python-pcl Documentation

Release 0.3

Tooru Oonuma

Dec 27, 2018

Contents

This is the python-pcl documentation.

CHAPTER 1

python-pcl Overview

python-pcl is an implementation of PointCloudLibrary-compatible. interface.

The following is a brief overview of supported subset of PointCloudLibrary interface:

- filters
- features
- keypoints
- registration
- kdtree
- octree
- segmentation
- sample_consensus
- surface
- recognition
- io
- visualization

CHAPTER 2

Installation Guide

2.1 Recommended Environments

We recommend Windows and these Linux distributions.

- Ubuntu 16.04/18.04 64bit
- MacOS 10.9/10.10/10.11/10.12
- Windows 7/8.1/10 64bit

The following versions of Python can be used: 2.7.6+, 3.5.1+, and 3.6.0+.

python-pcl is supported on Python 2.7.6+, 3.4.0, 3.5.0+, 3.6.0+. python-pcl uses C++ compiler such as g++. You need to install it before installing python-pcl. This is typical installation method for each platform:

```
Linux(Ubuntu)

PCL 1.7.2(use apt)

1.Install PCL Module.

$ sudo apt install libpcl-dev -y

Reference <https://packages.ubuntu.com/search?keywords=libpcl-dev>

PCL 1.8.0 (build module)([CI Test Timeout])

1.Build Module

Reference here.

MacOSX

use homebrew

1.Install PCL Module.

$ brew tap homebrew/science

$ brew install pcl

Warning:

Current Installer (2017/10/02) Not generated pcl-2d-1.8.pc file.(Issue #119)

Reference PointCloudLibrary Issue.

Pull qequests 1679.

Issue 1978.

circumvent:

$ copy travis/pcl-2d-1.8.pc file to /usr/local/lib/pkgconfig folder.

Windows

before Install module

Case1. use PCL 1.6.0

Windows SDK 7.1

            PCL All-In-One Installer

            32 bit

            64 bit

            OpenNI2[(PCL Install FolderPath)\3rdParty\OpenNI\OpenNI-(win32/x64)-1.3.2-
↪Dev.msi]
```

(continues on next page)

(continued from previous page)

```
Case2. use 1.8.1

            Visual Studio 2015 C++ Compiler Tools

            PCL All-In-One Installer

            32 bit

            64 bit

OpenNI2[(PCL Install FolderPath)\3rdParty\OpenNI2\OpenNI-Windows-(win32/x64)-2.2.msi]

        Common setting


Windows Gtk+ Download

Download file unzip. Copy bin Folder to pkg-config Folder

or execute powershell file [Install-GTKPlus.ps1].

Python Version use VisualStudio Compiler

set before Environment variable


1.PCL_ROOT


set PCL_ROOT=%PCL Install FolderPath%


2.PATH

(pcl 1.6.0)

$ set PATH=%PCL_ROOT%/bin/;%OPEN_NI_ROOT%/Tools;$(VTK_ROOT)/bin;%PATH%

(pcl 1.8.1)

$ set PATH=%PCL_ROOT%/bin/;%OPEN_NI2_ROOT%/Tools;$(VTK_ROOT)/bin;%PATH%
```

If you use old `setuptools`, upgrade it:

```
$ pip install -U setuptools
```

2.2 Dependencies

Before installing python-pcl, we recommend to upgrade `setuptools` if you are using an old one:

```
$ pip install -U setuptools
```

The following Python packages are required to install python-pcl. The latest version of each package will automatically be installed if missing.

- PointCloudLibrary 1.6.x 1.7.x 1.8.x 1.9.x
- NumPy 1.9, 1.10, 1.11, 1.12, 1.13, ...
- Cython >=0.25.2

2.3 Install python-pcl

2.3.1 Install python-pcl via pip

We recommend to install python-pcl via pip:

```
$ pip install python-pcl
```

Note: All optional PointCloudLibrary related libraries, need to be installed before installing python-pcl. After you update these libraries, please reinstall python-pcl because you need to compile and link to the newer version of them.

2.3.2 Install python-pcl from source

The tarball of the source tree is available via `pip download python-pcl` or from the release notes page. You can use `setup.py` to install python-pcl from the tarball:

```
$ tar zxf python-pcl-x.x.x.tar.gz
$ cd python-pcl-x.x.x
$ python setup.py install
```

You can also install the development version of python-pcl from a cloned Git repository:

```
$ git clone https://github.com/strawlab/python-pcl.git
$ cd pcl/Python
$ python setup.py install
```

2.3.3 When an error occurs...

Use `-vvvv` option with `pip` command. That shows all logs of installation. It may help you:

```
$ pip install python-pcl -vvvv
```

2.3.4 Install python-pcl for developers

python-pcl uses Cython (>=0.25.2). Developers need to use Cython to regenerate C++ sources from `pyx` files. We recommend to use `pip` with `-e` option for editable mode:

```
$ pip install -U cython
$ cd /path/to/python-pcl/source
$ pip install -e .
```

Users need not to install Cython as a distribution package of python-pcl only contains generated sources.

2.4 Uninstall python-pcl

Use pip to uninstall python-pcl:

```
$ pip uninstall python-pcl
```

Note: When you upgrade python-pcl, `pip` sometimes install the new version without removing the old one in `site-packages`. In this case, `pip uninstall` only removes the latest one. To ensure that python-pcl is completely removed, run the above command repeatedly until `pip` returns an error.

2.5 Upgrade python-pcl

Just use `pip` with `-U` option:

```
$ pip install -U python-pcl
```

2.6 Reinstall python-pcl

If you want to reinstall python-pcl, please uninstall python-pcl and then install it. We recommend to use `--no-cache-dir` option as `pip` sometimes uses cache:

```
$ pip uninstall python-pcl
$ pip install python-pcl --no-cache-dir
```

When you install python-pcl without PointCloudLibrary, and after that you want to use PointCloudLibrary, please reinstall python-pcl. You need to reinstall python-pcl when you want to upgrade PointCloudLibrary.

2.7 FAQ

CHAPTER 3

python-pcl Tutorial

3.1 Applications Tutorials

3.1.1 Aligning object templates to a point cloud

This tutorial gives an example of how some of the tools covered in the previous tutorials can be combined to solve a higher level problem - aligning a previously captured model of an object to some newly captured data.

- Original
- TestCode : None

3.1.2 Cluster Recognition and 6DOF Pose Estimation using VFH descriptors

In this tutorial we show how the Viewpoint Feature Histogram (VFH) descriptor can be used to recognize similar clusters in terms of their geometry.

- Original
- TestCode : None

3.1.3 Point Cloud Streaming to Mobile Devices with Real-time Visualization

This tutorial describes how to send point cloud data over the network from a desktop server to a client running on a mobile device.

- Original
- TestCode : None

3.1.4 Detecting people on a ground plane with RGB-D data

This tutorial presents a method for detecting people on a ground plane with RGB-D data.

- Original
- TestCode : None

3.2 Features Tutorials

3.2.1 How 3D Features work in PCL

This document presents a basic introduction to the 3D feature estimation methodologies in PCL.

- Original
- TestCode : None

3.2.2 Estimating Surface Normals in a PointCloud

This tutorial discusses the theoretical and implementation details of the surface normal estimation module in PCL.

- Original
- TestCode : None

3.2.3 Normal Estimation Using Integral Images

In this tutorial we will learn how to compute normals for an organized point cloud using integral images.

- Original
- TestCode : examples/official/Features/NormalEstimationUsingIntegralImages.py

3.2.4 Point Feature Histograms (PFH) descriptors

This tutorial introduces a family of 3D feature descriptors called PFH (Point Feature Histograms) and discusses their implementation details from PCL?fs perspective.

- Original
- TestCode : None

3.2.5 Fast Point Feature Histograms (FPFH) descriptors

This tutorial introduces the FPFH (Fast Point Feature Histograms) 3D descriptor and discusses their implementation details from PCL?fs perspective.

- Original
- TestCode : None

3.2.6 Estimating VFH signatures for a set of points

This document describes the Viewpoint Feature Histogram (VFH) descriptor, a novel representation for point clusters for the problem of Cluster (e.g., Object) Recognition and 6DOF Pose Estimation.

- Original
- TestCode : None

3.2.7 How to extract NARF features from a range image

In this tutorial, we will learn how to extract NARF features from a range image.

- Original
- TestCode : None

3.2.8 Moment of inertia and eccentricity based descriptors

In this tutorial we will learn how to compute moment of inertia and eccentricity of the cloud. In addition to this we will learn how to extract AABB and OBB.

- Original
- TestCode : examples/official/Features/moment_of_inertia.py

3.2.9 RoPs (Rotational Projection Statistics) feature

In this tutorial we will learn how to compute RoPS feature.

- Original
- TestCode : examples/official/Features/rops_feature.py

3.3 Filtering Tutorials

3.3.1 Filtering a PointCloud using a PassThrough filter

In this tutorial, we will learn how to remove points whose values fall inside/outside a user given interval along a specified dimension.

- Original
- TestCode : examples/official/Filtering/PassThroughFilter.py

3.3.2 Downsampling a PointCloud using a VoxelGrid filter

In this tutorial, we will learn how to downsample (i.e., reduce the number of points) a Point Cloud.

- Original
- TestCode : examples/official/Filtering/VoxelGrid_160.py

3.3.3 Removing outliers using a StatisticalOutlierRemoval filter

In this tutorial, we will learn how to remove sparse outliers from noisy data, using StatisticalRemoval.

- Original
- TestCode : examples/official/Filtering/statistical_removal.py

3.3.4 Projecting points using a parametric model

In this tutorial, we will learn how to project points to a parametric model (i.e., plane).

- Original
- TestCode : examples/official/Filtering/project_inliers.py

3.3.5 Extracting indices from a PointCloud

In this tutorial, we will learn how to extract a set of indices given by a segmentation algorithm.

- Original
- TestCode : examples/official/Filtering/extract_indices.py

3.3.6 Removing outliers using a Conditional or RadiusOutlier removal

In this tutorial, we will learn how to remove outliers from noisy data, using ConditionalRemoval, RadiusOutlierRemoval.

- Original
- TestCode : None

3.4 GPU Tutorials

3.4.1 Configuring your PC to use your Nvidia GPU with PCL

This tutorial explains how to configure PCL to use with a Nvidia GPU

- Original
- TestCode : None

3.4.2 Using Kinfu Large Scale to generate a textured mesh

This tutorial demonstrates how to use KinFu Large Scale to produce a mesh from a room, and apply texture information in post-processing for a more appealing visual result.

- Original
- TestCode : None

3.4.3 Detecting people and their poses using PointCloud Library

This tutorial presents a method for people and pose detection.

- Original
- TestCode : None

3.5 Input and Output Tutorials

3.5.1 The PCD (Point Cloud Data) file format

This document describes the PCD file format, and the way it is used inside PCL.

- Original
- TestCode : None

3.5.2 Reading Point Cloud data from PCD files

In this tutorial, we will learn how to read a Point Cloud from a PCD file.

- Original
- TestCode : None

3.5.3 Writing Point Cloud data to PCD files

In this tutorial, we will learn how to write a Point Cloud to a PCD file.

- Original
- TestCode : None

3.5.4 Concatenate the points of two Point Clouds

In this tutorial, we will learn how to concatenate both the fields and the point data of two Point Clouds. When concatenating fields, one PointClouds contains only XYZ data, and the other contains Surface Normal information.

- Original
- TestCode : None

3.5.5 The OpenNI Grabber Framework in PCL

In this tutorial, we will learn how to acquire point cloud data from an OpenNI camera.

- Original
- TestCode : None

3.5.6 The Velodyne High Definition LiDAR (HDL) Grabber

In this tutorial, we will learn how to acquire point cloud data from a Velodyne HDL.

- Original
- TestCode : None

3.5.7 The PCL Dinast Grabber Framework

In this tutorial, we will learn how to acquire point cloud data from a Dinast camera.

- Original
- TestCode : None

3.5.8 Grabbing point clouds from Ensenso cameras

In this tutorial, we will learn how to acquire point cloud data from an IDS-Imaging Ensenso camera.

- Original
- TestCode : None

3.5.9 Grabbing point clouds / meshes from davidSDK scanners

In this tutorial, we will learn how to acquire point cloud or mesh data from a davidSDK scanner.

- Original
- TestCode : None

3.5.10 Grabbing point clouds from DepthSense cameras

In this tutorial we will learn how to setup and use DepthSense cameras within PCL on both Linux and Windows platforms.

- Original
- TestCode : None

3.6 KeyPoint Tutorials

3.6.1 How to extract NARF keypoint from a range image

In this tutorial, we will learn how to extract NARF keypoints from a range image.

- Original
- TestCode : examples/official/keypoints/narf_keypoint_extraction.py

3.7 KdTree Tutorials

3.7.1 How to use a KdTree to search

In this tutorial, we will learn how to search using the nearest neighbor method for k-d trees

- Original
- TestCode : examples/official/kdtree/kdtree_search.py

3.8 Octree Tutorials

3.8.1 Point Cloud Compression

In this tutorial, we will learn how to compress a single point cloud and streams of point clouds.

- Original
- TestCode : None

3.8.2 Spatial Partitioning and Search Operations with Octrees

In this tutorial, we will learn how to use octrees for spatial partitioning and nearest neighbor search.

- Original
- TestCode : examples/official/octree/octree_search.py

3.8.3 Spatial change detection on unorganized point cloud data

In this tutorial, we will learn how to use octrees for detecting spatial changes within point clouds.

- Original
- TestCode : examples/official/octree/octree_change_detection.py

3.9 RangeImage Tutorials

3.9.1 How to create a range image from a point cloud

This tutorial demonstrates how to create a range image from a point cloud and a given sensor position.

- Original
- TestCode : None

3.9.2 How to extract borders from range images

This tutorial demonstrates how to extract borders (traversals from foreground to background) from a range image.

- Original
- TestCode : examples/official/RangeImage/range_image_border_extraction.py

3.10 Recognition Tutorials

3.10.1 3D Object Recognition based on Correspondence Grouping

This tutorial aims at explaining how to perform 3D Object Recognition based on the pcl_recognition module.

- Original
- TestCode : None

3.10.2 Implicit Shape Model

In this tutorial we will learn how the Implicit Shape Model algorithm works and how to use it for finding objects centers.

- Original
- TestCode : None

3.10.3 Tutorial: Hypothesis Verification for 3D Object Recognition

This tutorial aims at explaining how to do 3D object recognition in clutter by verifying model hypotheses in cluttered and heavily occluded 3D scenes.

- Original
- TestCode : None

3.11 Registration Tutorials

3.11.1 The PCL Registration API

In this document, we describe the point cloud registration API and its modules: the estimation and rejection of point correspondences, and the estimation of rigid transformations.

- Original
- TestCode : None

3.11.2 How to use iterative closest point

This tutorial gives an example of how to use the iterative closest point algorithm to see if one PointCloud is just a rigid transformation of another PointCloud.

- Original
- TestCode : examples/official/Registration/iterative_closest_point.py

3.11.3 How to incrementally register pairs of clouds

This document demonstrates using the Iterative Closest Point algorithm in order to incrementally register a series of point clouds two by two.

- Original
- TestCode : None

3.11.4 Interactive Iterative Closest Point

This tutorial will teach you how to build an interactive ICP program

- Original
- TestCode : None

3.11.5 How to use Normal Distributions Transform

This document demonstrates using the Normal Distributions Transform algorithm to register two large point clouds.

- Original
- TestCode : examples/official/Registration/normal_distributions_transform.py

3.11.6 In-hand scanner for small objects

This document shows how to use the In-hand scanner applications to obtain colored models of small objects with RGB-D cameras.

- Original
- TestCode : None

3.11.7 Robust pose estimation of rigid objects

In this tutorial, we show how to find the alignment pose of a rigid object in a scene with clutter and occlusions.

- Original
- TestCode : examples/official/Registration/alignment_prerejective.py

3.12 Sampleconsensus Tutorials

3.12.1 How to use Random Sample Consensus model

In this tutorial we learn how to use a RandomSampleConsensus with a plane model to obtain the cloud fitting to this model.

- Original
- TestCode : examples/official/SampleConsensus/random_sample_consensus.py

3.13 segmentation Tutorials

3.13.1 Plane model segmentation

In this tutorial, we will learn how to segment arbitrary plane models from a given point cloud dataset.

- Original
- TestCode : examples/official/Segmentation/Plane_model_segmentation.py

3.13.2 Cylinder model segmentation

In this tutorial, we will learn how to segment arbitrary cylindrical models from a given point cloud dataset.

- Original
- TestCode : examples/official/Segmentation/cylinder_segmentation.py

3.13.3 Euclidean Cluster Extraction

In this tutorial we will learn how to extract Euclidean clusters with the pcl::EuclideanClusterExtraction class.

- Original
- TestCode : examples/official/Segmentation/cluster_extraction.py

3.13.4 Region growing segmentation

In this tutorial we will learn how to use region growing segmentation algorithm.

- Original
- TestCode : None

3.13.5 Color-based region growing segmentation

In this tutorial we will learn how to use color-based region growing segmentation algorithm.

- Original
- TestCode : None

3.13.6 Min-Cut Based Segmentation

In this tutorial we will learn how to use min-cut based segmentation algorithm.

- Original
- TestCode : None

3.13.7 Conditional Euclidean Clustering

This tutorial describes how to use the Conditional Euclidean Clustering class in PCL: A segmentation algorithm that clusters points based on Euclidean distance and a user-customizable condition that needs to hold.

- Original
- TestCode : None

3.13.8 Difference of Normals Based Segmentation

In this tutorial we will learn how to use the difference of normals feature for segmentation.

- Original
- TestCode : None

3.13.9 Clustering of Pointclouds into Supervoxels - Theoretical primer

In this tutorial, we show to break a pointcloud into the mid-level supervoxel representation.

- Original
- TestCode : None

3.13.10 Identifying ground returns using ProgressiveMorphologicalFilter segmentation

In this tutorial, we show how to segment a point cloud into ground and non-ground returns.

- Original
- TestCode : None

3.13.11 Filtering a PointCloud using ModelOutlierRemoval

This tutorial describes how to extract points from a point cloud using SAC models

- Original
- TestCode : None

3.14 surface Tutorials

3.14.1 Smoothing and normal estimation based on polynomial reconstruction

In this tutorial, we will learn how to construct and run a Moving Least Squares (MLS) algorithm to obtain smoothed XYZ coordinates and normals.

- Original
- TestCode : None

3.14.2 Construct a concave or convex hull polygon for a plane model

In this tutorial we will learn how to calculate a simple 2D concave or convex hull polygon for a set of points supported by a plane.

- Original
- TestCode : examples/official/Surface/concave_hull_2d.py

3.14.3 Fast triangulation of unordered point clouds

In this tutorial we will learn how to run a greedy triangulation algorithm on a PointCloud with normals to obtain a triangle mesh based on projections of the local neighborhood.

- Original
- TestCode : None

3.14.4 Fitting trimmed B-splines to unordered point clouds

In this tutorial we will learn how to reconstruct a smooth surface from an unordered point-cloud by fitting trimmed B-splines.

- Original
- TestCode : None

3.15 Tracking Tutorials

3.15.1 Tracking Example

In this tutorial, we will learn how to construct and run a Moving Least Squares (MLS) algorithm to obtain smoothed XYZ coordinates and normals.

- Original Page : None (tutorials/sources/tracking/tracking_sample.cpp)
- TestCode : None

3.16 Visualization Tutorials

3.16.1 The CloudViewer

This tutorial demonstrates how to use the pcl visualization tools.

- Original
- TestCode : None

3.16.2 How to visualize a range image

This tutorial demonstrates how to use the pcl visualization tools for range images.

- Original
- TestCode : None

3.16.3 PCLVisualizer

This tutorial demonstrates how to use the PCLVisualizer class for powerful visualisation of point clouds and related data.

- Original
- TestCode : None

3.16.4 PCLPlotter

This tutorial demonstrates how to use the PCLPlotter class for powerful visualisation of plots, charts and histograms of raw data and explicit functions.

- Original
- TestCode : None

3.16.5 Visualization

This tutorial will give an overview on the usage of the PCL visualization tools.

- Original
- TestCode : None

3.16.6 Create a PCL visualizer in Qt with cmake

This tutorial shows you how to create a PCL visualizer within a Qt application.

- Original
- TestCode : None

3.16.7 Create a PCL visualizer in Qt to colorize clouds

This tutorial shows you how to color point clouds within a Qt application.

- Original
- TestCode : None

CHAPTER 4

python-pcl Reference Manual

This is the official reference of python-pcl, PointCloudLibrary-like API interface.

4.1 Indices and tables

- genindex
- modindex

4.2 Reference

4.2.1 pcl package

Submodules

pcl.pcl_visualization module

Module contents

CHAPTER 5

For python-pcl Developers

5.1 python-pcl Contribution Guide

This is a guide for all contributions to python-pcl. The development of python-pcl is running on the official repository at GitHub. Anyone that wants to register an issue or to send a pull request should read through this document.

5.1.1 Classification of Contributions

There are several ways to contribute to python-pcl community:

1. Registering an issue
2. Sending a pull request (PR)

This document mainly focuses on 1 and 2, though other contributions are also appreciated.

5.1.2 Release and Milestone

We are using GitHub Flow as our basic working process. In particular, we are using the master branch for our development, and releases are made as tags.

Releases are classified into three groups: major, minor, and revision. This classification is based on following criteria:

- **Major update** contains disruptive changes that break the backward compatibility.
- **Minor update** contains additions and extensions to the APIs keeping the supported backward compatibility.
- **Revision update** contains improvements on the API implementations without changing any API specification.

The release classification is reflected into the version number x.y.z, where x, y, and z corresponds to major, minor, and revision updates, respectively.

We set a milestone for an upcoming release. The milestone is of name 'vX.Y.Z', where the version number represents a revision release at the outset. If at least one *feature* PR is merged in the period, we rename the milestone to represent a minor release (see the next section for the PR types).

See also *API Compatibility Policy*.

5.1.3 Issues and PRs

Issues and PRs are classified into following categories:

- **Bug**: bug reports (issues) and bug fixes (PRs)
- **Enhancement**: implementation improvements without breaking the interface
- **Feature**: feature requests (issues) and their implementations (PRs)
- **NoCompat**: disrupts backward compatibility
- **Test**: test fixes and updates
- **Document**: document fixes and improvements
- **Example**: fixes and improvements on the examples
- **Install**: fixes installation script
- **Contribution-Welcome**: issues that we request for contribution (only issues are categorized to this)
- **Other**: other issues and PRs

Issues and PRs are labeled by these categories. This classification is often reflected into its corresponding release category: Feature issues/PRs are contained into minor/major releases and NoCompat issues/PRs are contained into major releases, while other issues/PRs can be contained into any releases including revision ones.

On registering an issue, write precise explanations on what you want python-pcl to be. Bug reports must include necessary and sufficient conditions to reproduce the bugs. Feature requests must include **what** you want to do (and **why** you want to do, if needed). You can contain your thoughts on **how** to realize it into the feature requests, though **what** part is most important for discussions.

If you can write code to fix an issue, send a PR to the master branch. Before writing your code for PRs, read through the *Coding Guidelines*. The description of any PR must contain a precise explanation of **what** and **how** you want to do; it is the first documentation of your code for developers, a very important part of your PR.

Once you send a PR, it is automatically tested on Travis CI for Linux and Mac OS X, and on AppVeyor for Windows. Your PR need to pass at least the test for Linux/MacOSX on Travis CI and Windows on AppVeyor. After the automatic test passes, some of the core developers will start reviewing your code. Note that this automatic PR test only includes CPU tests.

Even if your code is not complete, you can send a pull request as a *work-in-progress PR* by putting the `[WIP]` prefix to the PR title. If you write a precise explanation about the PR, core developers and other contributors can join the discussion about how to proceed the PR.

5.1.4 Coding Guidelines

We use PEP8 and a part of OpenStack Style Guidelines related to general coding style as our basic style guidelines.

To check your code, use `autopep8` and `flake8` command installed by `hacking` package:

```
$ pip install autopep8 hacking
$ autopep8 --global-config .pep8 path/to/your/code.py
$ flake8 path/to/your/code.py
```

To check Cython code, use `.flake8.cython` configuration file:

```
$ flake8 --config=.flake8.cython path/to/your/cython/code.pyx
```

The `autopep8` supports automatically correct Python code to conform to the PEP 8 style guide:

```
$ autopep8 --in-place --global-config .pep8 path/to/your/code.py
```

The `flake8` command lets you know the part of your code not obeying our style guidelines. Before sending a pull request, be sure to check that your code passes the `flake8` checking.

Note that `flake8` command is not perfect. It does not check some of the style guidelines. Here is a (not-complete) list of the rules that `flake8` cannot check.

- Relative imports are prohibited. [H304]
- Importing non-module symbols is prohibited.
- Import statements must be organized into three parts: standard libraries, third-party libraries, and internal imports. [H306]

In addition, we restrict the usage of *shortcut symbols* in our code base. They are symbols imported by packages and sub-packages of `python-pcl`. **It is not allowed to use such shortcuts in the ‘‘python-pcl‘‘ library implementation**. Note that you can still use them in `tests` and `examples` directories.

Once you send a pull request, your coding style is automatically checked by Travis CI for Linux and Mac OS X, and on AppVeyor for Windows. The reviewing process starts after the check passes.

The python-pcl is designed based on PointCloudLibrary's API design. python-pcl's source code and documents contain the original PointCloudLibrary ones. Please note the followings when writing the document.

- In order to identify overlapping parts, it is preferable to add some remarks that this document is just copied or altered from the original one. It is also preferable to briefly explain the specification of the function in a short paragraph, and refer to the corresponding function in PointCloudLibrary so that users can read the detailed document. However, it is possible to include a complete copy of the document with such a remark if users cannot summarize in such a way.
- If a function in python-pcl only implements a limited amount of features in the original one, users should explicitly describe only what is implemented in the document.

5.1.5 Testing Guidelines

Testing is one of the most important part of your code. You must test your code by unit tests following our testing guidelines. Note that we are using the nose package and the mock package for testing, so install nose and mock before writing your code:

```
$ pip install nose mock
```

In order to run unit tests at the repository root, you first have to build Cython files in place by running the following command:

```
$ python setup.py develop
```

Once the Cython modules are built, you can run unit tests simply by running `nosetests` command at the repository root:

```
$ nosetests
```

Tests are put into the `tests` directories.

Following this naming convention, you can run all the tests by just typing `nosetests` at the repository root:

```
$ nosetests
```

If you modify the code related to existing unit tests, you must run appropriate commands.

There are many examples of unit tests under the `tests` directory. They simply use the `unittest` package of the standard library.

Once you send a pull request, your code is automatically tested by Travis-CI and Appveyor. The reviewing process starts after the test passes.

Note: Some of numerically unstable tests might cause errors irrelevant to your changes. In such a case, we ignore the failures and go on to the review process, so do not worry about it.

5.2 API Compatibility Policy

This document expresses the design policy on compatibilities of python-pcl APIs. Development team should obey this policy on deciding to add, extend, and change APIs and their behaviors.

This document is written for both users and developers. Users can decide the level of dependencies on python-pcl implementations in their codes based on this document. Developers should read through this document before creating pull requests that contain changes on the interface. Note that this document may contain ambiguities on the level of supported compatibilities.

5.2.1 Versioning and Backward Compatibilities

The updates of python-pcl are classified into three levels: major, minor, and revision. These types have distinct levels of backward compatibilities.

- **Major update** contains disruptive changes that break the backward compatibility.
- **Minor update** contains addition and extension to the APIs keeping the supported backward compatibility.
- **Revision update** contains improvements on the API implementations without changing any API specifications.

Note that we do not support full backward compatibility, which is almost infeasible for Python-based APIs, since there is no way to completely hide the implementation details.

5.2.2 Processes to Break Backward Compatibilities

Deprecation, Dropping, and Its Preparation

Any APIs may be *deprecated* at some minor updates. In such a case, the deprecation note is added to the API documentation, and the API implementation is changed to fire deprecation warning (if possible). There should be another way to reimplement the same things previously written with the deprecated APIs.

Any APIs may be marked as *to be dropped in the future*. In such a case, the dropping is stated in the documentation with the major version number on which the API is planned to be dropped, and the API implementation is changed to fire the future warning (if possible).

The actual dropping should be done through the following steps:

- Make the API deprecated. At this point, users should not need the deprecated API in their new application codes.

- After that, mark the API as *to be dropped in the future*. It must be done in the minor update different from that of the deprecation.
- At the major version announced in the above update, drop the API.

Consequently, it takes at least two minor versions to drop any APIs after the first deprecation.

API Changes and Its Preparation

Any APIs may be marked as *to be changed in the future* for changes without backward compatibility. In such a case, the change is stated in the documentation with the version number on which the API is planned to be changed, and the API implementation is changed to fire the future warning on the certain usages.

The actual change should be done in the following steps:

- Announce that the API will be changed in the future. At this point, the actual version of change need not be accurate.
- After the announcement, mark the API as *to be changed in the future* with version number of planned changes. At this point, users should not use the marked API in their new application codes.
- At the major update announced in the above update, change the API.

5.2.3 Supported Backward Compatibility

This section defines backward compatibilities that minor updates must maintain.

Documented Interface

python-pcl has the official API documentation. Many applications can be written based on the documented features. We support backward compatibilities of documented features. In other words, codes only based on the documented features run correctly with minor/revision-updated versions.

Developers are encouraged to use apparent names for objects of implementation details. For example, attributes outside of the documented APIs should have one or more underscores at the prefix of their names.

Undocumented behaviors

Behaviors of python-pcl implementation not stated in the documentation are undefined. Undocumented behaviors are not guaranteed to be stable between different minor/revision versions.

Minor update may contain changes to undocumented behaviors. For example, suppose an API X is added at the minor update. In the previous version, attempts to use X cause AttributeError. This behavior is not stated in the documentation, so this is undefined. Thus, adding the API X in minor version is permissible.

Revision update may also contain changes to undefined behaviors. Typical example is a bug fix. Another example is an improvement on implementation, which may change the internal object structures not shown in the documentation. As a consequence, **even revision updates do not support compatibility of pickling, unless the full layout of pickled objects is clearly documented.**

Documentation Error

Compatibility is basically determined based on the documentation, though it sometimes contains errors. It may make the APIs confusing to assume the documentation always stronger than the implementations. We therefore may fix the documentation errors in any updates that may break the compatibility in regard to the documentation.

Note: Developers MUST NOT fix the documentation and implementation of the same functionality at the same time in revision updates as "bug fix". Such a change completely breaks the backward compatibility. If you want to fix the bugs in both sides, first fix the documentation to fit it into the implementation, and start the API changing procedure described above.

Object Attributes and Properties

Object attributes and properties are sometimes replaced by each other at minor updates. It does not break the user codes, except the codes depend on how the attributes and properties are implemented.

Functions and Methods

Methods may be replaced by callable attributes keeping the compatibility of parameters and return values in minor updates. It does not break the user codes, except the codes depend on how the methods and callable attributes are implemented.

Exceptions and Warnings

The specifications of raising exceptions are considered as a part of standard backward compatibilities. No exception is raised in the future versions with correct usages that the documentation allows, unless the API changing process is completed.

On the other hand, warnings may be added at any minor updates for any APIs. It means minor updates do not keep backward compatibility of warnings.

5.2.4 Installation Compatibility

The installation process is another concern of compatibilities. We support environmental compatibilities in the following ways.

- Supporting optional packages/libraries may be done in minor updates (e.g. supporting h5py in optional features).

Note: The installation compatibility does not guarantee that all the features of python-pcl correctly run on supported environments. It may contain bugs that only occurs in certain environments. Such bugs should be fixed in some updates.

CHAPTER 6

License

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

6.1 python-pcl

The python-pcl is designed based on Point Cloud Library's API. python-pcl's source code and documents contain the original Point Cloud Library ones.

Software License Agreement (BSD License)

Point Cloud Library (PCL) - www.pointclouds.org Copyright (c) 2009-2012, Willow Garage, Inc. Copyright (c) 2012-, Open Perception, Inc. Copyright (c) XXX, respective authors.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

- Neither the name of the copyright holder(s) nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

Python Module Index

p

Index

P