Handle Detector

1.0.0

Generated by Doxygen 1.7.6.1

Mon Mar 3 2014 15:51:22

Contents

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

Class Documentation

2.1 Affordances Class Reference

Affordances localizes grasp affordances and handles in a point cloud. It also provides helper methods to filter out points from the point cloud that are outside of the robot's workspace.

```
#include <affordances.h>
```

Public Member Functions

- void initParams (ros::NodeHandle node)

  Read the parameters from a ROS launch file.
- PointCloud::Ptr maxRangeFilter (const PointCloud::Ptr &cloud_in)

  Filter out all points from a given cloud that are outside of a sphere with a radius max_-range and center at the origin of the point cloud, and return the filtered cloud.
- PointCloud::Ptr workspaceFilter (const PointCloud::Ptr &cloud_in)

  Filter out all points from a given cloud that are outside of a cube defined by the workspace limits of the robot, and return the filtered cloud.
- std::vector< CylindricalShell > searchAffordances (const PointCloud::Ptr &cloud)

  Search grasp affordances (cylindrical shells) in a given point cloud.
- std::vector< std::vector < CylindricalShell > > searchHandles (const PointCloud::Ptr &cloud, std::vector< CylindricalShell > shells)

  Search handles in a set of cylindrical shells. If occlusion filtering is turned on (using the corresponding parameter in the ROS launch file), the handles found are filtered on possible occlusions.
- std::string getPCDFile ()

  Return the ∗.pcd file given by the corresponding parameter in the ROS launch file.

2.1.1 Detailed Description

Affordances localizes grasp affordances and handles in a point cloud. It also provides helper methods to filter out points from the point cloud that are outside of the robot's workspace.

Author

Andreas ten Pas

2.1.2 Member Function Documentation

2.1.2.1 void **Affordances::initParams** (ros::NodeHandle *node*)

Read the parameters from a ROS launch file.

Parameters

node	the ROS node with which the parameters are associated

2.1.2.2 PointCloud::Ptr **Affordances::maxRangeFilter** (const PointCloud::Ptr & *cloud_in*)

Filter out all points from a given cloud that are outside of a sphere with a radius max_-range and center at the origin of the point cloud, and return the filtered cloud.

Parameters

cloud_in	the point cloud to be filtered

2.1.2.3 std::vector< **CylindricalShell** > **Affordances::searchAffordances** (const PointCloud::Ptr & *cloud*)

Search grasp affordances (cylindrical shells) in a given point cloud.

Parameters

cloud	the point cloud in which affordances are searched for

2.1.2.4 std::vector< std::vector< **CylindricalShell** > > **Affordances::searchHandles** (const PointCloud::Ptr & *cloud,* std::vector< **CylindricalShell** > *shells*)

Search handles in a set of cylindrical shells. If occlusion filtering is turned on (using the corresponding parameter in the ROS launch file), the handles found are filtered on possible occlusions.

Parameters

cloud	the point cloud in which the handles lie (only required for occlusion filtering)
shells	the set of cylindrical shells to be searched for handles

2.1.2.5 PointCloud::Ptr Affordances::workspaceFilter (const PointCloud::Ptr & *cloud_in*)

Filter out all points from a given cloud that are outside of a cube defined by the workspace limits of the robot, and return the filtered cloud.

Parameters

cloud_in	the point cloud to be filtered

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

- src/affordances.h
- src/affordances.cpp

2.2 pcl::CurvatureEstimationTaubin< PointInT, PointOutT > Class Template Reference

CurvatureEstimationTaubin estimates the curvature for a set of point neighborhoods in the cloud using Taubin Quadric Fitting. This class uses the OpenMP standard to permit parallelized feature computation.

```
#include <curvature_estimation_taubin.h>
```

Public Types

- typedef Feature< PointInT, PointOutT >::PointCloudOut **PointCloudOut**

Public Member Functions

- CurvatureEstimationTaubin (unsigned int num_threads=0)

  Constructor. Set the number of threads to use.

- void fitQuadric (const std::vector< int > &indices, Eigen::VectorXd &quadric_parameters, Eigen::Vector3d &quadric_centroid, Eigen::Matrix3d &quadric_covariance_matrix)

  Fit a quadric to a given set of points, using their indices, and return the parameters of the quadric in implicit form, its centroid, and its covariance matrix. This method uses Taubin Quadric Fitting.

- void estimateMedianCurvature (const std::vector< int > &indices, const Eigen::VectorXd &quadric_parameters, double &median_curvature, Eigen::Vector3d &normal, Eigen::Vector3d &curvature_axis, Eigen::Vector3d &curvature_centroid, bool is_deterministic=false)

  Estimate the median curvature for a given quadric, using the indices of the point neighborhood that the quadric is fitted to and its parameters, and return the estimated curvature, the normal axis, the curvature axis, and the curvature centroid.

- void setNumSamples (int num_samples)

  Set the number of samples (point neighborhoods).

- void setNumThreads (int num_threads)

  Set the number of threads used for parallelizing Taubin Quadric Fitting.

- std::vector< std::vector< int > > const & getNeighborhoods () const

  Get the indices of each point neighborhood.

- std::vector< int > const & getNeighborhoodCentroids () const

  Get the centroid indices of each point neighborhood.

Static Public Member Functions

- static bool isSecondElementSmaller (const std::vector< double > &p1, const std::vector< double > &p2)

  Compares two vectors by looking at their second elements.

Protected Member Functions

- void computeFeature (PointCloudOut &output)

  Estimate the curvature for a set of point neighborhoods sampled from the cloud given by <setInputCloud()>, using the spatial locator in setSearchMethod(). This method creates its own set of randomly selected indices.

2.2.1 Detailed Description

template<typename PointInT, typename PointOutT>class pcl::CurvatureEstimationTaubin< PointInT, PointOutT >

CurvatureEstimationTaubin estimates the curvature for a set of point neighborhoods in the cloud using Taubin Quadric Fitting. This class uses the OpenMP standard to permit parallelized feature computation.

Author

Andreas ten Pas

2.2.2 Constructor & Destructor Documentation

2.2.2.1 template<typename PointInT , typename PointOutT > pcl::CurvatureEstimationTaubin< PointInT, PointOutT >::CurvatureEstimationTaubin (unsigned int *num_threads* = 0) [inline]

Constructor. Set the number of threads to use.

Parameters

num_threads	the number of threads to use (0: automatic)

2.2.3 Member Function Documentation

2.2.3.1 template<typename PointInT , typename PointOutT > void pcl::CurvatureEstimationTaubin< PointInT, PointOutT >::computeFeature (PointCloudOut & *output*) [protected]

Estimate the curvature for a set of point neighborhoods sampled from the cloud given by <setInputCloud()>, using the spatial locator in setSearchMethod(). This method creates its own set of randomly selected indices.

Note

If <setIndices()> is used, the set of indices is not randomly selected.

Parameters

output	the resultant point cloud that contains the curvature, normal axes, curvature axes, and curvature centroids

2.2.3.2 template<typename PointInT , typename PointOutT > void pcl::CurvatureEstimationTaubin< PointInT, PointOutT >::estimateMedianCurvature (const std::vector< int > & *indices*, const Eigen::VectorXd & *quadric_parameters*, double & *median_curvature*, Eigen::Vector3d & *normal*, Eigen::Vector3d & *curvature_axis*, Eigen::Vector3d & *curvature_centroid*, bool *is_deterministic* = false) [inline]

Estimate the median curvature for a given quadric, using the indices of the point neighborhood that the quadric is fitted to and its parameters, and return the estimated curvature, the normal axis, the curvature axis, and the curvature centroid.

Parameters

indices	the point cloud indices of the points
quadric_parameters	the quadric parameters as: a, b, c, d, e, f, g, h, i, j (ax^2 + by^2 + cz^2 + dxy + eyz + fxz + gx + hy + iz + j = 0)
median_curvature	the resultant, estimated median curvature of the quadric

normal	the normal axis of the quadric (direction vector)
curvature_-axis	the curvature axis of the quadric (direction vector)
curvature_-centroid	the centroid of curvature

2.2.3.3 template<typename PointInT , typename PointOutT > void **pcl::CurvatureEstimationTaubin**< PointInT, PointOutT >::**fitQuadric** (const std::vector< int > & *indices,* Eigen::VectorXd & *quadric_parameters,* Eigen::Vector3d & *quadric_centroid,* Eigen::Matrix3d & *quadric_covariance_matrix*) [inline]

Fit a quadric to a given set of points, using their indices, and return the parameters of the quadric in implicit form, its centroid, and its covariance matrix. This method uses Taubin Quadric Fitting.

Parameters

indices	the point cloud indices of the points
quadric_-parameters	the resultant quadric parameters as: a, b, c, d, e, f, g, h, i, j (ax^2 + by^2 + cz^2 + dxy + eyz + fxz + gx + hy + iz + j = 0)
quadric_-centroid	the resultant centroid of the quadric
quadric_-covariance_-matrix	the resultant covariance matrix of the quadric

2.2.3.4 template<typename PointInT , typename PointOutT > static bool **pcl::CurvatureEstimationTaubin**< PointInT, PointOutT >::**isSecondElementSmaller** (const std::vector< double > & *p1,* const std::vector< double > & *p2*) [inline, static]

Compares two vectors by looking at their second elements.

Parameters

p1	the first vector to compare
p2	the second vector to compare

2.2.3.5 template<typename PointInT , typename PointOutT > void **pcl::CurvatureEstimationTaubin**< PointInT, PointOutT >::**setNumSamples** (int *num_samples*) [inline]

Set the number of samples (point neighborhoods).

Parameters

num_-samples	the number of samples

2.2.3.6 template<typename PointInT , typename PointOutT > void pcl::CurvatureEstimationTaubin< PointInT, PointOutT >::setNumThreads (int *num_threads*) [inline]

Set the number of threads used for parallelizing Taubin Quadric Fitting.

Parameters

num_-samples	the number of samples

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

- src/curvature_estimation_taubin.h
- src/curvature_estimation_taubin.hpp

2.3 CylindricalShell Class Reference

CylindricalShell represents a cylindrical shell that consists of two colinear cylinders. A shell consists of an inner and an outer cylinder. The portion of the object to be grasped must fit inside the inner cylinder, and the radius of that cylinder must be no larger than the maximum hand aperture. The gap between the inner and outer cylinder must be free of obstacles and wide enough to be able to contain the robot fingers.

```
#include <cylindrical_shell.h>
```

Public Member Functions

- void fitCylinder (const PointCloud::Ptr &cloud, const std::vector< int > &indices, const Eigen::Vector3d &normal, const Eigen::Vector3d &curvature_axis)

  Fit a cylinder to a set of points in the cloud, using their indices, and the normal and the curvature axis given by the quadric fitting (see curvature_estimation_taubin.h). The fitted cylinder is the inner cylinder of the cylindrical shell.
- bool hasClearance (const PointCloud::Ptr &cloud, double maxHandAperture, double handleGap)

  Check whether the gap between the inner and outer cylinder of the shell is free of obstacles and wide enough to be able to contain the robot fingers.
- double getExtent () const

  Get the extent of the cylindrical shell.
- void setExtent (double extent)

  Set the extent of the cylindrical shell.

- double getRadius () const

  Get the radius of the cylindrical shell.
- int getNeighborhoodCentroidIndex () const

  Get the index of the centroid of the neighborhood associated with the cylindrical shell.
- void setNeighborhoodCentroidIndex (int index)

  Set the index of the centroid of the neighborhood associated with the cylindrical shell.
- Eigen::Vector3d getCentroid () const

  Get the centroid of the cylindrical shell.
- Eigen::Vector3d getCurvatureAxis () const

  Get the curvature axis of the cylindrical shell.
- Eigen::Vector3d getNormal () const

  Get the normal axis of the cylindrical shell.

2.3.1 Detailed Description

CylindricalShell represents a cylindrical shell that consists of two colinear cylinders. A shell consists of an inner and an outer cylinder. The portion of the object to be grasped must fit inside the inner cylinder, and the radius of that cylinder must be no larger than the maximum hand aperture. The gap between the inner and outer cylinder must be free of obstacles and wide enough to be able to contain the robot fingers.

Author

Andreas ten Pas

2.3.2 Member Function Documentation

2.3.2.1 void **CylindricalShell::fitCylinder** (const PointCloud::Ptr & *cloud*, const std::vector< int > & *indices*, const Eigen::Vector3d & *normal*, const Eigen::Vector3d & *curvature_axis*)

Fit a cylinder to a set of points in the cloud, using their indices, and the normal and the curvature axis given by the quadric fitting (see curvature_estimation_taubin.h). The fitted cylinder is the inner cylinder of the cylindrical shell.

Parameters

cloud	the point cloud
indices	the indices of the set of points in the cloud
normal	the normal given by quadric fitting
curvature_axis	the curvature axis given by quadric fitting

2.3.2.2 bool CylindricalShell::hasClearance (const PointCloud::Ptr & *cloud,* double *maxHandAperture,* double *handleGap*)

Check whether the gap between the inner and outer cylinder of the shell is free of obstacles and wide enough to be able to contain the robot fingers.

Parameters

cloud	the point cloud
maxHandAperture	the maximum robot hand aperture
handleGap	the required size of the gap around the handle

2.3.2.3 void CylindricalShell::setExtent (double *extent*) [inline]

Set the extent of the cylindrical shell.

Parameters

extent	the extent

2.3.2.4 void CylindricalShell::setNeighborhoodCentroidIndex (int *index*) [inline]

Set the index of the centroid of the neighborhood associated with the cylindrical shell.

Parameters

index	the index of the centroid

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

- src/cylindrical_shell.h
- src/cylindrical_shell.cpp

2.4 Messages Class Reference

Messages creates custom ROS messages to publish the results of the localization. The messages that can be created are: CylinderArray, Cylinder, and HandleList.

```
#include <messages.h>
```

Public Member Functions

- handle_detector::CylinderArrayMsg createCylinderArray (const std::vector< CylindricalShell > &list, std::string frame)

Create a CylinderArray message from a list of cylindrical shells.

- handle_detector::CylinderMsg createCylinder (const CylindricalShell &shell, std::string frame)

  Create a Cylinder message from a cylindrical shell.
- handle_detector::HandleListMsg createHandleList (const std::vector< std::vector< CylindricalShell > > &handles, std::string frame)

  Create a HandleList message from a list of cylindrical shells.

2.4.1 Detailed Description

Messages creates custom ROS messages to publish the results of the localization. The messages that can be created are: CylinderArray, Cylinder, and HandleList.

Author

Andreas ten Pas

2.4.2 Member Function Documentation

2.4.2.1 handle_detector::CylinderMsg Messages::createCylinder (const CylindricalShell & *shell*, std::string *frame*)

Create a Cylinder message from a cylindrical shell.

Parameters

shell	the cylindrical shell
frame	the frame in which the shell is located

2.4.2.2 handle_detector::CylinderArrayMsg Messages::createCylinderArray (const std::vector< CylindricalShell > & *list*, std::string *frame*)

Create a CylinderArray message from a list of cylindrical shells.

Parameters

list	the list of cylindrical shells
frame	the frame in which the shells are located

2.4.2.3 handle_detector::HandleListMsg Messages::createHandleList (const std::vector< std::vector< CylindricalShell > > & *handles*, std::string *frame*)

Create a HandleList message from a list of cylindrical shells.

Parameters

handles	the list of handles
frame	the frame in which the shells are located

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

- src/messages.h
- src/messages.cpp

2.5 Visualizer Class Reference

Visualizer creates ROS messages to visualize the results of the localization in RViz. The possible objects that can be visualized are: neighborhoods, cylindrical shells, and handles.

```
#include <visualizer.h>
```

Public Member Functions

- Visualizer (double marker_lifetime)

  Constructor. Set the lifetime of markers in RViz.
- MarkerArray createCylinders (const std::vector< CylindricalShell > &list, const std::string &frame)

  Create a MarkerArray message from a list of cylindrical shells.
- void createHandles (const std::vector< std::vector< CylindricalShell > > &handles, const std::string &frame, std::vector< MarkerArray > &marker_arrays, -MarkerArray &all_handle_markers)

  Create a list of MarkerArray messages and a MarkerArray from a list of handles. The former represents each handle as a MarkerArray message, and the latter represents all handles in a single MarkerArray message.

2.5.1 Detailed Description

Visualizer creates ROS messages to visualize the results of the localization in RViz. The possible objects that can be visualized are: neighborhoods, cylindrical shells, and handles.

Author

Andreas ten Pas

2.5.2 Constructor & Destructor Documentation

2.5.2.1 Visualizer::Visualizer (double *marker_lifetime*)

Constructor. Set the lifetime of markers in RViz.

Parameters

num_-threads	the lifetime in seconds

2.5.3 Member Function Documentation

2.5.3.1 MarkerArray Visualizer::createCylinders (const std::vector< CylindricalShell > & *list*, const std::string & *frame*)

Create a MarkerArray message from a list of cylindrical shells.

Parameters

list	the list of cylindrical shells
frame	the frame in which the shells are located

2.5.3.2 void Visualizer::createHandles (const std::vector< std::vector< CylindricalShell > > & *handles*, const std::string & *frame*, std::vector< MarkerArray > & *marker_arrays*, MarkerArray & *all_handle_markers*)

Create a list of MarkerArray messages and a MarkerArray from a list of handles. The former represents each handle as a MarkerArray message, and the latter represents all handles in a single MarkerArray message.

Parameters

handles	the list of handles
frame	the frame in which the handles are located
marker_-arrays	the resultant list of MarkerArray messages
all_handle_-markers	the resultant single MarkerArray message that consists of all handles

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

- src/visualizer.h
- src/visualizer.cpp