- and Mac OS
- More than 2500 optimized algorithms

Moduli OpenCV

OpenCV has a modular structure

- core
- imgproc
- video
- calib3d
- features2d
- objdetect
- highgui
- gpu
- ...



OpenCV

Processamento delle immagini

core - a compact module defining basic data structures, including the dense multi-dimensional array Mat and basic functions used by all other modules.

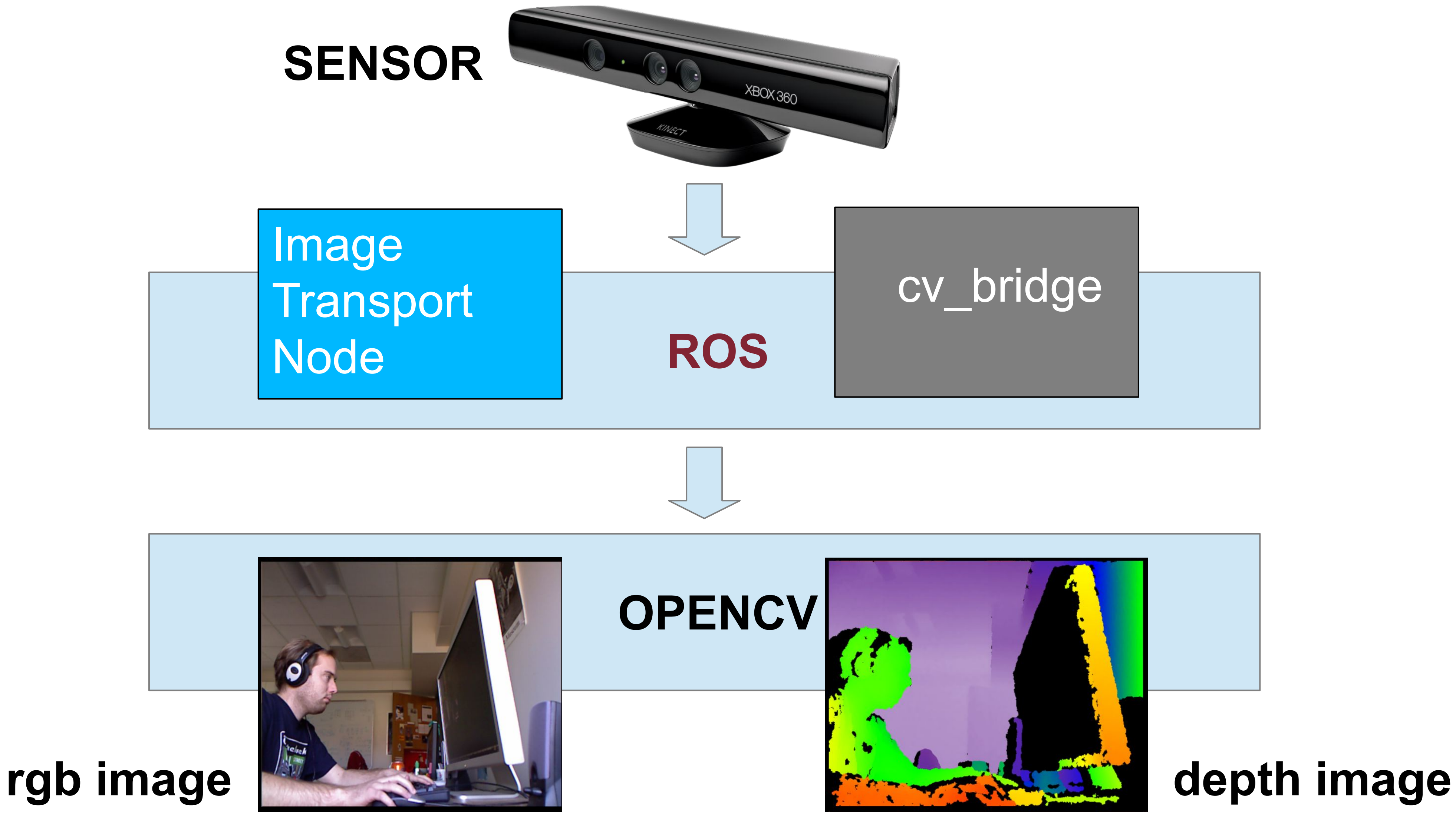
imgproc - an image processing module that includes linear and non-linear image filtering, geometrical image transformations (resize, affine and perspective warping, generic table-based remapping), color space conversion, histograms, and so on.

features2d - salient feature detectors, descriptors, and descriptor matchers.

highgui - an easy-to-use interface to video capturing, image and video codecs, as well as simple UI capabilities.

objdetect - detection of objects and instances of the predefined classes (for example, faces, eyes, mugs, people, cars, and so on).

OpenCV e ROS


SENSOR
XBOX 360
Image
Transport
Node
ROS
cv_bridge
OPENCV
rgb image
depth image

Gestione dati 3D

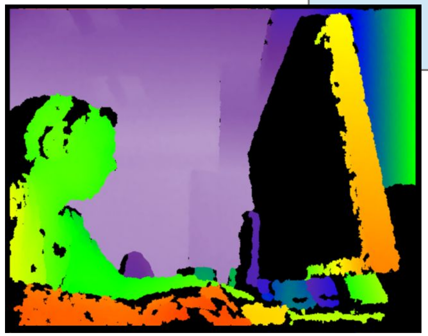
The Point Cloud Library (PCL) is a standalone, large scale, open project for 2D/3D image and point cloud processing

- Collection of Libraries focused on Point Cloud processing
- More than 450 developers/contributors
- Over 60 Tutorials and many examples
- BSD Licensed - free for commercial use

PCL e ROS

SENSOR

ROS



OPENCV

PCL

Point cloud: a definition

- A point cloud is a data structure used to represent a collection of multi-dimensional points
- It is commonly used to represent three-dimensional data

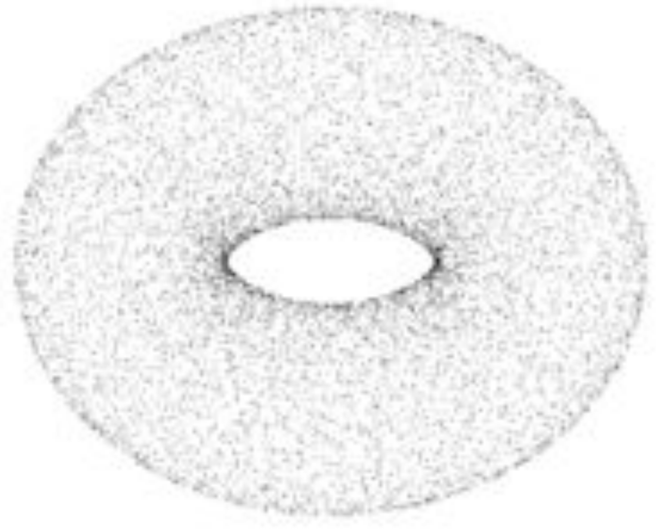


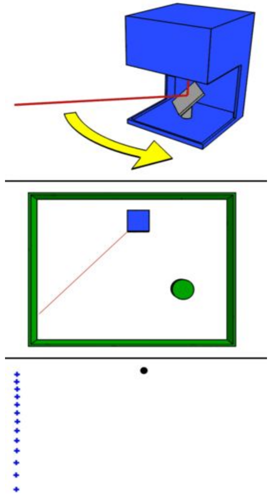
image from
https://en.wikipedia.org/wiki/Point_cloud

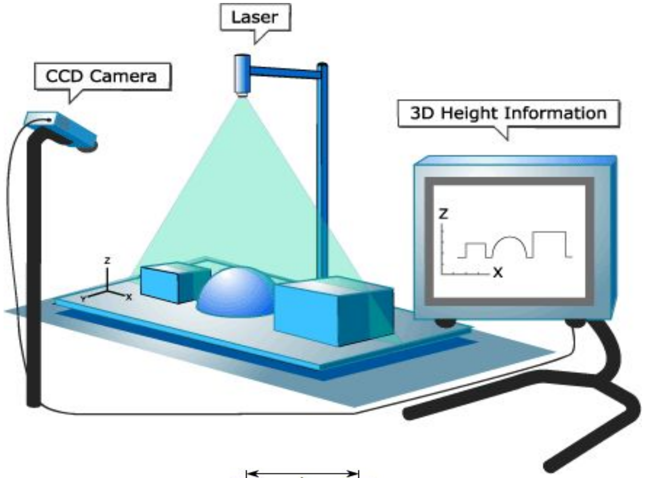
Point cloud: a definition

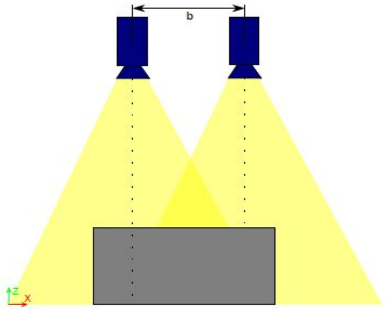
- The points in the point cloud usually represent the X, Y, and Z geometric coordinates of a sampled surface
- Each point can hold additional information: RGB colors, intensity values, etc...

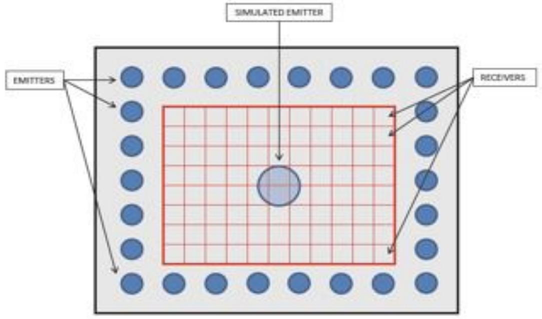
Where do they come from?

- 2/3D Laser scans
- Laser triangulation
- Stereo cameras
- RGB-D cameras
- Structured light cameras
- Time of flight cameras



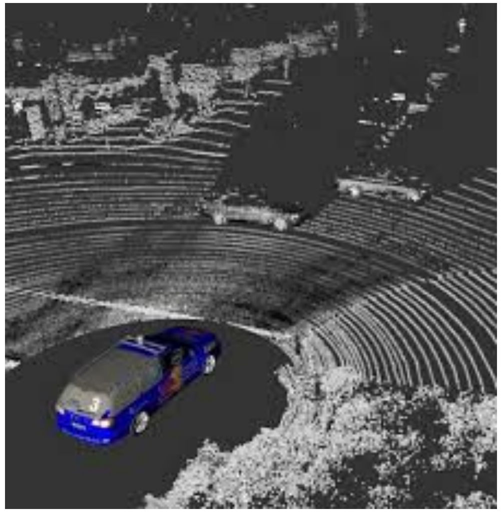


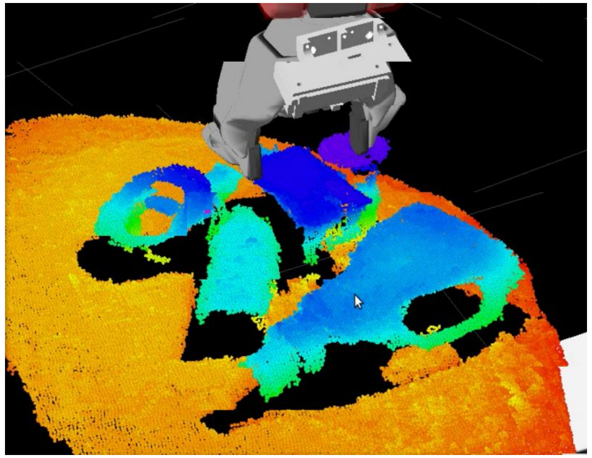




Point clouds in robotics

- Navigation / Obstacle avoidance
- Object recognition and registration
- Grasping and manipulation





Point Cloud Library

→ pointclouds.org

- The Point Cloud Library (PCL) is a standalone, large scale, open source (C++) library for 2D/3D image and point cloud processing

- PCL is released under the terms of the BSD license and thus free for commercial and research use

Point Cloud Library

- PCL provides the 3D processing pipeline for ROS, so you can also get the perception pcl stack and still use PCL standalone

- Among others, PCL depends on Boost, Eigen, OpenMP,...

PCL Basic Structures: PointCloud

A PointCloud is a templated C++ class that contains the following data fields:

- **width (int)** - specifies the width of the point cloud dataset in the number of points.
  → the total number of points in the cloud (equal with the number of elements in points) for unorganized datasets
  → the width (total number of points in a row) of an organized point cloud dataset
- **height (int)** - Specifies the height of the point cloud dataset in the number of points
  → set to 1 for unorganized point clouds
  → the height (total number of rows) of an organized point cloud dataset
- **points (std::vector <PointT>)** - Contains the data array where all the points of type PointT are stored.

PointCloud vs PointCloud2

We distinguish between two data formats for the point clouds:

- **PointCloud<PointType>** with a specific data type (for actual usage in the code)

- **PointCloud2** as a general representation containing a header defining the point cloud structure (e.g., for loading, saving or sending as a ROS message)

- Conversion between the two frameworks is easy:
  ⟶pcl::fromROSMsg and pcl::toROSMsg

- Important: clouds are often handled using smart pointers, e.g.:
  ⟶**PointCloud<PointType>::Ptr cloud_ptr;**

Point Types

PointXYZ - float x, y, z
PointXYZI - float x, y, z, intensity
PointXYZRGB - float x, y, z, rgb
PointXYZRGBA - float x, y, z, uint32 t rgba
Normal - float normal[3], curvature
PointNormal - float x, y, z, normal[3], curvature

→ See pcl/include/pcl/point_types.h for more examples

CMakeLists.txt

```
project(pcl_test)
cmake_minimum_required (VERSION 2.8)
cmake_policy(SET CMP0015 NEW)

find_package(PCL 1.7 REQUIRED)
add_definitions(${PCL_DEFINITIONS})

include_directories(... ${PCL_INCLUDE_DIRS})
link_directories(... ${PCL_LIBRARY_DIRS})

add_executable(pcl_test pcl_test.cpp ...)
target_link_libraries(pcl_test ${PCL_LIBRARIES})
```

PCL structure

PCL is a collection of smaller, modular C++ libraries:

- **libpcl_features**: many 3D features (e.g., normals and curvatures, boundary points, moment invariants, principal curvatures, Point Feature Histograms (PFH), Fast PFH, ...)
- **libpcl_surface**: surface reconstruction techniques (e.g., meshing, convex hulls, Moving Least Squares, ...)
- **libpcl_filters**: point cloud data filters (e.g., downsampling, outlier removal, indices extraction, projections, ...)
- **libpcl_io**: I/O operations (e.g., writing to/reading from PCD (Point Cloud Data) and BAG files)
- **libpcl_segmentation**: segmentation operations (e.g.,cluster extraction, Sample Consensus model fitting, polygonal prism extraction, ...)
- **libpcl_registration**: point cloud registration methods (e.g., Iterative Closest Point (ICP), non linear optimizations, ...)
- **libpcl_range_image**: range image class with specialized methods

Point Cloud file format

Point clouds can be stored to disk as files, into the PCD (Point Cloud Data) format:

```
# Point Cloud Data (PCD) file format v.5
FIELDS x y z rgba
SIZE 4 4 4 4
TYPE F F F U
WIDTH 307200
HEIGHT 1
POINTS 307200
DATA binary
...<data>...
```

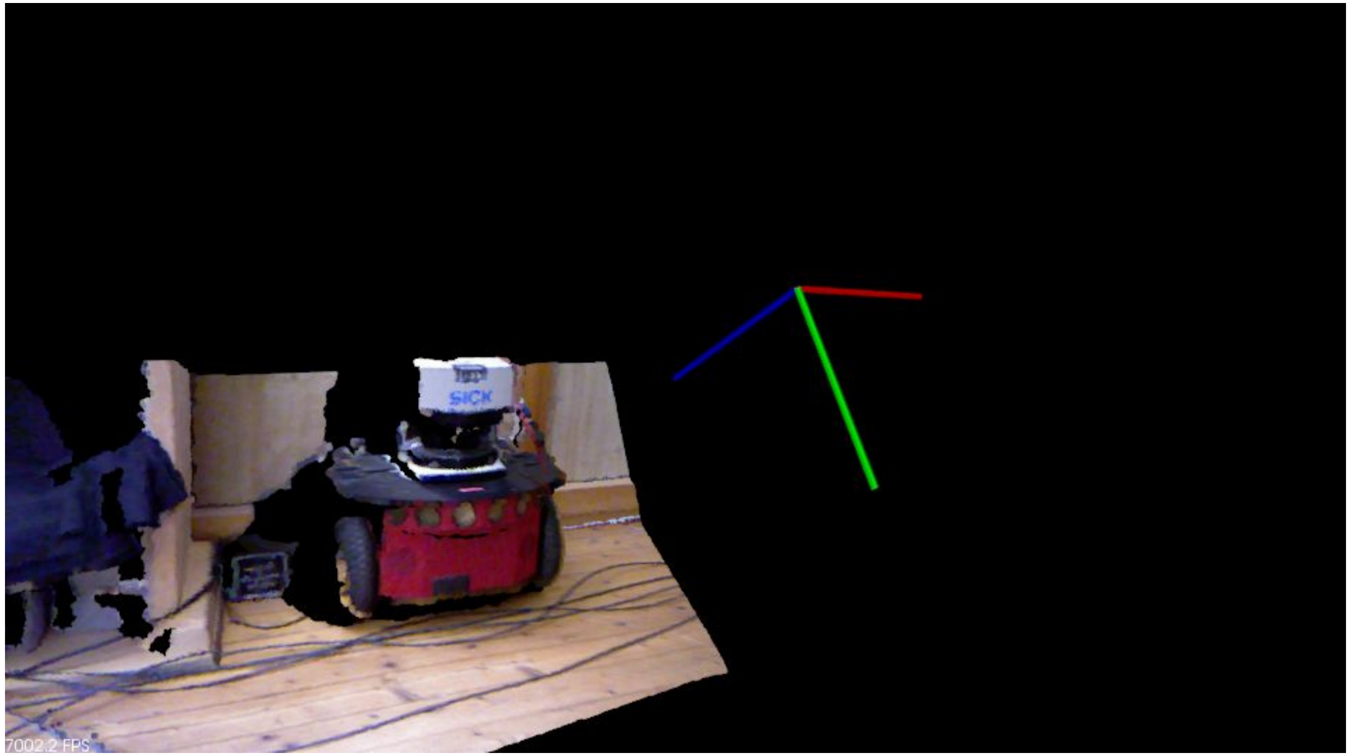
Funtions: `pcl::io::loadPCDFile` and `pcl::io::savePCDFile`

Example: create and save a PC

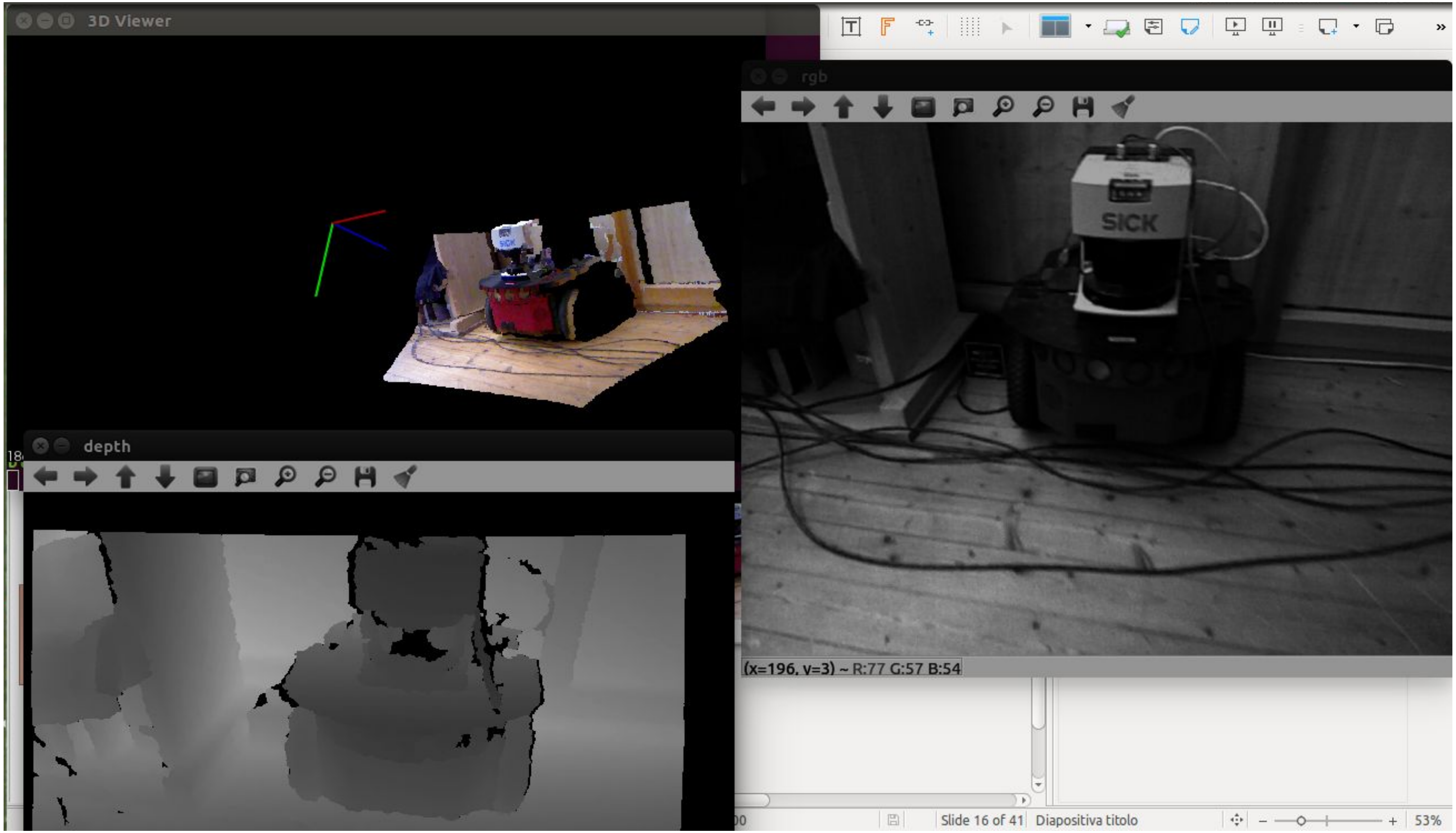
```
#include<pcl/io/pcd_io.h>
#include<pcl/point_types.h>

  //....

  pcl::PointCloud<pcl::PointXYZ>:: Ptr cloud_ptr(new pcl::PointCloud<pcl::PointXYZ>);
  cloud->width = 50;
  cloud->height = 1;
  cloud->isdense = false;
  cloud->points.resize(cloud.width*cloud.height);
  for(size_t i = 0; i < cloud.points.size(); i++){
    cloud->points[i].x = 1024*rand()/(RANDMAX+1.0f);
    cloud->points[i].y = 1024*rand()/(RANDMAX+1.0f);
    cloud->points[i].z = 1024*rand()/(RANDMAX+1.0f);
  }
  pcl::io::savePCDFileASCII("testpcd.pcd",*cloud);

  //....
```

Visualize a cloud

```
boost::shared_ptr<pcl::visualization::PCLVisualizer> viewer(new
                 pcl::visualization::PCLVisualizer("3D Viewer"));
viewer->setBackgroundColor(0, 0, 0);
viewer->addPointCloud<pcl::PointXYZ>(in_cloud, cloud_color, "Input cloud");
viewer->initCameraParameters();
viewer->addCoordinateSystem(1.0);
while(!viewer->wasStopped()) {
   viewer->spinOnce(1);
}
```



depth2cloud.cpp



Basic Module Interface

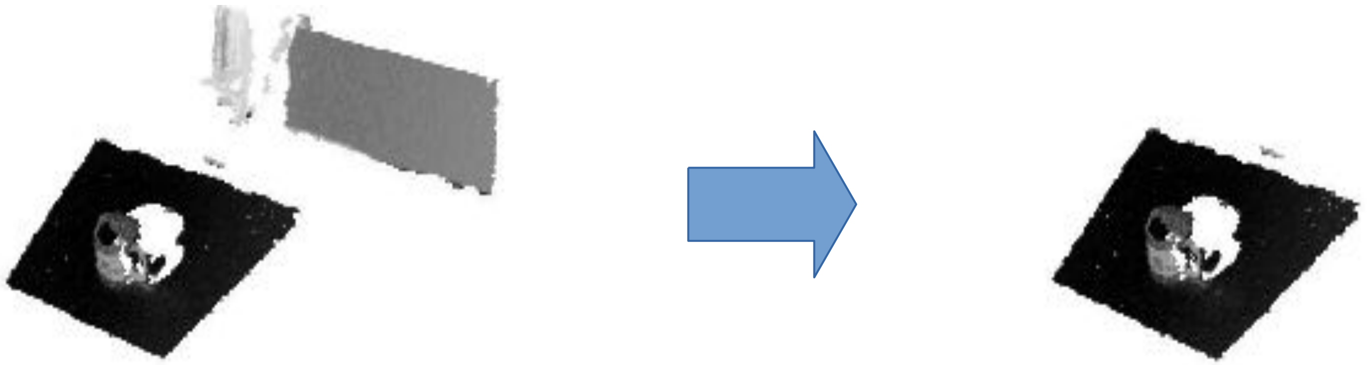
Filters, Features, Segmentation all use the same basic usage interface:

- use **setInputCloud()** to give the input
- set some parameters
- call **compute()** or **filter()** or **align()** or **...** to get the output

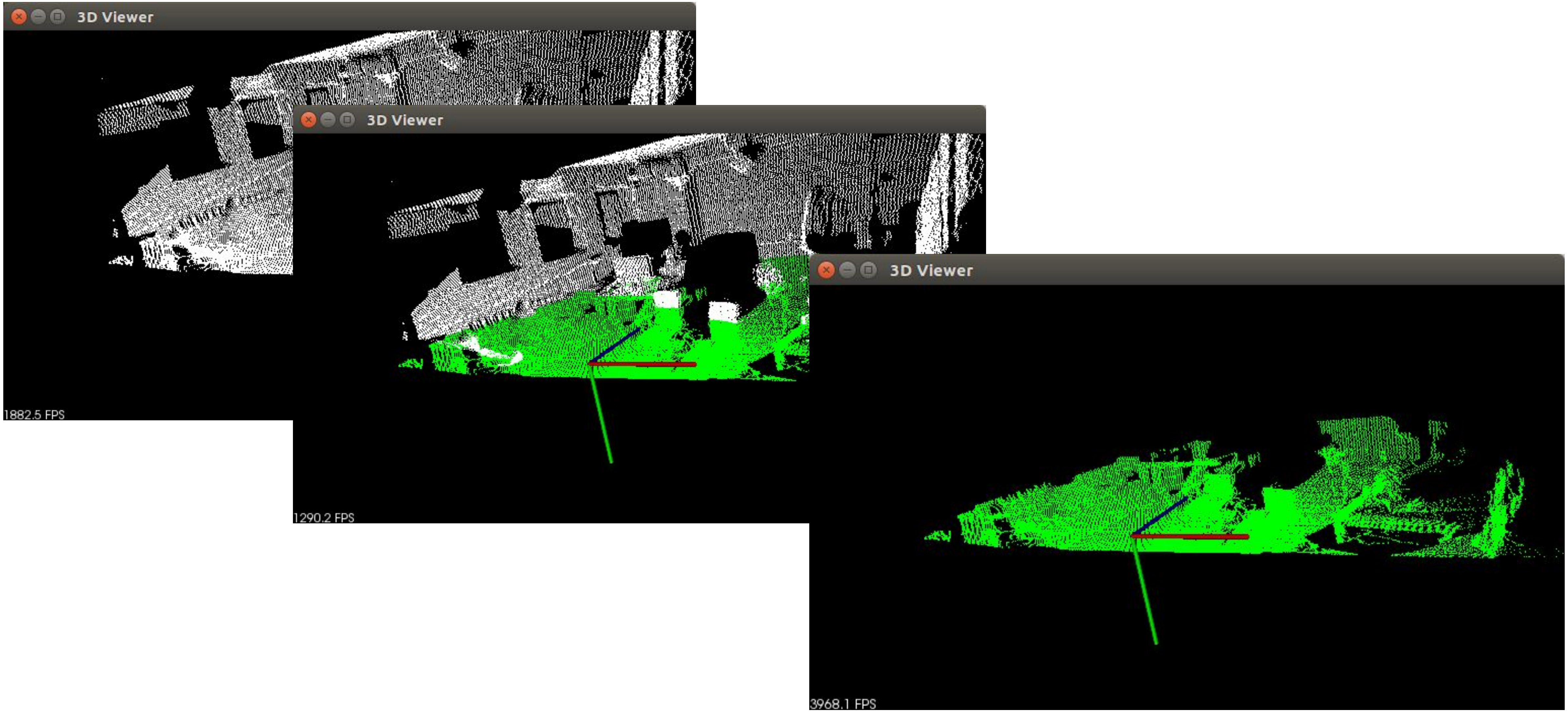
PassThrough Filter

Filter out points outside a specified range in one dimension.

```
pcl::PassThrough<T> pass_through;
pass_through.setInputCloud(in_cloud);
pass_through.setFilterLimits (0.0, 0.5);
pass_through.setFilterFieldName("z");
pass_through.filter(*cutted_cloud);
```



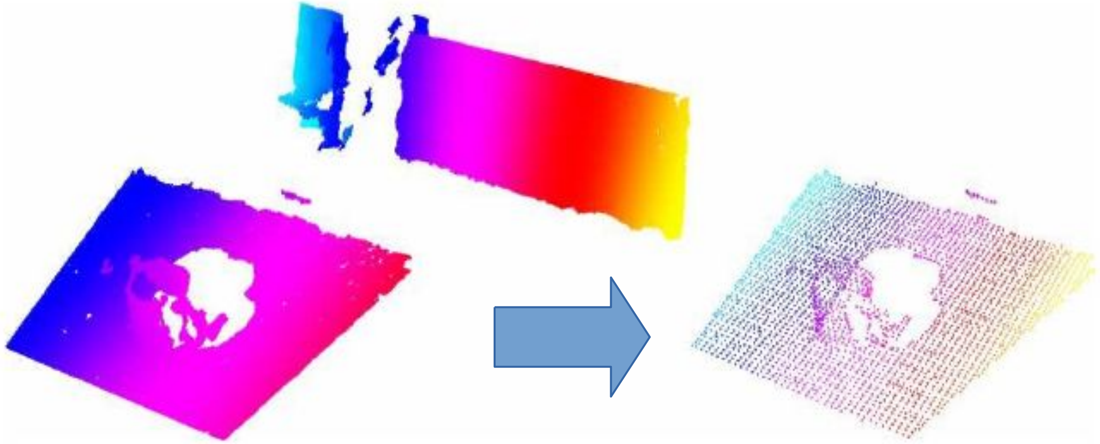
cloud_filters.cpp



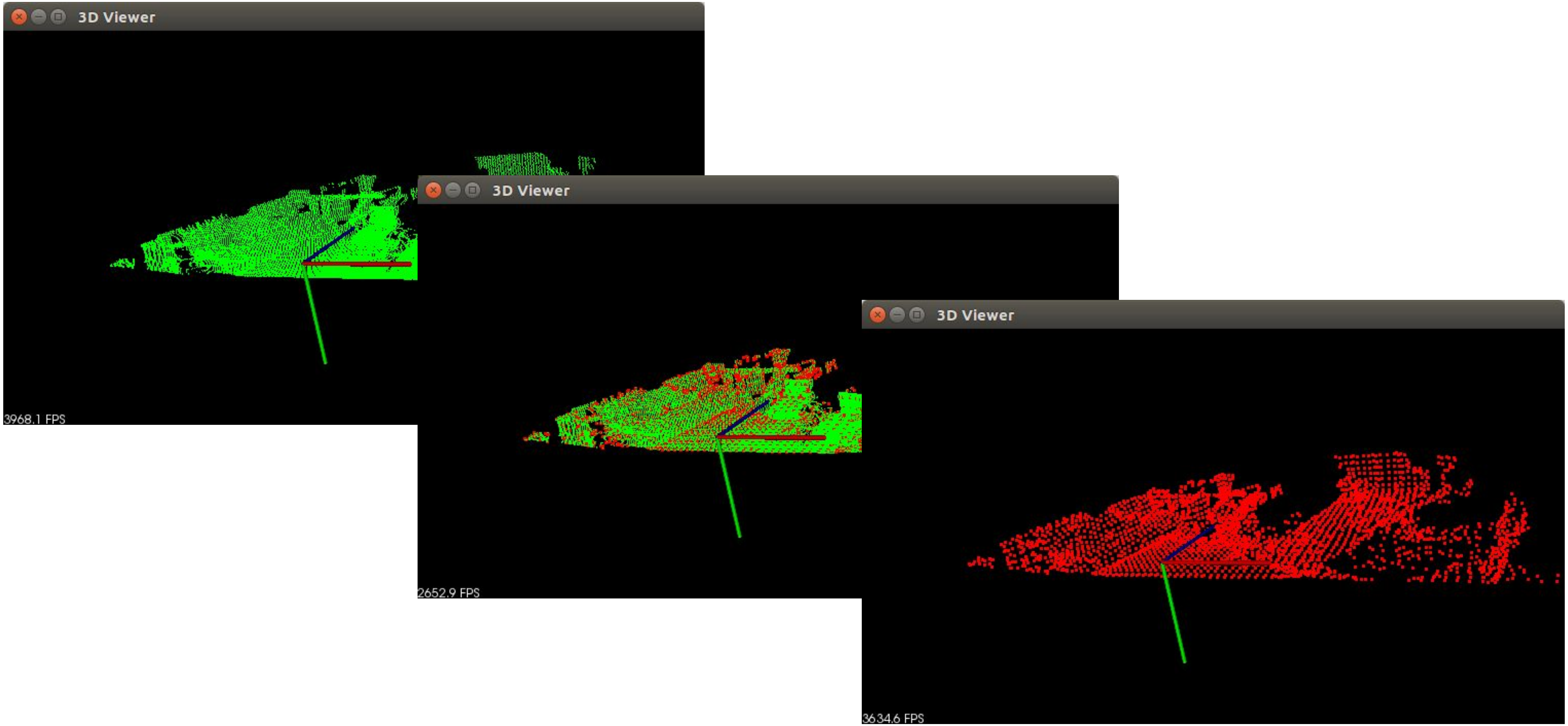
Downsampling

Voxelize the cloud to a 3D grid. Each occupied voxel is approximated by the centroid of the points inside it.

```
pcl::VoxelGrid<T> voxel_grid;
voxel_grid.setInputCloud(input_cloud);
voxel_grid.setLeafSize(0.01, 0.01, 0.01);
voxel_grid.filter(*subsamp_cloud);
```



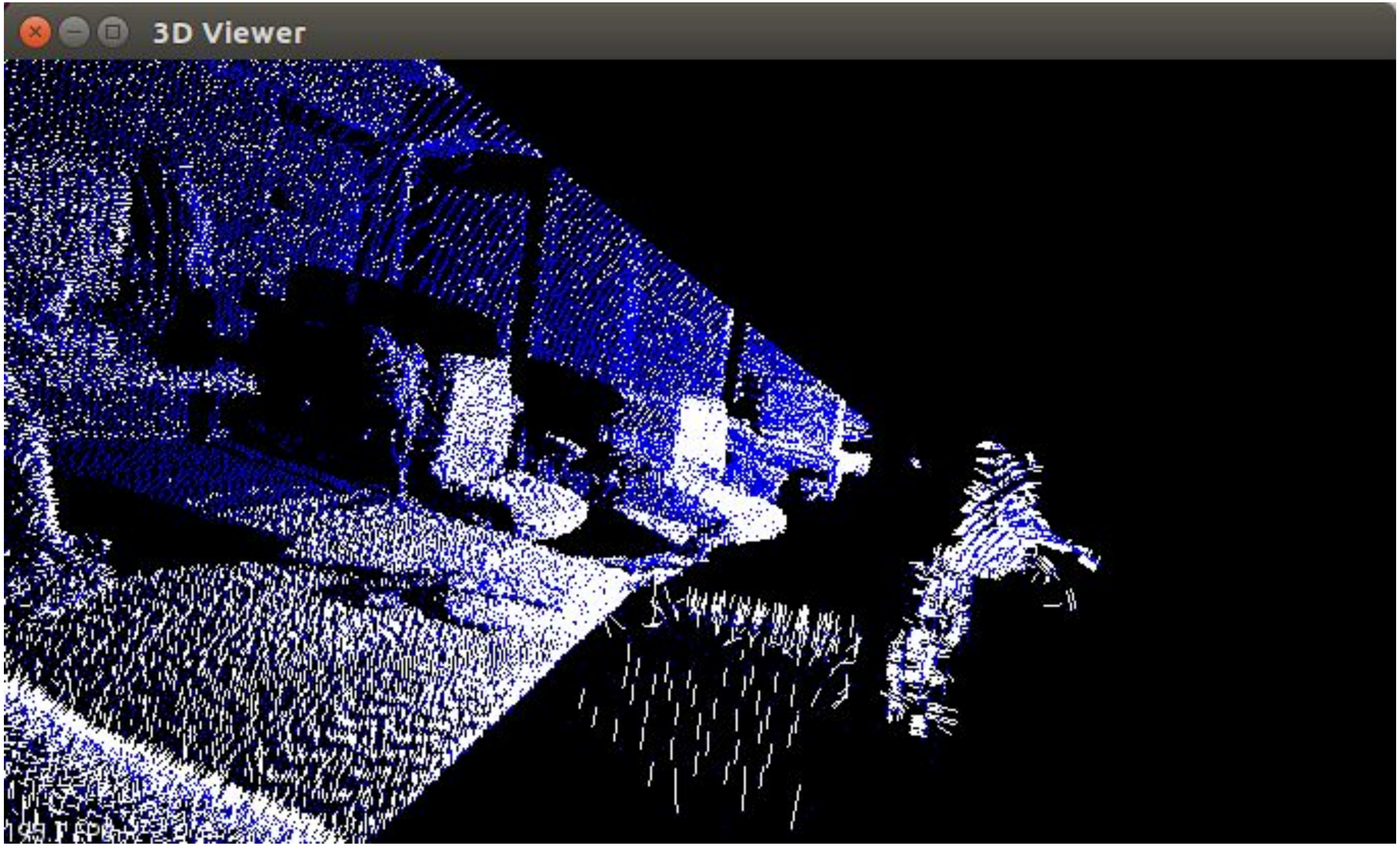
cloud_filters.cpp



Features example: normals

```
pcl::NormalEstimation<T, pcl::Normal> ne;
ne.setInputCloud(in_cloud);
pcl::search::KdTree<pcl::PointXYZ>::Ptr tree(new
                    pcl::search::KdTree<pcl::PointXYZ>());
ne.setSearchMethod(tree);
ne.setRadiusSearch(0.03);
ne.compute(*cloud_normals)
```

cloud_normals.cpp



Segmentation

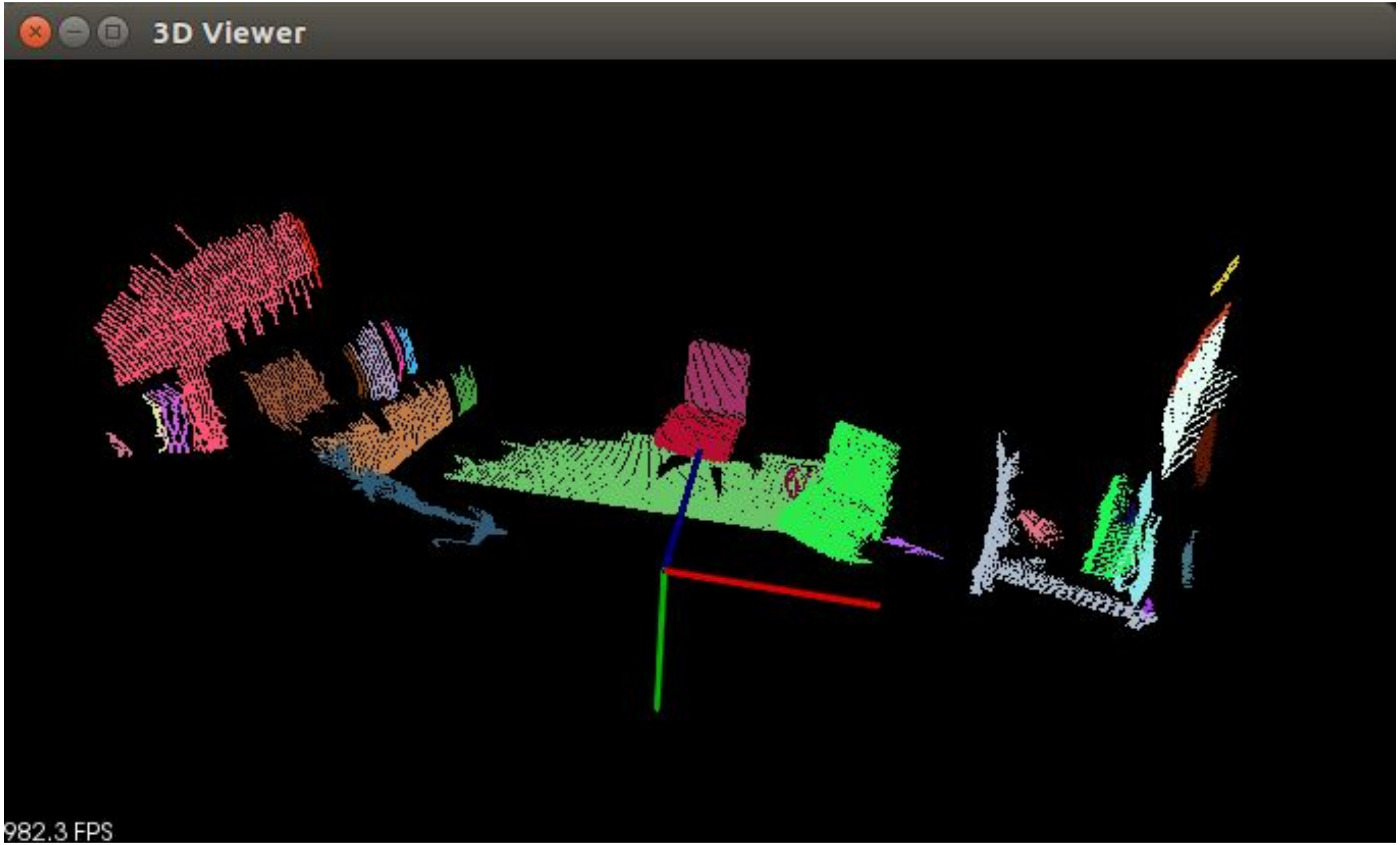
A clustering method divides an unorganized point cloud into smaller, correlated, parts

`EuclideanClusterExtraction` uses a distance threshold to the nearest neighbors of each point to decide if the two points belong to the same cluster.

Segmentation example

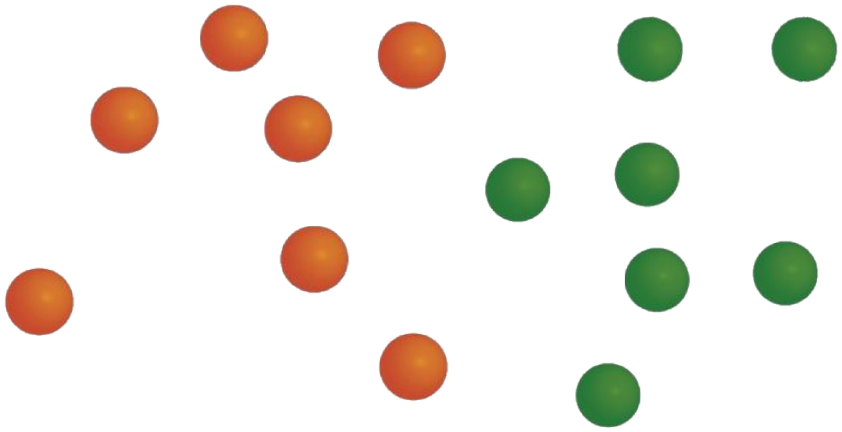
```
pcl::EuclideanClusterExtraction<T> ec;
ec.setInputCloud(in_cloud);
ec.setMinClusterSize(100);
ec.setClusterTolerance(0.05); //distance threshold
ec.extract(cluster_indices);
```

clustering.cpp



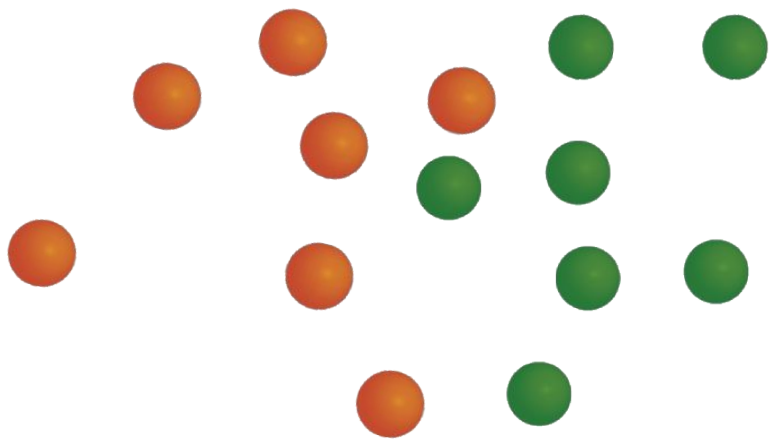
Point Cloud Registration

We want to find the translation and the rotation that maximize the overlap between two point clouds



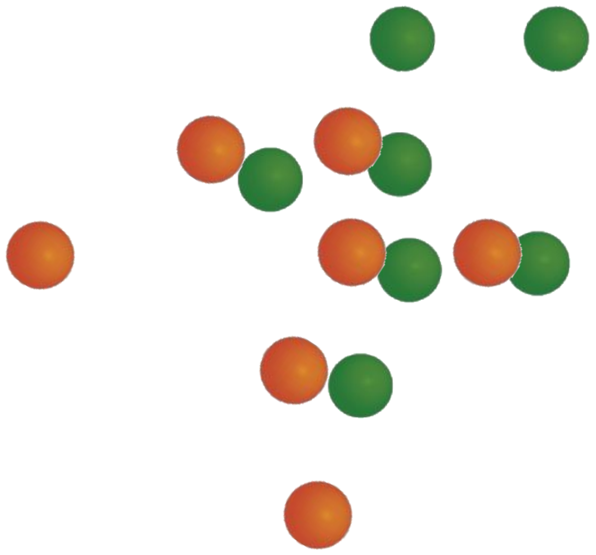
Point Cloud Registration

We want to find the translation and the rotation that maximize the overlap between two point clouds



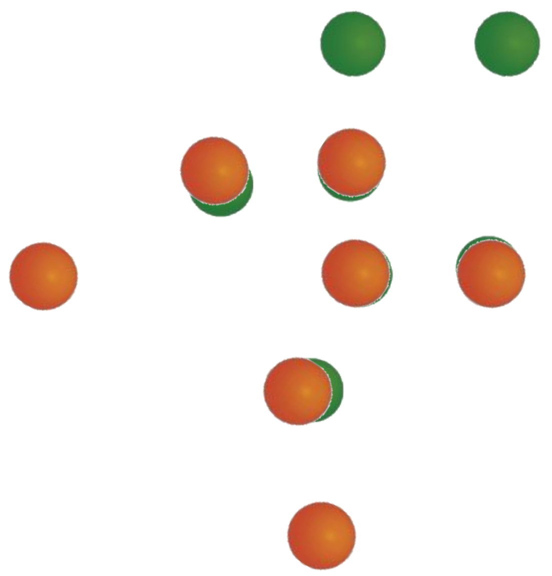
Point Cloud Registration

We want to find the translation and the rotation that maximize the overlap between two point clouds



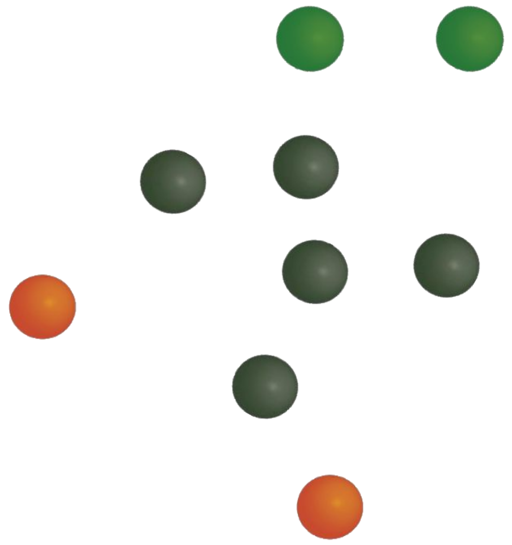
Point Cloud Registration

We want to find the translation and the rotation that maximize the overlap between two point clouds



Point Cloud Registration

We want to find the translation and the rotation that maximize the overlap between two point clouds



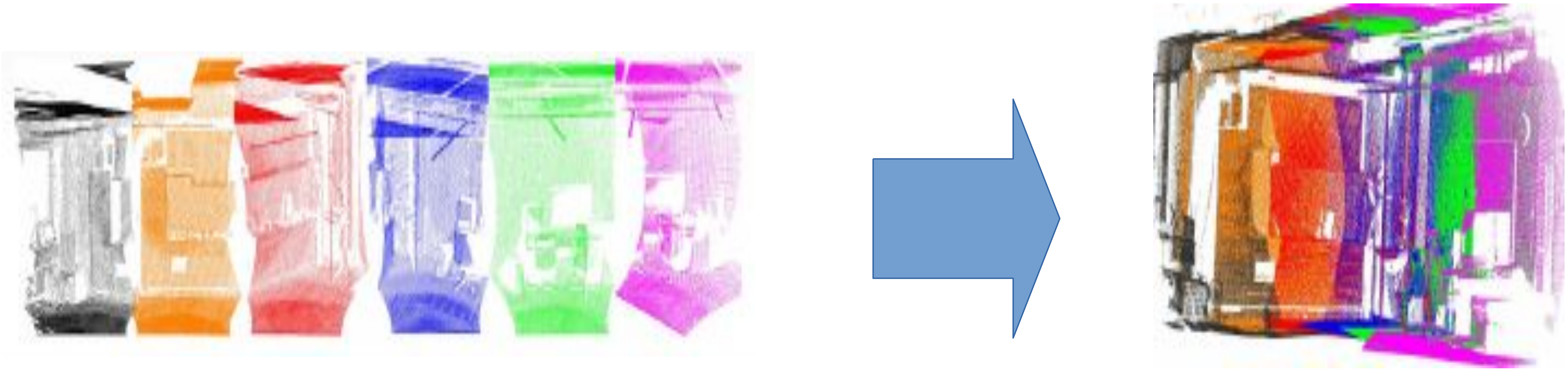
Iterative Closest Point

ICP iteratively revises the transformation (translation,rotation) needed to minimize the distance between the points of two raw scans

Input: points from two raw scans, initial estimation of the transformation, criteria for stopping the iteration

Output: refined transformation

Iterative Closest Point: Example



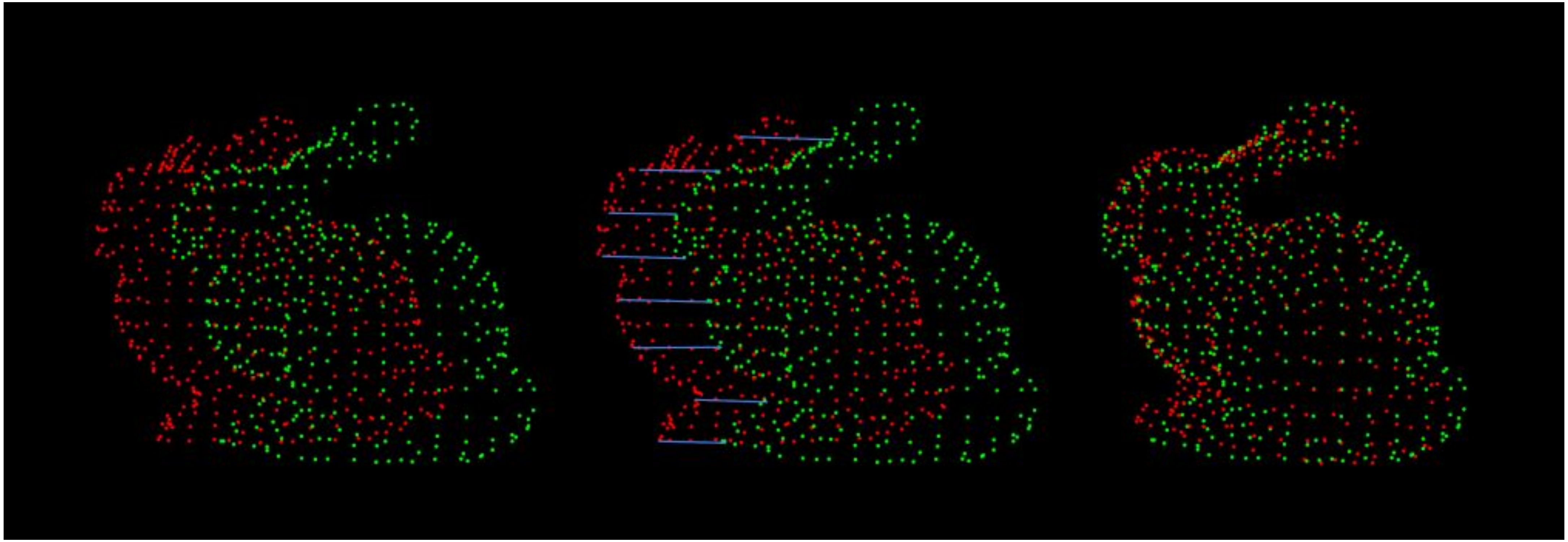
Iterative Closest Point: Algorithm

1. Associate points of the two cloud using 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)

Iterative Closest Point: Code

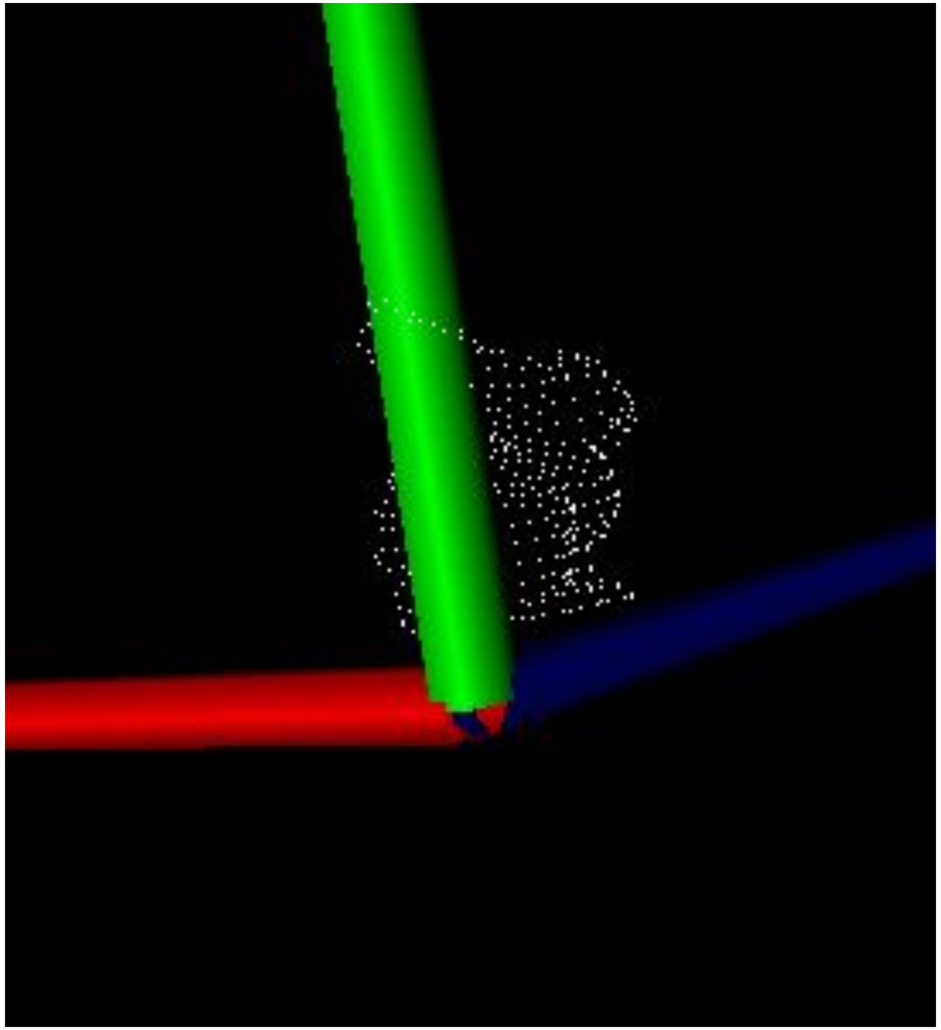
```
IterativeClosestPoint<PointXYZ, PointXYZ> icp;
// Set the input source and target
icp.setInputCloud(cloud_source);
icp.setInputTarget(cloud_target);
// Set the max correspondence distance to 5cm
icp.setMaxCorrespondenceDistance(0.05);
// Set the maximum number of iterations (criterion 1)
icp.setMaximumIterations(50);
// Set the transformation epsilon (criterion 2)
icp.setTransformationEpsilon(1e-8);
// Set the euclidean distance difference epsilon (criterion 3)
icp.setEuclideanFitnessEpsilon(1);
// Perform the alignment
icp.align(*cloud_source_registered);
// Align cloud_source to cloud_source_registered
Eigen::Matrix4f transformation = icp.getFinalTransformation();
```

Iterative Closest Point: Example



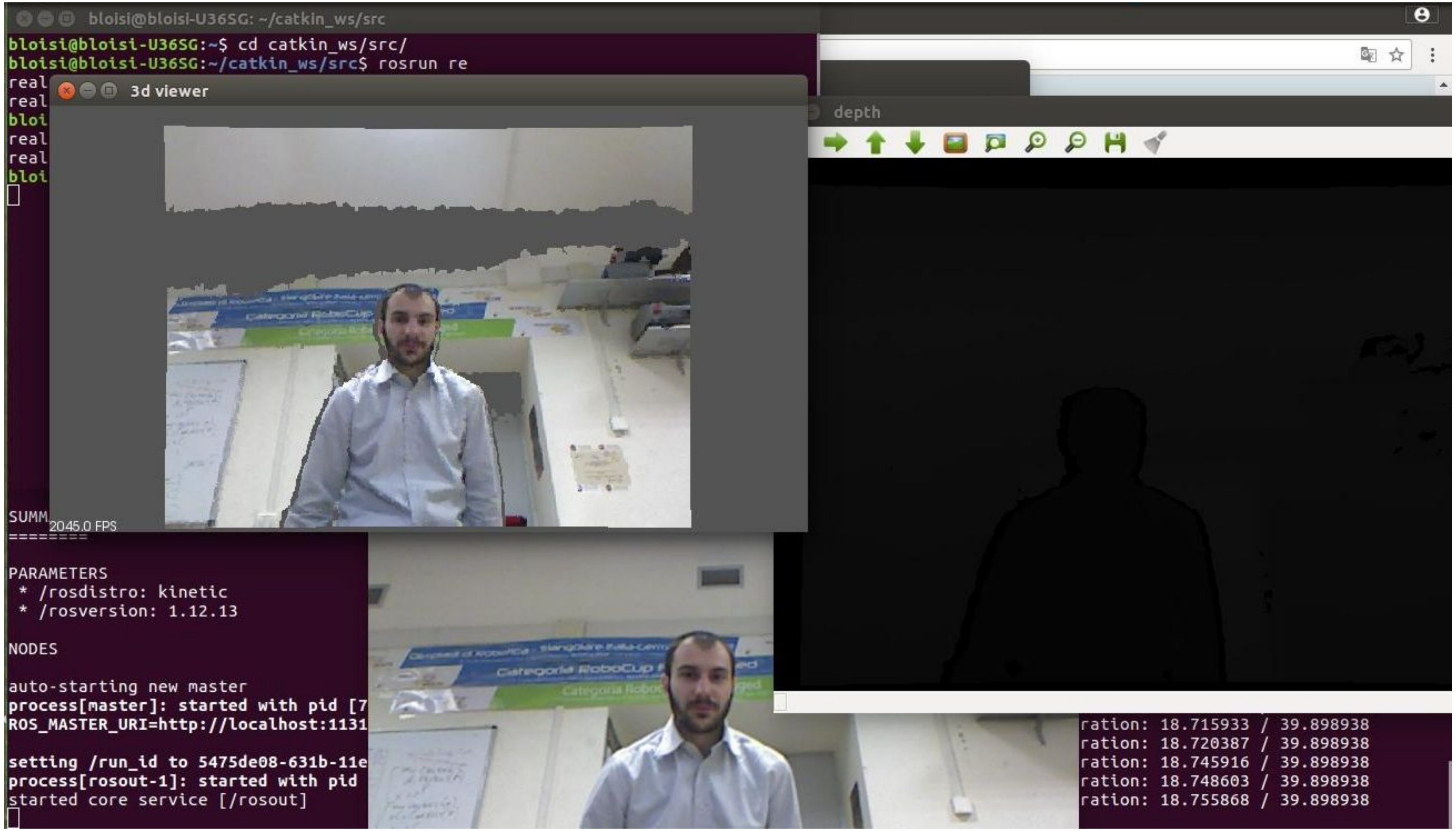
http://www.pointclouds.org/assets/rss2011/08_registration.pdf

icp.cpp



http://www.pointclouds.org/assets/rss2011/08_registration.pdf

basic 3D visualizer



https://github.com/dbloisi/basic_3d_visualizer

basic 3D visualizer: CMakeLists.txt

CMakeLists.txt (~/catkin_ws/src/basic_3d_visualizer) - gedit

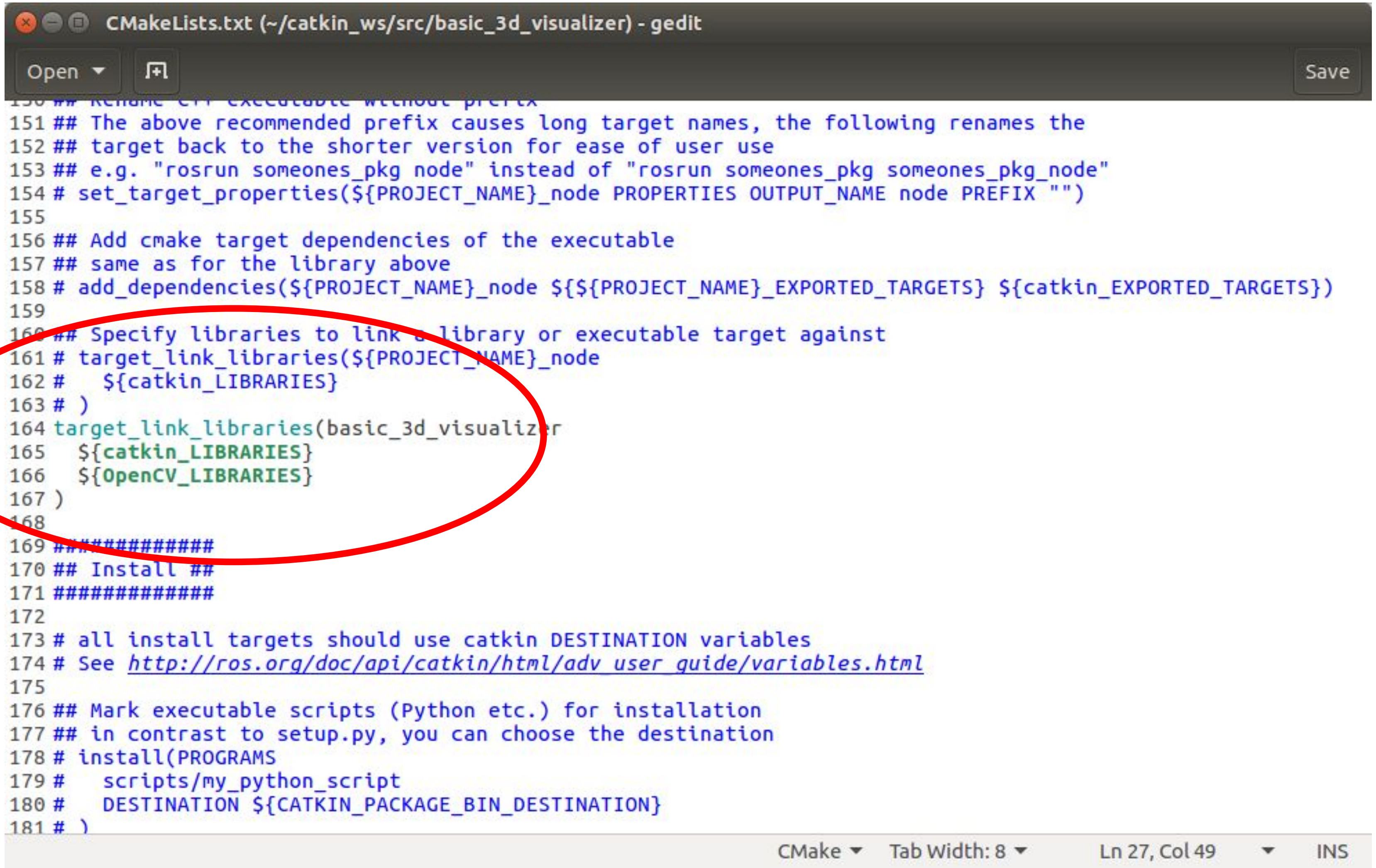
Open Save

```
cmake_minimum_required(VERSION 2.8.3)
project(basic_3d_visualizer)

## Compile as C++11, supported in ROS Kinetic and newer
# add_compile_options(-std=c++11)

## Find catkin macros and libraries
## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
## is used, also find other catkin packages
find_package(catkin REQUIRED COMPONENTS
  cv_bridge
  pcl_conversions
  pcl_ros
  roscpp
  sensor_msgs
  std_msgs
)

find_package(OpenCV REQUIRED
  COMPONENTS
  opencv_highgui
  opencv_objdetect
  CONFIG
)

## System dependencies are found with CMake's conventions
# find_package(Boost REQUIRED COMPONENTS system)


## Uncomment this if the package has a setup.py. This macro ensures
## modules and global scripts declared therein get installed
```

CMake Tab Width: 8 Ln 27, Col 49 INS

https://github.com/dbloisi/basic_3d_visualizer

basic 3D visualizer: CMakeLists.txt



CMakeLists.txt (~/catkin_ws/src/basic_3d_visualizer) - gedit

Open Save

```
## The above recommended prefix causes long target names, the following renames the
## target back to the shorter version for ease of user use
## e.g. "rosrun someones_pkg node" instead of "rosrun someones_pkg someones_pkg_node"
# set_target_properties(${PROJECT_NAME}_node PROPERTIES OUTPUT_NAME node PREFIX "")

## Add cmake target dependencies of the executable
## same as for the library above
# add_dependencies(${PROJECT_NAME}_node ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})

## Specify libraries to link a library or executable target against
# target_link_libraries(${PROJECT_NAME}_node
#   ${catkin_LIBRARIES}
# )
target_link_libraries(basic_3d_visualizer
  ${catkin_LIBRARIES}
  ${OpenCV_LIBRARIES}
)

#############
## Install ##
#############

# all install targets should use catkin DESTINATION variables
# See http://ros.org/doc/api/catkin/html/adv_user_guide/variables.html

## Mark executable scripts (Python etc.) for installation
## in contrast to setup.py, you can choose the destination
# install(PROGRAMS
#   scripts/my_python_script
#   DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
# )
```

CMake Tab Width: 8 Ln 27, Col 49 INS

https://github.com/dbloisi/basic_3d_visualizer

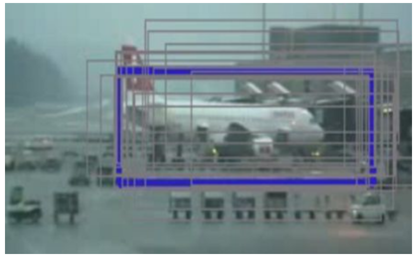
UNIVERSITÀ DEGLI STUDI
DELLA BASILICATA

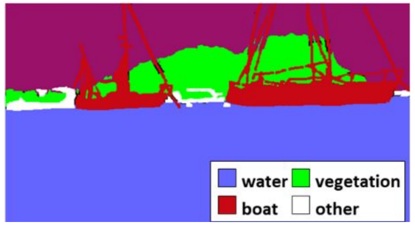


Corso di Sistemi Informativi
A.A. 2018/19

Docente
Domenico Daniele Bloisi

Visualizzazione dati 3D pcl





Giugno 2019