Adaptive Analysis of Geometric Tolerances with Low-Cost Sensors

Generated by Doxygen 1.8.18

Chapter 1

Documentation

This is the code documentation for the Bachelor's Thesis "Adaptive Analysis of Geometric Tolerances with Low-Cost Sensors".

1.1 How to setup the programming environment

In the following it is described briefly how to setup all necessary libraries used in this project.

The references are set mostly using variables in CMakeLists.txt which have to be updated accordingly. Alternatively, environment variables could be set.

1.1.1 Visual Studio and Cmake

Visual Studio 2019 was used and can be downloaded here: https://visualstudio.microsoft.com/vs/. Visual Studio Cmake Tools have to be installed.

Furthermore, Cmake itself has to be installed. The minimum version is 3.6: https://cmake.org/download/.

1.1.2 Turntable HW-Control

The files have to be downloaded from the github directory and the source code has to be built.

Afterwards, in CMakeLists TURNTABLE_DIR has to be adjusted and DLL_TURNTABLE has to be checked.

1.1.3 Realsense SDK

The realsense sdk can be downloaded here: https://github.com/IntelRealSense/librealsense/releases.

Version 2.33.1 was used for this project. Adjust REALSENSE_DIR in CmakeLists accordingly.

1.1.4 Point Cloud Library

PCL can be downloaded here: https://github.com/PointCloudLibrary/pcl/releases. The Version used for this project was 1.9.1.

Using the windows installer, an environment variable should be set automatically. If not, add it manually: PCL_R↩ OOT (to the root directory of PCL).

Furthermore add PCL_ROOT%\bin to your PATH variable.

1.1.5 Open CV

Download OpenCV and compile it: https://github.com/opencv/opencv.

OpenCV contrib has to be downloaded and built as well (some modules are used from that): https↩ ://github.com/opencv/opencv_contrib.

Adjust OPENCV_DIR accordingly.

1.1.6 JT Open Library

JT Open Libary can be downloaded here: https://www.plm.automation.siemens.com/store/en-us/trial/jt
html.

After registration, a 60 days trial version begins.

In CMakeLists the variables JTTK_LIB and JTTK_INCLUDE have to be adjusted accordingly.

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

Class Documentation

3.1 Comparison Class Reference

Class for the subproblem of the comparison.

`#include <Comparison.h>`

Public Member Functions

- Comparison (std::string debug)
- bool hausdorff (PTC::Ptr cam, PTC::Ptr comp, float &hausdorff, float &avg_dist)
- bool hausdorff (PTC::Ptr cam, PTC::Ptr comp, std::vector< double > thresholds, pcl::PointCloud< pcl::↩ PointXYZRGB >::Ptr comp_hausdorff, std::vector< int > &defects, float &hausdorff, float &avg_dist, bool outlierDeletion=false, bool noCloudNeeded=false)
- void findEdges (PTC::Ptr cloud, double edge_radius, int maxNeighbors, double leafSize)
- bool findCircle (PTC::Ptr plane, pcl::ModelCoefficients::Ptr planeCoeffs, PTC::Ptr circle, pcl::Model↩ Coefficients::Ptr coefficients, double distThresh, int maxIterations, double minInlierRatio, double radius=0, double radiusLowerDelta=0, double radiusUpperDelta=0, Eigen::Vector3f *center=new Eigen::Vector3f(), double epsCenter=0, double edge_radius=0, int maxNeighbors=0, double leafSize=0)
- bool findLine (PTC::Ptr plane, PTC::Ptr line, pcl::ModelCoefficients::Ptr coefficients, double distThresh, int maxIterations, double minInlierRatio, Eigen::Vector3f *axis=new Eigen::Vector3f(), double epsAxis=0, PT↩ C::Ptr ref=PTC::Ptr(new PTC), double epsRef=0, double edge_radius=0, int maxNeighbors=0, double leaf↩ Size=0)
- bool findPlane (PTC::Ptr cloud, PTC::Ptr plane, pcl::ModelCoefficients::Ptr coefficients, pcl::PointCloud< pcl::Normal >::Ptr normals, double normal_weight, double distThresh, int maxIterations, double min↩ InlierRatio, Eigen::Vector3f *normalAxis=new Eigen::Vector3f(), double epsAngle=0, double distanceFrom↩ Origin=0, double epsDist=0, PTC::Ptr projectedPlane=PTC::Ptr(new PTC))
- bool findCyl (PTC::Ptr cloud, PTC::Ptr cyl, pcl::ModelCoefficients::Ptr coefficients, pcl::PointCloud< pcl::↩ Normal >::Ptr normals, double normal_weight, double distThresh, int maxIterations, double minInlierRatio, Eigen::Vector3f *axis=new Eigen::Vector3f(), double epsAxis=0, double radius=0, double epsRadius=0, Eigen::Vector3f *center=new Eigen::Vector3f(), double epsCenter=0, PTC::Ptr projectedCyl=PTC::Ptr(new PTC))
- bool findFeature (pcl::SACSegmentationFromNormals< PT, pcl::Normal > seg, PTC::Ptr remainingCloud, PTC::Ptr featureCloud, pcl::ModelCoefficients::Ptr coefficients, pcl::PointCloud< pcl::Normal >::Ptr normals, double normalWeight, double distThresh, int maxIterations, int minInliers, bool projectInliers, PTC::Ptr projectedCloud=PTC::Ptr(new PTC), pcl::PointCloud< pcl::Normal >::Ptr featureNormals=pcl::PointCloud< pcl::Normal >::Ptr(new pcl::PointCloud< pcl::Normal >))
- void checkTolerances (PTC::Ptr cam, std::vector< Dimension > &tolerances, double edge_radius, int max↩ Neighbors, double leafSize, double minInlierRatio, double maxTolerance, int numberOfIterations_Tol, dou- ble tolerance_search_radius, double plane_normal_weight, double cyl_normal_weight, double plane_dist↩ _thresh, double line_dist_thresh, double circle_dist_thresh, double plane_epsDist, double circle_epsCenter, double cyl_epsCenter, double angleThres, double angular_angleThres)
- void visualizeTolerances (PTC::Ptr cam, std::vector< Dimension > tolerances, std::string name="Tolerances")

3.1.1 Detailed Description

Class for the subproblem of the comparison.

This class allows to perform all kinds of feature detections: edges, circles, lines, planes and cylinders. Furthermore, it also offers a more generic feature detection function. Moreover, the function for the whole tolerance check is implemented as well.

Author

Justin Heinz, Oliver Krumpek et Al.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 Comparison()

```
Comparison::Comparison (
            std::string debug )
```

Standard constructor

Parameters

in	*debug*	- how many debug info should be displayed (0 - none,1 - some,2 - all)

3.1.3 Member Function Documentation

3.1.3.1 checkTolerances()

```
void Comparison::checkTolerances (
            PTC::Ptr cam,
            std::vector< Dimension > & tolerances,
            double edge_radius,
            int maxNeighbors,
            double leafSize,
            double minInlierRatio,
            double maxTolerance,
            int numberOfIterations_Tol,
            double tolerance_search_radius,
            double plane_normal_weight,
            double cyl_normal_weight,
            double plane_dist_thresh,
            double line_dist_thresh,
            double circle_dist_thresh,
```

```
            double plane_epsDist,
            double circle_epsCenter,
            double cyl_epsCenter,
            double angleThres,
            double angular_angleThres )
```

Algorithm for checking all tolerances of a given tolerance definition. Measured is set to -1, when a feature could not be found

Parameters

in	*cam*	- the cloud in which the features should be found and measured
in,out	*tolerances*	- the vector of tolerances used. It is returned with the measured value.
in	*edge_radius*	- search radius for edge detection
in	*maxNeighbors*	- The maximum number of neighbors a point is allowed to have within the search radius to be considered as an edge
in	*leafSize*	- leafSize for the downsampling
in	*minInlierRatio*	- the minimum number of inliers in relation to the maximum number of possible inliers a feature must have to be considered as valid
in	*maxTolerance*	- the maximal detectable tolerance during the tolerance check
in	*numberOfIterations_Tol*	- the number of iterations for the feature detection
in	*plane_normal_weight*	- the weight of the deviation in the surface normals in comparison to the deviation in the position for the plane detection
in	*cyl_normal_weight*	- the weight of the deviation in the surface normals in comparison to the deviation in the position for the cylinder detection#
in	*plane_dist_thresh*	- distance Threshold for the plane detection
in	*line_dist_thresh*	- distance Threshold for the line detection
in	*circle_dist_thresh*	- distance Threshold for the circle detection
in	*plane_epsDist*	- the allowed deviation in distance for the plane detection
in	*circle_epsCenter*	- allowed deviation with respect to the center for the circle detection
in	*angularThres*	- the allowed angular deviation to the respetively given axes in rad
in	*angularThres*	- the allowed angular deviation to the respetively given axes in rad for angular tolerances

3.1.3.2 findCircle()

```
bool Comparison::findCircle (
            PTC::Ptr plane,
            pcl::ModelCoefficients::Ptr planeCoeffs,
            PTC::Ptr circle,
            pcl::ModelCoefficients::Ptr coefficients,
            double distThresh,
            int maxIterations,
            double minInlierRatio,
            double radius = 0,
            double radiusLowerDelta = 0,
            double radiusUpperDelta = 0,
            Eigen::Vector3f * center = new Eigen::Vector3f(),
            double epsCenter = 0,
            double edge_radius = 0,
```

```
            int maxNeighbors = 0,
            double leafSize = 0 )
```

Detection of circles in a plane.

Parameters

in,out	*plane*	- plane used for the circle detection. The edges in the plane are returned.
in	*planeCoeffs*	- the coefficients of the plane
out	*circle*	- a pointcloud representing the circle found
out	*coefficients*	- the coefficients of the circle found
in	*distThresh*	- distance Threshold for the feature detection
in	*maxIterations*	- the number of iterations for the feature detection
in	*minInlierRatio*	- the minimum number of inliers in relation to the maximum number of possible inliers a feature must have to be considered as valid
in	*radius*	- the expected radius of the circle
in	*radiusLowerDelta*	- the maximal allowed negative deviation
in	*radiusUperDelta*	- the maximal allowed positive deviation
in	*center*	- expected center point of the circle
in	*epsCenter*	- allowed deviation with respect to the center
in	*edge_radius*	- search radius for edge detection
in	*maxNeighbors*	- The maximum number of neighbors a point is allowed to have within the search radius to be considered as an edge
in	*leafSize*	- leafSize for the downsampling

Returns

true if a circle within the thresholds was found, false otherwise

3.1.3.3 findCyl()

```
bool Comparison::findCyl (
            PTC::Ptr cloud,
            PTC::Ptr cyl,
            pcl::ModelCoefficients::Ptr coefficients,
            pcl::PointCloud< pcl::Normal >::Ptr normals,
            double normal_weight,
            double distThresh,
            int maxIterations,
            double minInlierRatio,
            Eigen::Vector3f * axis = new Eigen::Vector3f(),
            double epsAxis = 0,
            double radius = 0,
            double epsRadius = 0,
            Eigen::Vector3f * center = new Eigen::Vector3f(),
            double epsCenter = 0,
            PTC::Ptr projectedCyl = PTC::Ptr(new PTC) )
```

Detection of cylinders.

Parameters

in,out	*cloud*	- cloud used for the line detection. The cloud without the cylinder is returned
out	*cyl*	- a pointcloud representing the cylinder found
out	*coefficients*	- the coefficients of the cylinder found
in	*normals*	- the surface normals of the cloud
in	*normal_weight*	- the weight of the deviation in the surface normals in comparison to the deviation in the position
in	*distThresh*	- distance Threshold for the feature detection
in	*maxIterations*	- the number of iterations for the feature detection
in	*minInlierRatio*	- the minimum number of inliers in relation to the maximum number of possible inliers a feature must have to be considered as valid
in	*axis*	- the expected center axis of the cylinder
in	*epsAngle*	- the allowed angular deviation to the given axis in rad
in	*radius*	- the expected radius of the cylinder
in	*epsRadius*	- the allowed deviation in radius
in	*center*	- expected center point of the cylinder
in	*epsCenter*	- allowed deviation with respect to the center
out	*projectedCyl*	- the cylinder with all inliers projected to the found model

Returns

true if a cylinder within the thresholds was found, false otherwise

3.1.3.4 findEdges()

```
void Comparison::findEdges (
            PTC::Ptr cloud,
            double edge_radius,
            int maxNeighbors,
            double leafSize )
```

Detection of edges using a radius outlier removal filter. Beforehand the cloud is downsampled to guarantee a predictable point density.

Parameters

in,out	*cloud*	- cloud to detect edges in
in	*edge_radius*	- search radius used for the outlier removal filter
in	*maxNeighbors*	- The maximum number of neighbors a point is allowed to have within the search radius to be considered as an edge
out	*leafSize*	- leafSize for the downsampling

3.1.3.5 findFeature()

```
bool Comparison::findFeature (
```

```
            pcl::SACSegmentationFromNormals< PT, pcl::Normal > seg,
            PTC::Ptr remainingCloud,
            PTC::Ptr featureCloud,
            pcl::ModelCoefficients::Ptr coefficients,
            pcl::PointCloud< pcl::Normal >::Ptr normals,
            double normalWeight,
            double distThresh,
            int maxIterations,
            int minInliers,
            bool projectInliers,
            PTC::Ptr projectedCloud = PTC::Ptr(new PTC),
            pcl::PointCloud< pcl::Normal >::Ptr featureNormals = pcl::PointCloud< pcl:↩
:Normal >::Ptr(new pcl::PointCloud< pcl::Normal >) )
```

Detection of generic features.

Attention

: The projectInliers Filter contains bugs for cylinder in PCL 1.9.1. Thus the code was reimplemented manually.

Parameters

in	*seg*	- the segmentation object, which already has been initialized with the model type and all necessary constraints for the feature detection.
in,out	*remainingCloud*	- cloud used for the feature detection. The cloud without the feature is returned
out	*featureCloud*	- a pointcloud representing the feature found
out	*coefficients*	- the coefficients of the feature found
in	*normals*	- the surface normals of the cloud
in	*normal_weight*	- the weight of the deviation in the surface normals in comparison to the deviation in the position
in	*distThresh*	- distance Threshold for the feature detection
in	*maxIterations*	- the number of iterations for the feature detection
in	*minInliers*	- the minimum number of inliers a feature must have to be considered as valid
in	*projectInliers*	- If true, all inliers are projected to the model
out	*projectedCloud*	- the feature with all inliers projected to the found model
out	*featureNormals*	- the normals of the feature found

Returns

true if a cylinder within the thresholds was found, false otherwise

3.1.3.6 findLine()

```
bool Comparison::findLine (
            PTC::Ptr plane,
            PTC::Ptr line,
            pcl::ModelCoefficients::Ptr coefficients,
            double distThresh,
            int maxIterations,
            double minInlierRatio,
```

```
            Eigen::Vector3f * axis = new Eigen::Vector3f(),
            double epsAxis = 0,
            PTC::Ptr ref = PTC::Ptr(new PTC),
            double epsRef = 0,
            double edge_radius = 0,
            int maxNeighbors = 0,
            double leafSize = 0 )
```

Detection of lines.

Parameters

in,out	*plane*	- plane used for the line detection. The edges in the plane are returned.
out	*line*	- a pointcloud representing the line found
out	*coefficients*	- the coefficients of the line found
in	*distThresh*	- distance Threshold for the feature detection
in	*maxIterations*	- the number of iterations for the feature detection
in	*minInlierRatio*	- the minimum number of inliers in relation to the maximum number of possible inliers a feature must have to be considered as valid
in	*axis*	- the axis on which the line search is performed
in	*epsAxis*	- the allowed angular deviation to the given axis in rad
in	*ref*	- the cloud of the reference line
in	*epsRef*	- the allowed average distance to the reference line
in	*edge_radius*	- search radius for edge detection
in	*maxNeighbors*	- The maximum number of neighbors a point is allowed to have within the search radius to be considered as an edge
in	*leafSize*	- leafSize for the downsampling

Returns

true if a line within the thresholds was found, false otherwise

3.1.3.7 findPlane()

```
bool Comparison::findPlane (
            PTC::Ptr cloud,
            PTC::Ptr plane,
            pcl::ModelCoefficients::Ptr coefficients,
            pcl::PointCloud< pcl::Normal >::Ptr normals,
            double normal_weight,
            double distThresh,
            int maxIterations,
            double minInlierRatio,
            Eigen::Vector3f * normalAxis = new Eigen::Vector3f(),
            double epsAngle = 0,
            double distanceFromOrigin = 0,
            double epsDist = 0,
            PTC::Ptr projectedPlane = PTC::Ptr(new PTC) )
```

Detection of planes.

Parameters

in,out	*cloud*	- cloud used for the line detection. The cloud without the plane is returned
out	*plane*	- a pointcloud representing the plane found
out	*coefficients*	- the coefficients of the plane found
in	*normals*	- the surface normals of the cloud
in	*normal_weight*	- the weight of the deviation in the surface normals in comparison to the deviation in the position
in	*distThresh*	- distance Threshold for the feature detection
in	*maxIterations*	- the number of iterations for the feature detection
in	*minInlierRatio*	- the minimum number of inliers in relation to the maximum number of possible inliers a feature must have to be considered as valid
in	*normalAxis*	- the expected normal axis of the plane
in	*epsAngle*	- the allowed angular deviation to the given axis in rad
in	*distanceFromOrigin*	- the expected distance between the plane and the origin
in	*epsDist*	- the allowed deviation in distance
out	*projectedPlane*	- the plane with all inliers projected to the found model

Returns

true if a plane within the thresholds was found, false otherwise

3.1.3.8 hausdorff() [1/2]

```
bool Comparison::hausdorff (
            PTC::Ptr cam,
            PTC::Ptr comp,
            float & hausdorff,
            float & avg_dist )
```

Hausdorff comparison without returning a colored hausdorff cloud. Keep in mind that outlierDeletion is set to false automatically.

Parameters

in	*cam*	- measured cloud
in	*comp*	- cloud of the component
out	*hausdorff*	- hausdorff distance
out	*avg_dist*	- average distance of all points.

Returns

true if comp contains defects, false otherwise true if comp contains defects, false otherwise

3.1.3.9 hausdorff() [2/2]

```
bool Comparison::hausdorff (
            PTC::Ptr cam,
            PTC::Ptr comp,
            std::vector< double > thresholds,
            pcl::PointCloud< pcl::PointXYZRGB >::Ptr comp_hausdorff,
            std::vector< int > & defects,
            float & hausdorff,
            float & avg_dist,
            bool outlierDeletion = false,
            bool noCloudNeeded = false )
```

Hausdorff comparison and returning a colored hausdorff cloud. Source: Krumpek, O. et al.: RobotScan. Projektbericht, Berlin, 2018.

Parameters

in	*cam*	- measured cloud
in	*comp*	- cloud of the component
in	*thresholds*	- threshold vector to classify points (threshold_outlier,threshold_defect,threshold_warning)
out	*comp_hausdorff*	- colored pointcloud to visualize defects
out	*hausdorff*	- hausdorff distance
out	*avg_dist*	- average distance of all points
in	*outlierDeletion*	- If true, delets all points with a distance greater than threshold_outlier to the nearest neighbor in the other cloud. Those points are also not considered for hausdorff und avg_dist calculation.
in	*noCloudNeeded*	- If true, the part for creating the comp_hausdorff cloud is skipped

Returns

true if comp contains defects, false otherwise

3.1.3.10 visualizeTolerances()

```
void Comparison::visualizeTolerances (
            PTC::Ptr cam,
            std::vector< Dimension > tolerances,
            std::string name = "Tolerances" )
```

Visualizes all tolerances.

Parameters

in	*cam*	- the cloud on which the tolerance definition is based
in	*tolerances*	- the vector of tolerances used
in	*name*	- name of the visualizer window

The documentation for this class was generated from the following file:

- include/Comparison.h

3.2 Depthcam Class Reference

Real Sense Wrapper class.

`#include <Depthcam.h>`

Public Member Functions

- Depthcam (rs2::context ctx, rs2::device &dev, std::string preset, int disparity_shift, int laser_power, std::string debug)
- void stop ()
- void ApplyPreset (std::string path, rs400::advanced_mode &advanced_mode_dev)
- void SetupDictionary (int width, int height, double squareLength, double markerLength, cv::aruco::PREDE↩ FINED_DICTIONARY_NAME dict_name)
- void createPCLPointCloud (double spat_alpha, double spat_delta, double spat_magnitude, double temp_↩ alpha, double temp_delta, int temp_persistency)
- void removeBackground (PTC::Ptr cloud, bool RemoveUnderground)
- bool detectChArUcoCorners (std::vector< cv::Point2f > &charucoCorners, std::vector< int > &charucoIds, bool Intrinsics)
- bool estimateExtrinsics (std::vector< int > charucoIds, std::vector< cv::Point2f > charucoCorners)
- bool calibrateIntrinsics ()
- bool calibrateExtrinsics ()
- void determineFromMasterCam (Eigen::Matrix4f &MasterCam_ExtrinsicMatrix)
- void calibrateStatic (Eigen::Matrix4f &MasterCam_ExtrinsicMatrix)
- void refineCalibration (Eigen::Matrix4f &error)
- std::string getDeviceInfo ()
- void transformationError (std::vector< cv::Vec3d > master_tvecs, std::vector< int > master_ids, Eigen::↩ Matrix4f &master_extrinsics)

Public Attributes

- EIGEN_MAKE_ALIGNED_OPERATOR_NEW cv::Mat color_mat
- PTC::Ptr cloud
- pcl::PointCloud< pcl::PointXYZRGB >::Ptr cloudRGB
- rs2::device dev
- Eigen::Matrix4f ColorToDepth
- Eigen::Matrix4f ExtrinsicMatrix
- Eigen::Matrix3d IntrinsicMatrix
- Eigen::Matrix< double, 5, 1 > DistortionCoeff
- Eigen::Matrix4f FromMasterCam
- PTC::Ptr background
- cv::Ptr< cv::aruco::Dictionary > dictionary = new cv::aruco::Dictionary
- cv::Ptr< cv::aruco::CharucoBoard > board = new cv::aruco::CharucoBoard
- std::vector< cv::Vec3d > corner_tvecs
- std::vector< std::vector< cv::Point2f > > corners
- std::vector< int > ids
- std::vector< double > ReprojectionErrors
- std::vector< int > DetectedMarkers

3.2.1 Detailed Description

Real Sense Wrapper class.

This class allows to create pointclouds, remove the background in a pointclouds, perform the RGB intrinsic and extrinsic calibration. Sources: https://docs.opencv.org/3.4.9/df/d4a/tutorial_charuco_detection.html, https://github.com/IntelRealSense/librealsense/issues/1601, https://github.com/IntelRealSense/librealsense/issues/1021

Author

Justin Heinz

3.2.2 Constructor & Destructor Documentation

3.2.2.1 Depthcam()

```
Depthcam::Depthcam (
            rs2::context ctx,
            rs2::device & dev,
            std::string preset,
            int disparity_shift,
            int laser_power,
            std::string debug )
```

Create the depthcam object, which provides the functions to interact with the realsense library.

Parameters

in	*ctx*	- context of the real sense camera
in	*dev*	- device of the real sense camera
in	*preset*	- preset which should be used for recording
in	*disparity_shift*	- the disparity shift to use
in	*laser_power*	- the value for the laser power
in	*debug*	- Debug level(0 - none,1 - some,2 - all)

3.2.3 Member Function Documentation

3.2.3.1 ApplyPreset()

```
void Depthcam::ApplyPreset (
            std::string path,
            rs400::advanced_mode & advanced_mode_dev )
```

Apply a preset to the real sense camera

Parameters

in	*path*	- path to the preset json file
in	*advanced_mode_dev*	- pointer to the device in advanced mode

3.2.3.2 calibrateExtrinsics()

```
bool Depthcam::calibrateExtrinsics ( )
```

Performs the extrinsic calibration between the color stream and the world coordinate system using a charuco board.

Returns

true when successful, false otherwise

3.2.3.3 calibrateIntrinsics()

```
bool Depthcam::calibrateIntrinsics ( )
```

Performs the RGB intrinsic calibration using 10 pictures of a charuco board.

Returns

true when successful, false otherwise

3.2.3.4 calibrateStatic()

```
void Depthcam::calibrateStatic (
            Eigen::Matrix4f & MasterCam_ExtrinsicMatrix )
```

Determine the extrinsic matrix based on the cam-to-cam transformation.

Parameters

in	*MasterCam_ExtrinsicMatrix*	- extrinsic matrix of the master camera

3.2.3.5 createPCLPointCloud()

```
void Depthcam::createPCLPointCloud (
```

```
            double spat_alpha,
            double spat_delta,
            double spat_magnitude,
            double temp_alpha,
            double temp_delta,
            int temp_persistency )
```

Wait for frames of the camera, than create the librealsense point cloud and convert it to the PCL format.

Parameters

in	*spat_alpha*	- alpha parameter for the spatial filter
in	*spat_delta*	- delta parameter for the spatial filter
in	*spat_magnitude*	- magnitude parameter for the spatial filter
in	*temp_alpha*	- alpha parameter for the temporal filter
in	*temp_delta*	- delta parameter for the temporal filter
in	*temp_persistency*	- persistency parameter for the temporal filter

3.2.3.6 detectChArUcoCorners()

```
bool Depthcam::detectChArUcoCorners (
            std::vector< cv::Point2f > & charucoCorners,
            std::vector< int > & charucoIds,
            bool Intrinsics )
```

Detect charuco corners. Source: https://docs.opencv.org/3.4.9/df/d4a/tutorial_charuco_detection.html

Parameters

out	*charucoCorners*	- the 2D positions of the detected corners
out	*charucoIds*	- the IDs of the detected corners
in	*Intrinsics*	- When true, uses the intrinsic matrix to derive the translation vectors to the individual corners

Returns

true when some corners could be detected, false if not enough corners have been found

3.2.3.7 determineFromMasterCam()

```
void Depthcam::determineFromMasterCam (
            Eigen::Matrix4f & MasterCam_ExtrinsicMatrix )
```

Determine the cam-to-cam transformation based on its own extrinsic matrix.

Parameters

in	*MasterCam_ExtrinsicMatrix*	- extrinsic matrix of the master camera

3.2.3.8 estimateExtrinsics()

```
bool Depthcam::estimateExtrinsics (
            std::vector< int > charucoIds,
            std::vector< cv::Point2f > charucoCorners )
```

Uses a set of charuco corners to derive the extrinsic matrix. Source: https://docs.opencv.org/3.4.9/df/d4a/tutorial_charuco_detection.html

Parameters

out	*charucoIds*	- the IDs of the detected corners
out	*charucoCorners*	- the 2D positions of the detected corners

Returns

true when the extrinsic matrix could be determined, false when not enough markers could be used

3.2.3.9 getDeviceInfo()

```
std::string Depthcam::getDeviceInfo ( )
```

Returns the device info representing the realsense camera.

Returns

device number

3.2.3.10 refineCalibration()

```
void Depthcam::refineCalibration (
            Eigen::Matrix4f & error )
```

Refines the extrinsic calibration.

Parameters

in	*error*	- matrix representing the calibration error

3.2.3.11 removeBackground()

```
void Depthcam::removeBackground (
            PTC::Ptr cloud,
            bool RemoveUnderground )
```

Remove everything which is not on the charuco board.

Parameters

in,out	*cloud*	- the cloud whose background should be removed
in	*RemoveUnderground*	- When true, also remove the points of the charuco board itself

3.2.3.12 SetupDictionary()

```
void Depthcam::SetupDictionary (
            int width,
            int height,
            double squareLength,
            double markerLength,
            cv::aruco::PREDEFINED_DICTIONARY_NAME dict_name )
```

Setup the dictionary of the charuco board.

Parameters

in	*width*	- number of markers in horizontal direction
in	*height*	- number of markers in vertical direction
in	*squareLenght*	- length of one square
in	*markerLenght*	- length of one marker
in	*dict_name*	- type of dictionary to use

3.2.3.13 stop()

```
void Depthcam::stop ( )
```

Stops the pipeline.

3.2.3.14 transformationError()

```
void Depthcam::transformationError (
            std::vector< cv::Vec3d > master_tvecs,
```

```
            std::vector< int > master_ids,
            Eigen::Matrix4f & master_extrinsics )
```

Calculates the error of the cam-to-cam calibration.

Parameters

in	*master_tvecs*	- translational vector of all corners detected by the master cam
in	*master_ids*	- ids of all corners detected by the master cam
in	*master_extrinsics*	- extrinsic matrix of the master camera

3.2.4 Member Data Documentation

3.2.4.1 background

```
PTC::Ptr Depthcam::background
```

Point cloud of the background (charuco board)

3.2.4.2 board

```
cv::Ptr<cv::aruco::CharucoBoard> Depthcam::board = new cv::aruco::CharucoBoard
```

The charuco board used

3.2.4.3 cloud

```
PTC::Ptr Depthcam::cloud
```

Last recorded point cloud

3.2.4.4 cloudRGB

```
pcl::PointCloud<pcl::PointXYZRGB>::Ptr Depthcam::cloudRGB
```

Last recorded colored point cloud

3.2.4.5 color_mat

```
EIGEN_MAKE_ALIGNED_OPERATOR_NEW cv::Mat Depthcam::color_mat
```

Last recorded color matrix.

3.2.4.6 ColorToDepth

```
Eigen::Matrix4f Depthcam::ColorToDepth
```

Extrinsic Matrix between the color stream and the depth stream.

3.2.4.7 corner_tvecs

```
std::vector<cv::Vec3d> Depthcam::corner_tvecs
```

The translation vectors from the color stream to the individual corners of the charuco board

3.2.4.8 corners

```
std::vector<std::vector<cv::Point2f> > Depthcam::corners
```

The 2D points of the corners detected

3.2.4.9 DetectedMarkers

```
std::vector<int> Depthcam::DetectedMarkers
```

Vector of numbers of detected markers

3.2.4.10 dev

```
rs2::device Depthcam::dev
```

Reference to the realsense device

3.2.4.11 dictionary

```
cv::Ptr<cv::aruco::Dictionary> Depthcam::dictionary = new cv::aruco::Dictionary
```

Dictionary used for the charuco board

3.2.4.12 DistortionCoeff

```
Eigen::Matrix<double,5,1> Depthcam::DistortionCoeff
```

Distortion coefficients of the color stream

3.2.4.13 ExtrinsicMatrix

```
Eigen::Matrix4f Depthcam::ExtrinsicMatrix
```

Extrinsic Matrix between the color stream and the world coordinate

3.2.4.14 FromMasterCam

```
Eigen::Matrix4f Depthcam::FromMasterCam
```

Transformation between the color stream and the color stream of the master camera

3.2.4.15 ids

```
std::vector<int> Depthcam::ids
```

The IDs of the corners detected

3.2.4.16 IntrinsicMatrix

```
Eigen::Matrix3d Depthcam::IntrinsicMatrix
```

Intrinsic Matrix of the color stream

3.2.4.17 ReprojectionErrors

```
std::vector<double> Depthcam::ReprojectionErrors
```

Vector of reprojection errors

The documentation for this class was generated from the following file:

- include/Depthcam.h

3.3 Dimension Struct Reference

Struct for saving all important information about a tolerance.

```
#include <Dimension.hpp>
```

Public Attributes

- EIGEN_MAKE_ALIGNED_OPERATOR_NEW int id
- std::vector< PTC::Ptr > references
- std::vector< double > coeffs
- int type
- PT textposition
- double value
- double **upperDelta**
- double **lowerDelta**
- double measured
- bool ok
- std::vector< PTC::Ptr > measuredReferences

3.3.1 Detailed Description

Struct for saving all important information about a tolerance.

Author

Justin Heinz

3.3.2 Member Data Documentation

3.3.2.1 coeffs

```
std::vector<double> Dimension::coeffs
```

Coefficients of the referenced geometries used to define the tolerance.

3.3.2.2 id

```
EIGEN_MAKE_ALIGNED_OPERATOR_NEW int Dimension::id
```

Tolerance ID as defined in the JT file.

3.3.2.3 measured

```
double Dimension::measured
```

The measured value for the tolerance.

3.3.2.4 measuredReferences

```
std::vector<PTC::Ptr> Dimension::measuredReferences
```

Clouds of the referenced geometries used to measure the value for the toleranced.

3.3.2.5 ok

```
bool Dimension::ok
```

Set to true, if measured value is within specified limits. False otherwise.

3.3.2.6 references

```
std::vector<PTC::Ptr> Dimension::references
```

Clouds of the referenced geometries used to define the tolerance.

3.3.2.7 textposition

```
PT Dimension::textposition
```

Point to define the position of the 3D text.

3.3.2.8 type

```
int Dimension::type
```

Type of the tolerance: 1 - linear 2 - angular 3 - Radial 4 - curve length (not implemented) 5 - flatness 6 - circularity

3.3.2.9 value

```
double Dimension::value
```

The value of the tolerance and the upper and lower allowed deviation.

The documentation for this struct was generated from the following file:

- include/Dimension.hpp

3.4 FileIO Class Reference

Class for file input and output for Clouds, Tolerance Definitions, Matrices and the Results format.

```
#include <FileIO.h>
```

Public Member Functions

- FileIO ()
- void loadParameters (std::map< std::string, std::string > &parameters)
- void saveParameters (std::map< std::string, std::string > &parameters)
- template<typename T >
  bool loadCloud (std::string text, typename pcl::PointCloud< T >::Ptr cloud, std::string def="", bool scaling↩
  Required=false)
- template<typename T >
  void saveCloud (std::string path, typename pcl::PointCloud< T >::Ptr cloud)
- bool loadTolerances (std::string path="")
- template<typename T >
  bool queryScaling (std::string path, typename pcl::PointCloud< T >::Ptr cloud=pcl::PointCloud< T >↩
  ::Ptr(new pcl::PointCloud< T >))
- void saveResults ()
- bool loadResults (std::string path)
- template<typename scalar , int rows, int cols>
  void saveMatrix (std::string path, typename Eigen::Matrix< scalar, rows, cols > &mat)
- template<typename scalar , int rows, int cols>
  bool loadMatrix (std::string path, Eigen::Matrix< scalar, rows, cols > &mat)

Public Attributes

- EIGEN_MAKE_ALIGNED_OPERATOR_NEW std::vector< Dimension > tolerances

3.4.1 Detailed Description

Class for file input and output for Clouds, Tolerance Definitions, Matrices and the Results format.

Author

Justin Heinz, Siemens PLM Software

3.4.2 Constructor & Destructor Documentation

3.4.2.1 FileIO()

```
FileIO::FileIO ( )  [inline]
```

Default constructor.

3.4.3 Member Function Documentation

3.4.3.1 loadCloud()

```
template<typename T >
bool FileIO::loadCloud (
            std::string text,
            typename pcl::PointCloud< T >::Ptr cloud,
            std::string def = "",
            bool scalingRequired = false )
```

Query a user for a path to a pointcloud and load it. Supported file formats: .pcd, .stl, .ply, .txt, .jt For .txt files, loadResults(path) is called. For .jt files, loadTolerances(path) is called.

Parameters

in	*text*	- Text to show to the user to ask for a path.
out	*cloud*	- the cloud loaded as a response
in	*def*	- When set, it is the default path name to use, when the user types "default".
in	*scalingRequired*	- When set, queryScaling is called.

Returns

true when the cloud was loaded successfully, false when an error occured

3.4.3.2 loadMatrix()

```
template<typename scalar , int rows, int cols>
bool FileIO::loadMatrix (
            std::string path,
            Eigen::Matrix< scalar, rows, cols > & mat )  [inline]
```

Load an Eigen Matrix from a text file.

Parameters

in	*path*	- Where to load the matrix from
out	*mat*	- Loaded matrix

Returns

true when successful, false when an error occured

3.4.3.3 loadParameters()

```
void FileIO::loadParameters (
            std::map< std::string, std::string > & parameters )
```

Reads in parameters.txt. All parameters are stored using the structure "Key:Value\n". If the file does not exist, saveparameters(parameters) is called.

Parameters

in,out	*parameters*	- the map of parameters

3.4.3.4 loadResults()

```
bool FileIO::loadResults (
            std::string path )
```

Load in the results file.

Parameters

in	*path*	- The path to the results file.

Returns

true when successful, false when an error occured or if the results file is in an invalid format

3.4.3.5 loadTolerances()

```
bool FileIO::loadTolerances (
            std::string path = "" )
```

Load tolerances for the tolerance comparison. At the end queryScaling() is called. Source: Parts of this procedure have been copied from an example by Siemens PLM.

Parameters

in	*path*	- Path to load the tolerance definition from. When empty, the user is queried to enter a path.

Returns

true when the tolerance definition was loaded successfully, false when an error occured

3.4.3.6 queryScaling()

```
template<typename T >
bool FileIO::queryScaling (
            std::string path,
            typename pcl::PointCloud< T >::Ptr cloud = pcl::PointCloud< T >::Ptr(new pcl:↩
:PointCloud< T >) )
```

Query the user to enter the unit which is used for the cloud. The cloud is scaled accordingly. Supported↩ : mikro,mm,cm,dm,m.

Parameters

in	*path*	- The path to the cloud.
in,out	*cloud*	- The pointer to the scaled cloud. If empty, tolerance is scaled instead.

Returns

true when successful, false when an error occured or an invalid unit was entered.

3.4.3.7 saveCloud()

```
template<typename T >
void FileIO::saveCloud (
            std::string path,
            typename pcl::PointCloud< T >::Ptr cloud )
```

Save a pointcloud in the .pcd format.

Parameters

in	*path*	- Path to the location where the cloud should be saved.
in	*cloud*	- the cloud to be saved.

3.4.3.8 saveMatrix()

```
template<typename scalar , int rows, int cols>
void FileIO::saveMatrix (
            std::string path,
            typename Eigen::Matrix< scalar, rows, cols > & mat )  [inline]
```

Save an Eigen Matrix with a specified format in a text file.

Parameters

in	*path*	- Where to save the text file
in	*mat*	- Matrix to save

3.4.3.9 saveParameters()

```
void FileIO::saveParameters (
            std::map< std::string, std::string > & parameters )
```

Writes parameters to parameters.txt. All parameters are stored using the structure "Key:Value\n".

Parameters

in	*parameters*	- the map of parameters

3.4.3.10 saveResults()

```
void FileIO::saveResults ( )
```

Save the tolerances in Results.txt. Format: id:textposition:value:measured:ok

3.4.4 Member Data Documentation

3.4.4.1 tolerances

```
EIGEN_MAKE_ALIGNED_OPERATOR_NEW std::vector<Dimension> FileIO::tolerances
```

Vector of type Dimension to store the tolerances.

The documentation for this class was generated from the following file:

- include/FileIO.h

3.5 Menu Class Reference

Core Class for all functions the user can execute.

```
#include <Menu.h>
```

Public Member Functions

- Menu ()
- void reconnect ()
- void displayIntro ()
- void extrinsics ()
- void intrinsics ()
- void record ()
- void registration ()
- void merge ()
- void hausdorff ()
- void settings ()
- void view ()
- void snap ()
- void tolerances ()

3.5.1 Detailed Description

Core Class for all functions the user can execute.

This class is used to perform the Image Recording using the Depthcam class, the Registration and the Comparison. The user can invoke the member functions. Furthermore, all parameters used in this program are defined here.

Author

Justin Heinz

3.5.2 Constructor & Destructor Documentation

3.5.2.1 Menu()

```
Menu::Menu ( )
```

Standard constructor

3.5.3 Member Function Documentation

3.5.3.1 displayIntro()

```
void Menu::displayIntro ( )
```

Display the help text.

3.5.3.2 extrinsics()

```
void Menu::extrinsics ( )
```

Perform the extrinsic calibration.

3.5.3.3 hausdorff()

```
void Menu::hausdorff ( )
```

Query the user for the cloud after registration and the cloud of the component. Downsample both with the minimum leaf size and perform the hausdorff comparison.

3.5.3.4 intrinsics()

```
void Menu::intrinsics ( )
```

Perform the RGB intrinsic calibration

3.5.3.5 merge()

```
void Menu::merge ( )
```

Merge two clouds.

3.5.3.6 reconnect()

```
void Menu::reconnect ( )
```

Clear the Cams vector and try to connect to the cameras again.

3.5.3.7 record()

```
void Menu::record ( )
```

Query the user how many scans should be recorded and how many degree the turntable should be rotated after each scan. Then start the recording doing the following in a loop: Extrinsic calibration of the master cam, deduce the extrinsic calibration of the slave cameras using the FromMasterCam Transformation, record point clouds, remove the background, save the clouds and rotate the turntable. Afterwards, the whole cloud is postprocessed and downsampled and then saved.

3.5.3.8 registration()

```
void Menu::registration ( )
```

Query the user for the measured cloud and the cloud of the component, calculate the maximum distance between two points in the component cloud and use that information to downsample both clouds. Afterwards, the normals of both clouds are estimated and they are used for the initial alignment and then for ICP. Save the cloud after registration.

3.5.3.9 settings()

```
void Menu::settings ( )
```

Display all parameters and their current values. Then allow the user to type a parameter and its new value.

3.5.3.10 snap()

```
void Menu::snap ( )
```

Record a point cloud from all cameras and show the output. Save the recorded cloud.

3.5.3.11 tolerances()

```
void Menu::tolerances ( )
```

Query the user for a tolerance definition (.jt) and a point cloud. Perform the tolerance comparison and visualize the tolerances.

3.5.3.12 view()

```
void Menu::view ( )
```

View a point cloud, tolerance definition or the results file.

The documentation for this class was generated from the following file:

- include/Menu.h

3.6 Registration Class Reference

Class for the subproblem of the registration.

`#include <Registration.h>`

Public Member Functions

- EIGEN_MAKE_ALIGNED_OPERATOR_NEW Registration (std::string debug)
- template<typename T >
  void downsample (typename pcl::PointCloud< T >::Ptr cloud, typename pcl::PointCloud< T >::Ptr cloud_↩
  sparse, double &leafSize, int max_numberOfPoints=std::numeric_limits< int >::max())
- template<typename T >
  void postprocess (typename pcl::PointCloud< T >::Ptr cloud, int meanK, double thresh)
- pcl::PointCloud< pcl::Normal >::Ptr getNormals (PTC::Ptr cloud, double searchRadius)
- pcl::PointCloud< pcl::FPFHSignature33 >::Ptr getLocalFeatures (PTC::Ptr cloud, pcl::PointCloud< pcl::↩
  Normal >::Ptr cloud_normals, double searchRadius)
- bool findInitialGuess (PTC::Ptr cloud_source, pcl::PointCloud< pcl::Normal >::Ptr source_normals, PTC::Ptr cloud_target, pcl::PointCloud< pcl::Normal >::Ptr target_normals, double &score, Eigen::Matrix4f &transformation, double minSampleDistance, double correspondenceDistance, int maxIterations, double search↩
  Radius)
- void demean (PTC::Ptr cloudCentroid, PTC::Ptr cloudDemean, bool inverse=false)
- bool cloudRegistration (PTC::Ptr cloud_source, pcl::PointCloud< pcl::Normal >::Ptr source_normals, P↩
  TC::Ptr cloud_target, pcl::PointCloud< pcl::Normal >::Ptr target_normals, double &score, Eigen::Matrix4f &transformation, double correspondenceDistance, int maxIterations, double maxFitnessScore=0)

3.6.1 Detailed Description

Class for the subproblem of the registration.

This class allows postprocessing functionalities, estimation of normals, downsampling and performing the registration using SAC Initial Alignment or ICP. Source: http://pointclouds.org/documentation/tutorials/template_alignment.php

Author

Justin Heinz

3.6.2 Constructor & Destructor Documentation

3.6.2.1 Registration()

```
EIGEN_MAKE_ALIGNED_OPERATOR_NEW Registration::Registration (
            std::string debug )
```

Constructor of the registration class.

Parameters

in	*debug*	- how many debug info should be displayed (0 - none,1 - some,2 - all)

3.6.3 Member Function Documentation

3.6.3.1 cloudRegistration()

```
bool Registration::cloudRegistration (
            PTC::Ptr cloud_source,
            pcl::PointCloud< pcl::Normal >::Ptr source_normals,
            PTC::Ptr cloud_target,
            pcl::PointCloud< pcl::Normal >::Ptr target_normals,
            double & score,
            Eigen::Matrix4f & transformation,
            double correspondenceDistance,
            int maxIterations,
            double maxFitnessScore = 0 )
```

Performs the pose refinement using ICP. Good initial pose estimation needed.

Parameters

in,out	*cloud_source*	- cloud which has to be aligned
in	*source_normals*	- normals of the source cloud
in	*cloud_target*	- target cloud
in	*target_normals*	- normals of the target cloud
out	*score*	- fitness score of the registration
out	*transformation*	- estimated transformation to align clouds
in	*correspondenceDistance*	- maximum allowed initial distance between two features of the clouds
in	*maxIterations*	- number of iterations used
in	*maxFitnessScore*	- When greater than zero, this parameter is used as a second termination criterium. When the fitness score between two consecutive steps is below this value, ICP is considered to have converged.

3.6.3.2 demean()

```
void Registration::demean (
            PTC::Ptr cloudCentroid,
            PTC::Ptr cloudDemean,
            bool inverse = false )
```

Calculate the centroid of a cloud and use it to demean a second cloud.

Parameters

in	*cloudCentroid*	- cloud used for the centroid calculation
in,out	*cloudDemean*	- cloud to demean
in	*inverse*	- If true, the calculated centroid is inverted. Assuming an already demeaned cloudDemean, this flag can be used to move cloudDemean to the centroid of cloudCentroid.

3.6.3.3 downsample()

```
template<typename T >
void Registration::downsample (
            typename pcl::PointCloud< T >::Ptr cloud,
            typename pcl::PointCloud< T >::Ptr cloud_sparse,
            double & leafSize,
            int max_numberOfPoints = std::numeric_limits< int >::max() )
```

Uses a voxelgrid to downsample a given point cloud.

Parameters

in	*cloud*	- the cloud to be downsampled
out	*cloud_sparse*	- the downsampled cloud
in	*leafSize*	- Leafsize to use for the voxel grid
in	*max_numberOfPoints*	- maximum number of points in the cloud after downsampling. When the number is higher, the leafsize is increased and the downsampling is performed again.

3.6.3.4 findInitialGuess()

```
bool Registration::findInitialGuess (
            PTC::Ptr cloud_source,
            pcl::PointCloud< pcl::Normal >::Ptr source_normals,
            PTC::Ptr cloud_target,
            pcl::PointCloud< pcl::Normal >::Ptr target_normals,
            double & score,
            Eigen::Matrix4f & transformation,
            double minSampleDistance,
            double correspondenceDistance,
            int maxIterations,
            double searchRadius )
```

Find the initial 6D pose of the component in the cam cloud pointer using FPFH. Source: http↩
://pointclouds.org/documentation/tutorials/template_alignment.php

Parameters

in,out	*cloud_source*	- cloud which has to be aligned
in	*source_normals*	- normals of the source cloud
in	*cloud_target*	- target cloud
in	*target_normals*	- normals of the target cloud
out	*score*	- fitness score of the registration
out	*transformation*	- estimated transformation to align clouds
in	*minSampleDistance*	- minimum distance between chosen samples
in	*correspondenceDistance*	- maximum allowed initial distance between two features of the clouds
in	*maxIterations*	- number of iterations used
in	*searchRadius*	- used for the FPFH estimation

3.6.3.5 getLocalFeatures()

```
pcl::PointCloud<pcl::FPFHSignature33>::Ptr Registration::getLocalFeatures (
            PTC::Ptr cloud,
            pcl::PointCloud< pcl::Normal >::Ptr cloud_normals,
            double searchRadius )
```

Esimate the fast point feature histograms for a cloud

Parameters

in	*cloud*	- cloud used for FPFH estimation
in	*cloud_normals*	- the normals of the cloud
in	*searchRadius*	- the radius used for FPFH estimation

Returns

pointer to the estimated FPFHs

3.6.3.6 getNormals()

```
pcl::PointCloud<pcl::Normal>::Ptr Registration::getNormals (
            PTC::Ptr cloud,
            double searchRadius )
```

Estimate the normals for a cloud.

Parameters

in	*cloud*	- cloud used for normal estimation
in	*searchRadius*	- used for normal estimation

Returns

pointer to calculated normals

3.6.3.7 postprocess()

```
template<typename T >
void Registration::postprocess (
            typename pcl::PointCloud< T >::Ptr cloud,
            int meanK,
            double thresh )
```

Uses a statistical outlier removal filter to postprocess clouds.

Parameters

in,out	*cloud*	- the cloud to postprocess
in	*meanK*	- the number of neighbors to use for calculating the mean
in	*thresh*	- the maximum allowed distance of a point with respect to the calculated mean as a multiple of the standard deviation

The documentation for this class was generated from the following file:

- include/Registration.h

3.7 Visual Class Reference

Class for Visualization of Point Clouds.

```
#include <Visual.h>
```

Public Member Functions

- EIGEN_MAKE_ALIGNED_OPERATOR_NEW Visual ()
- Visual (std::string name, PTC::Ptr cloud)
- Visual (std::string name, PTC::Ptr cloud, PTC::Ptr component)
- Visual (std::string name, PTC::Ptr cloudA, PTC::Ptr cloudB, PTC::Ptr cloudC)
- Visual (std::string name, pcl::PointCloud< pcl::PointXYZRGB >::Ptr cloud)
- void updateCloud (PTC::Ptr cloud)
- void updateCloud (PTC::Ptr cloud, PTC::Ptr component)
- void updateCloud (PTC::Ptr cloudA, PTC::Ptr cloudB, PTC::Ptr cloudC)
- void updateCloud (pcl::PointCloud< pcl::PointXYZRGB >::Ptr colorCloud)
- void processOutput ()
- void addText (std::string text, char color='0')
- void addText (std::string text, PT position, int r=255, int g=255, int b=255)
- void addNormals (PTC::Ptr cloud, pcl::PointCloud< pcl::Normal >::Ptr normals)
- void closeViewer ()

3.7.1 Detailed Description

Class for Visualization of Point Clouds.

Author

Justin Heinz

3.7.2 Constructor & Destructor Documentation

3.7.2.1 Visual() [1/5]

```
EIGEN_MAKE_ALIGNED_OPERATOR_NEW Visual::Visual ( )
```

Standard constructor

3.7.2.2 Visual() [2/5]

```
Visual::Visual (
            std::string name,
            PTC::Ptr cloud )
```

Constructor to show the camera cloud in a new PCLVisualizer window with a specific name.

Parameters

in	*name*	- the name of the window
in	*cloud*	- the cloud to be visualized (green)

3.7.2.3 Visual() [3/5]

```
Visual::Visual (
            std::string name,
            PTC::Ptr cloud,
            PTC::Ptr component )
```

Constructor to show the camera cloud as well as the component cloud in a new PCLVisualizer window with a specific name.

Parameters

in	*name*	- the name of the window
in	*cloud*	- the camera cloud to be visualized (green)
in	*component*	- the component cloud to be visualized (red)

3.7.2.4 Visual() [4/5]

```
Visual::Visual (
            std::string name,
            PTC::Ptr cloudA,
            PTC::Ptr cloudB,
            PTC::Ptr cloudC )
```

Constructor to show three clouds in a new PCLVisualizer window with a specific name.

Parameters

in	*name*	- the name of the window
in	*cloudA*	- the first cloud to be visualized (green)
in	*cloudB*	- the second cloud to be visualized (red)
in	*cloudC*	- the third cloud to be visualized (blue)

3.7.2.5 Visual() [5/5]

```
Visual::Visual (
            std::string name,
            pcl::PointCloud< pcl::PointXYZRGB >::Ptr cloud )
```

Constructor to show the a colored cloud in a new PCLVisualizer window with a specific name.

Parameters

in	*name*	- the name of the window
in	*cloud*	- the RGB cloud to be visualized

3.7.3 Member Function Documentation

3.7.3.1 addNormals()

```
void Visual::addNormals (
            PTC::Ptr cloud,
            pcl::PointCloud< pcl::Normal >::Ptr normals )
```

Adds normals to a cloud in the visualizer.

Parameters

in	*cloud*	- the cloud to attach normals to
in	*normals*	- pointer to the normals to use

3.7.3.2 addText() [1/2]

```
void Visual::addText (
            std::string text,
            char color = '0' )
```

Adds a 2D Text to the visualizer.

Parameters

in	*text*	- the text to display
in	*color*	- For "r", the text is displayed in red with a slight vertical offset in the upper left hand corner. For "g", the text is displayed in green in the upper left hand corner. Else, the text is displayed in white in the upper left hand corner.

3.7.3.3 addText() [2/2]

```
void Visual::addText (
            std::string text,
            PT position,
            int r = 255,
            int g = 255,
            int b = 255 )
```

Adds a 3D Text to the visualizer.

Parameters

in	*text*	- the text to display
in	*position*	- Sets the position of the upper left hand corner of the text.
in	*r*	- Sets the red value
in	*g*	- Sets the green value
in	*b*	- Sets the blue value

3.7.3.4 closeViewer()

```
void Visual::closeViewer ( )
```

Closes the viewer window.

3.7.3.5 processOutput()

```
void Visual::processOutput ( )
```

Updates the window. Waits for the user to press SPACE.

3.7.3.6 updateCloud() [1/4]

```
void Visual::updateCloud (
            pcl::PointCloud< pcl::PointXYZRGB >::Ptr colorCloud )
```

Updates the colored cloud visualized.

Parameters

in	*cloud*	- the cloud to be updated

3.7.3.7 updateCloud() [2/4]

```
void Visual::updateCloud (
            PTC::Ptr cloud )
```

Updates the cloud visualized.

Parameters

in	*cloud*	- the cloud to be updated

3.7.3.8 updateCloud() [3/4]

```
void Visual::updateCloud (
            PTC::Ptr cloud,
            PTC::Ptr component )
```

Updates the clouds visualized.

Parameters

in	*cloud*	- the cloud to be updated
in	*component*	- the component cloud to be updated

3.7.3.9 updateCloud() [4/4]

```
void Visual::updateCloud (
            PTC::Ptr cloudA,
            PTC::Ptr cloudB,
            PTC::Ptr cloudC )
```

Updates the clouds visualized.

Parameters

in	*cloudA*	- the first cloud to be updated
in	*cloudB*	- the second cloud to be updated
in	*cloudC*	- the third cloud to be updated

The documentation for this class was generated from the following file:

- include/Visual.h

Index