OpenRAVE

0.5.0

Generated by Doxygen 1.6.3

Thu Mar 22 20:42:05 2012

Contents

1 Open Robotics Automation Virtual Environment

Version: 0.5.0

This document covers the core OpenRAVE concepts, the C++ API, C++ examples, and C++ usage of the plugins offered in the base installation.

Visit the {**official documentation page**} for information on all other OpenRAVE features.

1.1 Getting Started

- Building and Installing
- Basic Usage and Loading Environments
- {Environment Variables}

1.2 Architecture

- Introduction to the OpenRAVE Architecture
- Base Interface Concepts
- Writing Plugins and Interfaces
- Writing and Using Controllers
- Cloning

1.3 Reference

- Base Interface Classes
- Executable Programs
- Geometric Primitives
- Affine Math
- C++ Examples

Documentation in Other Languages: `Japanese`

2 Basic Usage and Loading Environments

To check out the plugins loaded type

```
openrave --listplugins
```

To load a simple robot into the scene type:

```
openrave robots/puma.robot.xml
```

If a plugin declares a ProblemInsance named **MyProblem**, then to launch the problem at startup type

```
openrave robots/puma.robot.xml -problem MyProblem 'myarguments'
```

The `openrave.py` program is similar to `openrave`, but supports much more functionality. In fact, `openrave` potentially calls into `openrave.py` for commands it cannot support.

3 Writing Plugins and Interfaces

3.1 Contents

Every plugin needs to export several functions as defined in Plugin Export Functions to notify OpenRAVE what interfaces it has. When a plugin is first loaded, it is validated by the environment and its OpenRAVEGetPluginAttributes function will be called in order so the OpenRAVE core can register the names of its provided interfaces. Plugins themselves can query functionality offer by other plugins through the environment's interface querying functions.

3.2 Making a Simple Interface

Example plugincpp.cpp creates a OpenRAVE::ProblemInstance interface named **MyProblem** and have it offer two commands: **numbodies** and **load**.

The first #include the compiler sees has to be openrave/openraveh. Then for the main C++ file, we include openrave/pluginh for several helper functions.

```
#include <openrave/openrave.h>
#include <openrave/plugin.h>
#include <boost/bind.hpp>

using namespace std;
using namespace OpenRAVE;

class MyModule : public ModuleBase
{
```

Now register the two commands of the problem. `boost::bind` is necessary for specifying member functions as callbacks.

Provide the implementations for the member functions:

It is recommend to plugin authors to include openrave/pluginh in their main C++ file, this will provide implementations for the export functions and ask the user to provide a new set of functions CreateInterfaceValidated and GetPluginAttributesValidated.

Providing **MyProblem** would look like:

In order to tell OpenRAVE what is provided, have to define:

3.3 Building the Plugin

3.3.1 Using CMake (Linux and Windows)

The main build system of OpenRAVE is cmake, and FindOpenRAVE.cmake can be used to find the OpenRAVE installation. An example of the `CMakeLists.txt` file for compiling a plugin using FindOpenRAVE.cmake is:

```
cmake_minimum_required (VERSION 2.6)
project (plugincpp)
find_package(OpenRAVE REQUIRED)
include_directories(${OpenRAVE_INCLUDE_DIRS})
link_directories(${OpenRAVE_LIBRARY_DIRS})
add_library(plugincpp SHARED plugincpp.cpp)
set_target_properties(plugincpp PROPERTIES COMPILE_FLAGS "${OpenRAVE_CXX_FLAGS}" LINK_FLAGS "${OpenR
target_link_libraries(plugincpp ${OpenRAVE_LIBRARIES})
```

3.3.2 Other Build Systems

If not using CMake, then here's how the development files are organized:

Linux Users

Depending on where openrave was installed, a openrave-config should have been created in the `$OPENRAVE_INSTALL/bin` directory. It is possible to call `openrave-config --cflags` to get the correct paths and flags to include in gcc to link with `libopenrave.so`.

3.4 Using the Plugin

There are several ways to load the generated plugin.

- The most simplest method is to add its installation directory to {OPENRAVE_-PLUGINS}. OpenRAVE will automatically load it up at start up. You can confirm this is the case using:

```
openrave --listplugins
```

- A more explicit way is to load it from the command line using any one of the following methods:

```
openrave --loadplugin $SOMEPATH/libplugincpp
openrave --loadplugin $SOMEPATH/libplugincpp.so
openrave --loadplugin ./libplugincpp.so
```

  where `$SOMEPATH` is the absolute/relative path of the shared object.

- Another way is to load it from the C++/Python/APIs:

  C++

```
RaveLoadPlugin(env,"plugincpp");
```

  Python

```
RaveLoadPlugin('plugincpp')
```

  Octave

```
orEnvLoadPlugin('plugincpp');
```

Once the plugin is loaded, we can create the interface and call its commands to load an environment and return the number of bodies:

C++

```
ProblemInstancePtr prob = RaveCreateProblem(env,"MyProblem");
env->AddModule(prob,"");
stringstream sinput, sout;
// input the load command
sinput << "load data/lab1.env.xml";
if( !prob->SendCommand(sout,sinput) ) {
    RAVELOG_WARN("command failed!\n");
}
else {
    sinput.str(""); // have to reset the stream from the previous command
    sinput << "numbodies"; // input the numbodies command
    prob->SendCommand(sout,sinput);
    int numbodies;
    sout >> numbodies;
    RAVELOG_INFO("number of bodies are: %d\n",numbodies);
}
```

Python

```
prob = RaveCreateProblem(env,'MyProblem')
env.AddModule(prob,args='')
cmdout = prob.SendCommand('load data/lab1.env.xml')
if cmdout is None:
    raveLogWarn('command failed!')
else:
    cmdout = prob.SendCommand('numbodies')
    print 'number of bodies are: ',cmdout
```

Octave (only simple commands possible)

```
prob = orEnvCreateProblem('MyProblem');
orProblemSendCommand('load data/lab1.env.xml',prob);
numbodies = orProblemSendCommand('numbodies',prob);
disp(['number of bodies are: ' num2str(numbodies)])
```

3.5 Documenting Interfaces

The format of all interface documentation is the widely adopted standard `reStructuredText`. The description of an interface and all information about its usage should be provided by two places:

- OpenRAVE::InterfaceBase::GetDescription() - Returns the full documentation of the interface description. If opening new sections, do not to use '-'.
- OpenRAVE::InterfaceBase::RegisterCommand() - Help string in every command registered. If opening new sections, do not to use '-', '=', and '∼'.

These descriptions are automatically parsed using Sphinx and put on the web.

The reason why doxygen and other commenting tools are not adopted for plugin documentation is because the Base Interface Classes are the only binding between plugins. Even if the header file or provided functions of a particular plugin were provided, other plugins would not be able to use them if not offered through the OpenRAVE's channels.

3.6 Loading Plugins

Many mechanisms have been put in place to prevent mismatching/old plugins to be loaded by the core. Using interfaces from stale plugins can lead to unexpected crashes that are very difficult to debug. It is possible to automatically come up with a unique hash of the interface functions and members by running each interface through a C++ lexer and then creating a 128bit unique md5 hash. In order to protect plugins compiled with a different version, OpenRAVE creates a md5 hash from each interface class definition using cpp-gen-md5 and stores them in openrave/interfacehashesh.

The interface hash can be retrieved using OpenRAVE::RaveGetInterfaceHash. For an interface to be loaded successfully, the plugin has to check that the hash the core is using matches the hash compiled with the plugin. These types of checks ensure that stale plugins will never be loaded; helper functions are offered in openrave/pluginh, which all plugin authors should use.

4 C++ Examples

4.1 Building

All C++ examples are stored in `$OPENRAVE_INSTALL/share/openrave/cppexamples`. Use the `CMakeLists.txt` file to build the examples by:

```
cd `openrave-config --prefix`/share/openrave/cppexamples
mkdir build
cd build
cmake ..
make
```

4.1.1 OpenRAVE Installations

- FindOpenRAVE::cmake - include this in cmake projects for automatic detection of OpenRAVE.

4.2 Examples Plugins

- customreader::cpp
- plugincpp::cpp

4.3 Examples loading the OpenRAVE core:

- ikfastloader::cpp
- orcollision::cpp
- orconveyormovement::cpp
- orikfilter::cpp
- orloadviewer::cpp
- orshowsensors::cpp
- ormulticontrol::cpp
- orplanning_ik::cpp
- orplanning_planner::cpp
- orplanning_module::cpp
- orpythonbinding::cpp
- ortrajectory::cpp

4.4 Linking with Other Libraries

- opencvsaving::cpp

5 Executable Programs

- openrave_exe
- openrave-config

5.1 openrave-config

Used to find the OpenRAVE installation location and all the include/development/examples files.

Usage:

```
openrave-config
  [--prefix[=DIR]]
  [--exec-prefix[=DIR]]
  [--version] [--cflags]
  [--libs]
  [--libs-core]
  [--libs-only-l]
  [--libs-only-L]
  [--cflags-only-I]
  [--shared-libs]
  [--python-dir]
  [--octave-dir]
  [--matlab-dir]
  [--usage | --help]
```

6 Building and Installing

OpenRAVE uses CMake, which can create the correct build systems files for a variety of system configurations. Refer to the pages below for specific systems:

- Building and Installing on Linux
- Building and Installing on Windows
- Building and Installing on Mac OSX

6.1 CMake Options

The root Makefile creates a `build` directory and calls cmake with a set of options. All compiled object files are stored in `build`. There are many cmake options that control how OpenRAVE is built. Once OpenRAVE is built, most of them can be set by executing

```
ccmake build
```

in the root sources folder. It is also possible to call cmake from the command line with a new set of options:

```
cd build; cmake -DCMAKE_INSTALL_PREFIX=/my/new/install/dir -DCMAKE_BUILD_TYPE=Debug ..
```

- **Video Recording** To disable ffmpeg video recording, find the `OPT_VIDEORECORDING` option and turn it to `OFF`. It is possible to set these options during the cmake build time by executing cmake yourself

  ```
  cmake -DOPT_VIDEORECORDING=OFF ..
  ```

- **Double Precision** To compile OpenRAVE with **double precision** (ie all dReal types become **doubles**), set `OPT_DOUBLE_PRECISION` to `ON` during cmake. For example:

  ```
  cmake -DOPT_DOUBLE_PRECISION=ON ..
  ```

- **Static Libraries** To also build static libraries of openrave use the `OPT_STATIC` option. Plugins will still be linked dynamically.

  ```
  cmake -DOPT_STATIC=ON ..
  ```

- **Plugin Compilation** It is possible to only compile the core and skip the plugins using the `OPT_PLUGINS` option.

  ```
  cmake -DOPT_PLUGINS=OFF ..
  ```

6.2 Install Directory Structure

- bin
  - openrave - Start a simple environment and attach the default viewer, collision checkers, physics engines, and servers.
  - **`openrave.py`** - Start a simple environment through `openravepy`. Allows access to all openrave functionality including database generation and examples.
  - openrave-config - configuration file that helps users find the OpenRAVE installation.
  - openrave-hashpy - Query the body and robot hashes.
- include
  - openrave - Directory for all public OpenRAVE header files.
  - **openrave-core.h** - Header file for instantiating the OpenRAVE Core
- lib
  - **libopenrave** (.so, .a, .dll, .lib) - Library that all plugins should link to offering the Base Interface Classes.
  - **libopenrave-core** (.so, .a, .dll, .lib) - Library that allows any application to start the OpenRAVE core internally.
  - cmake
    - openraveX.Y
      - **openrave-config.cmake** - cmake file for searching for openrave installations
      - **openrave-config-version.cmake** - cmake file for searching for openrave installations
  - pythonX.Y
    - site-packages (dist-packages)
      - openravepy - Python bindings, database generators, and examples. Directory structure is explained `here`.
- share
  - openrave
    - cppexamples - All C++ examples
      - **CMakeLists.txt** - The `cmake` file used to compile all the examples.
    - data - Loadable OpenRAVE environment and object files.
    - matlab - All Matlab scripting functions and examples. Include this directory into the MATLAB path.

* models - 3D model resources
* octave - All Octave scripting functions and examples. Include this directory into the OCTAVE_PATH.
* plugins - Plugins compiled by the OpenRAVE main distribution.
* robots - Robot XML files usually containing just a `<robot>` or `<kinbody>` tag.

6.3 Building and Installing on Linux

Download the sources from sourceforge.

```
svn co https://openrave.svn.sourceforge.net/svnroot/openrave/tags/latest_stable openrave
```

OpenRAVE works only with qt4! Before compiling SoQt or OpenRAVE make sure to remove any qt3 related dev packages like `qt3-dev-tools` or `libqt3-headers`.

If there are compilation errors with templates, might have to use gcc version 4.1 or greater. You can check your version of gcc by `gcc --version`

6.3.1 Installing from Package Manager

6.3.1.1 Ubuntu

```
sudo add-apt-repository ppa:openrave/release
sudo sh -c 'echo "deb-src http://ppa.launchpad.net/openrave/release/ubuntu `lsb_release -cs` main" >
sudo apt-get update
sudo apt-get build-dep openrave
```

6.3.1.2 Fedora Core Users

Might need to add the livna yum repository for ffmpeg. To install the necessary packages type

```
sudo yum install qt4-devel Coin2-devel glew-devel gsm-devel x264 x264-devel ffmpeg-devel libdc1394-d
```

Be careful if installing **soqt-devel** from the package manager, it may be compiled with the wrong version of Qt. If compiling SoQt from sources, check `QTDIR` and make sure it points to the correct version.

6.3.2 Necessary libraries for Old Linux Distros

(Only install these libraries from sources if not in package manager.

- `boost`
- `Coin3d`
- `SoQt`

- **x86-64 users:** SoQt might give a compilation error in SoQtComponent.cpp. To fix it, go into `src/Inventor/Qt/SoQtComponent.cpp`:103 and replace `unsigned long key` with `SbDict::Key key`.

6.3.3 Collision Checkers and Physics Simulators

Although OpenRAVE is not tied to any particular collision checker, it requires at least one is installed to get basic functions. Here are the following libraries that have OpenRAVE plugins:

- `ODE Collision/Physics` - install from sources 0.8-0.10.1 tested to work. It is possible to compile ODE with double precision.
  - For 0.10.1, configuring with `--enable-new-trimesh` option will randomly crash openrave, so use at your own risk.
  - If not using enable-new-trimesh, disabling asserts (via `--disable-asserts`) is necessary due to some weird bug in normalization bug.
  - For 0.10+. If installing on x86-64 distro, configure ODE with `--enable-shared`.

- `Bullet Collision` - Need to install to a system directory so OpenRAVE can find it using `pkg-config`. If using autotools `configure`, need to remove `install-sh` before running `autogen.sh`. Bullet is detected using `pkg-config`, so `bullet.pc` is needed.

- `PQP Collision` - source code separate inside **pqprave** plugin.

6.3.4 Python

In order to use the python bindings through the `openravepy` module, you'll need to install python, boost python, python numpy, and python sympy. .

To setup the python path in bash add the following line:

```
export PYTHONPATH=$PYTHONPATH:`openrave-config --python-dir`
```

6.3.5 Octave/Matlab

Both Octave and Matlab are supported and the OpenRAVE build system automatically detects and compiles the mex files for each.

Octave users:

- Make sure `mkoctfile` is in your path. If installing octave from the package managers, also install the **octave headers** package.
- Add `$OPENRAVE_INSTALL/share/openrave/octave` to OCTAVE_PATH (default is `/usr/local/share/openrave/octave`).

Matlab users:

- make sure `mex` is in the system path
- Compile the *.cpp files inside `$OPENRAVE_INSTALL/share/openrave-x`.y/matlab and addit to the MATLAB Path.

6.3.6 Building OpenRAVE

After all necessary libraries have been installed, it is time to finally build and install OpenRAVE. In the root openrave folder type

```
make
```

This will compile all files. To install all the files type

```
make install
```

Because multiple openrave versions can co-exist in a system, to install all files except symlinks so that previous installations are not clobbered, do:

```
make altinstall
```

To choose a different install directory, type

```
make prefix=/my/new/dir
```

In case libraries are added after the initial install of OpenRAVE, remove the cmake cached files using:

```
rm build/CMakeCache.txt
```

If the cached files are not removed, the new system settings will not be noticed.

6.3.7 Bash Shell

Can enable automatic completion of several openrave programs like `openrave.py` by sourcing the `openravebash` file. Add this to your ∼./.bashrc file:

```
source `openrave-config --prefix`/share/openrave/openravebash
```

6.4 Building and Installing on Windows

NOTE: Only Windows XP has been tested. Windows 7 and others have been reported to work, but might need to modify the executable properties to **windows95 compatibility mode** after the executable has been built.

It is recommended to use OpenRAVE from the official `Windows Installers on Sourceforge`.

6.4.1 Installation of 3rd Party Software

Download the sources from `sourceforge` (`Tortoise SVN` is recommended). The subversion url is:

```
https://openrave.svn.sourceforge.net/svnroot/openrave/tags/latest_stable
```

Check out the openrave sources **in a path whose directories do not contain spaces**!!!. For example `C:\openrave`.

A lot of the 3rd party libraries are already inside the openrave sources. However, the following need to be installed:

- `Latest Boost C++ Libraries` - Select **Multithreaded DLL**!
  - A local boost installation is included in the sources in case cannot use a system install.
- `Qt4 MSVC Installer`
- `CMake`.
- For openravepy python bindings (**make sure Python version is compatible with Boost.Python! 2.6 is recommended!**):
  - `Python 2.6.6`
  - `numpy superpack`
  - **Highly Recommended:** `PyReadline` and `IPython`
  - **Recommended:** `SciPy` - the scientific library for Python is recommended.

- SymPy - Python Symbolic Mathematics.
- **openravepy** will not compile in Debug mode unless the Python debug libraries are also installed.

6.4.1.1 Installation of 3rd Party Software

Both Octave and Matlab are supported and the OpenRAVE build system automatically detects and compiles the mex files for each.

- If using Octave, make sure to add the path to `mkoctfile` to the $Path environment variable. If you are using Matlab, make sure that the path to `mex` is in the PATH environment variable (ie, typing `mex` on the command line gives the MATLAB compiler). On Windows Mik-Tex overwrites the `mex` program with its own version, so any matlab paths have to be declared before Mik-Tex in the PATH variable. You can check if the paths are set correctly by starting up the command prompt and typing `mkoctfile` or `mex`.

Octave users:

- Make sure `mkoctfile` is in your system path.
- If you having problems with the JVM, try installing octave using gnuplot (rather than jhandles).
- Compiling mex files should use the same compiler as was used to build the Octave package. As of October 2008, the official Octave binaries are compiled with Visual Studio 2005 (freely available).
- Add `$OPENRAVE_INSTALL\share\openrave\octave` to the Octave path (default is `C:\Program Files\openrave\share\openrave\octave`).

Matlab users:

- make sure `mex` is in your path and actually points to the Matlab `mex` program.
- Compile the *.cpp files inside `$OPENRAVE_INSTALL\share\openrave-*\matlab` and addit to the MATLAB Path. Can use runmex.bat for that.

6.4.2 Building OpenRAVE

Run the `CMake` GUI and specify a build directory somewhere out of the current source directory. Click on the Configure and Generate buttons. For Visual Studio users, this will generate a **OpenRAVE.sln** file. Open the Microsoft Visual Studio solution file and select the **RelWithDebInfo** or **Release** configuration, and build everything. Once done, build the INSTALL project. This should install everything in `C:\Program Files\openrave`. The installation directory can be changed by configuring `CMake's` CMAKE_INSTALL_PREFIX variable to a new path. If using the command-line, can specify the install directory with `-DCMAKE_INSTALL_PREFIX="my/new/install/dir"`.

Before running anything, have to modify the following environment variables:

- `PYTHONPATH` - add `"C:\\Program Files\\openrave\\share\\openrave"`
- `Path` - add `"C:\\Program Files\\openrave\\bin"` to the front. Furthermore, if Qt and Boost are installed in separate directories, have to add the location of their DLLs to Path (for example `C:\Qt\4.7.1\bin` and `C:\Program Files\boost\boost_1_44\lib`). Be very careful when putting it in the back since several user have had problems with different Qt libraries conflicting!

6.4.3 Updating Subversion

Whenever updating subversion, it should just be sufficient to run `runcmake_win.bat` again and then open the solution file build all the projects, and then manually build the INSTALL project. If an update to libraries or programs happens (for example Octave/MATLAB/python was installed/uninstalled), it is recommended to clear the cmake cache by first removing `build\CMakeCache.txt` before running `runcmake_win.bat`.

6.5 Building and Installing on Mac OSX

First execute this command:

```
sudo port install cmake boost libxml2 glew py27-numpy py27-pyplusplus
```

The py27-xx are for python 2.7 users. If a different version of python will be used, the names will have to change correspondingly. Note that OpenRAVE automatically chooses the newest version of python it can find, therefore it is necessary that the python bindings also correspond to this version.

Sometimes it is easiest to install boost python directly from sources. Once installed, make sure to set the `BOOST_ROOT` environment variable to point to it before compiling OpenRAVE.

It is also necessary to install these libraries:

- Qt - download the installer. Users have reported problems with using **port install qt4-mac** due to Cocoa. It is highly recommended to use the official qt4 installer package.
- Coin3D - build from sources
- SoQt - build from sources. Might ask you to set QTDIR to the correct directory.
- ffmpeg and other video recording codecs

After this, follow the instructions on Building and Installing on Linux.

7 Introduction to the OpenRAVE Architecture

7.1 Fundamental Structure

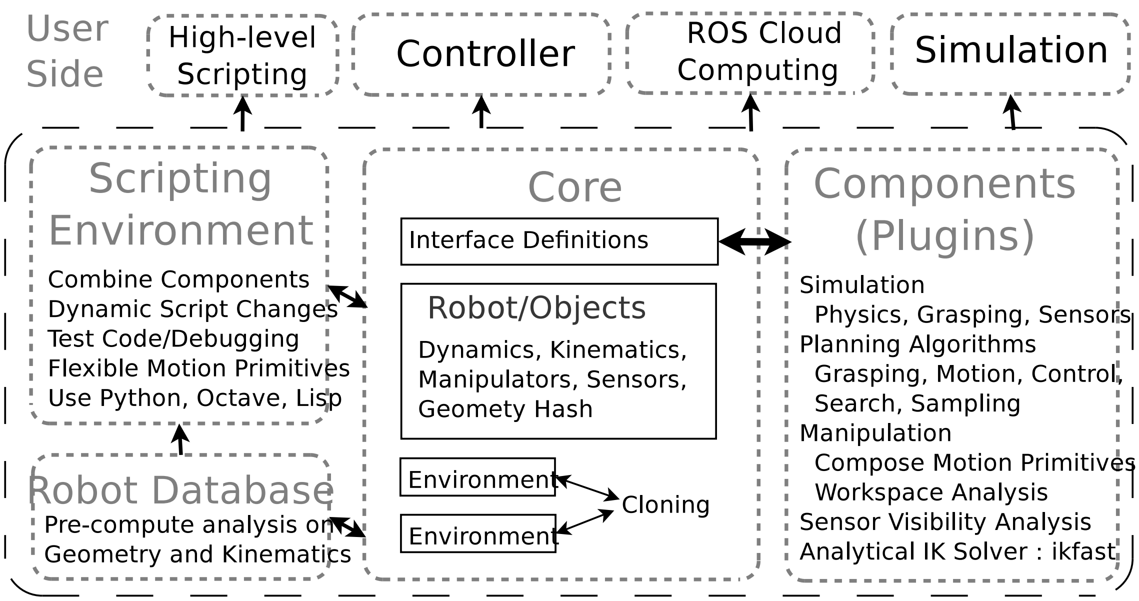


Figure 1: OpenRAVE architecture

OpenRAVE is divided in four main components as shown in the above figure:

- **Core Layer** The core is composed of a set of Base Interface Classes defining how plugins share information, and it provides an environment interface that maintains a world state, which serves as the gateway to all functions offered through OpenRAVE. The global openrave state manages the loaded plugins, multiple independent environments, and logging. On the other hand, the environment combines collision checkers, viewers, physics engines, the kinematic world, and all its interfaces into a coherent robotics world state.

- **Plugins Layer** OpenRAVE is designed as a plugin-based architecture where a plugin offers implementations of the Base Interface Classes that are loaded dynamically into the environment. Plugins can be linked with other robotics libraries allowing OpenRAVE to expand its functionality, or it can offer OpenRAVE services to another robotics system. During startup, OpenRAVE parses the {OPENRAVE_PLUGINS} environment variable and loads all the plugins it finds.

  - Refer to Writing Plugins and Interfaces for a tutorial on how to build and compile plugins.
  - Refer to Base Interface Concepts for interface details.

- **Scripting Layer** OpenRAVE provides scripting environments for `Python` and `Octave/Matlab`. Python communicates with the core layer directly with in-memory calls, making it extremely fast. On the other hand, the Octave/Matlab scripting protocol send commands through TCP/IP, with a plugin offering a text server on the OpenRAVE core side. Scripting allows real-time modifications to any aspect of the environment without requiring shutdown, making it ideal to testing new algorithms. The Python scripting is so powerful, that most of the OpenRAVE examples and demo code are offered through it. In fact, users should treat the scripting language as an integral part of the entire system, not as a replacement to the C++ API.

- **Robot Database Layer** Implements a planning knowledge-base and provides simple interfaces for its access and generation parameters. The database itself mostly consists of kinematic, quasi-static, dynamic, and geometric analyses of the robot and the task. If the robot is defined properly, then all these functions should work out of the box.

All the base planners and modules should be applicable to any robot structure that can be thrown at it. One of OpenRAVE's strongest points when compared with other planning packages is the idea of being able to apply algorithms in openrave to any robot, with very little modification. Recently, a planning database structure has been introduced that allows computation of properties like convex hull decomposition, grasp sets, reachability maps, analytic inverse kinematics, etc. If the robot is defined properly, then all these functions should work out of the box.

The main API is coded in C++ using the Boost C++ libraries [Dawes et al (1998-present)] as a really solid basis of low-level management and storage structures. The Boost flavors of shared pointers allow object pointers to be safely reference counted in a heavily multi-threaded environment. Shared pointers also allow handles and interfaces to be passed to the user without having to every worry about the user calling upon invalid objects or un- loaded shared objects. Furthermore, OpenRAVE uses functors and other abstracted objects commonly seen in higher level languages to specify function pointers for sampling distri- butions, event callbacks, setting robot configuration state, etc. The Boost-enabled design makes the the C++ API really safe and reliable to use along with saving the users a lot of trouble doing bookkeeping on their end. Furthermore, it allows the Resource Acquisition Is Initialization (RAII) design pattern [Stroustrup (2001)] to be fully exploited allowing users to ignore the complexities of multi-threaded resource management.

7.2 Environment Concepts

All of OpenRAVE's services are offered through the environment. For example, requesting a planner interface called 'BiRRT' is done through RaveCreatePlanner(). The environment supports:

- managing and communicating with plugins
- collision checking
- loading scenes and objects
- managing objects and triangulation
- drawing/plotting

Whenever objects in the environment are written or read, the user has to **lock** the environment mutex mutex GetMutex(). This prevents any other process from modifying the enviornment while the user is working. Because the environment uses 'recursive mutexes, it allows a mutex to be locked as many times as needed within the same thread. This has allowed all environment functions that require locking to always guarantee the mutex is locked, regardless if the user has locked the mutex. (Note that this only applies to environment functions, and not interface functions).

7.2.1 Locking

Because OpenRAVE is a highly multi-threaded environment, the environment state like bodies and loaded interfaces could be simultaneously accessed. In order to safely write or read the state, a user has to lock the environment, which prevents any other process from modifying the environment while the user is working. By using recursive locks, it allows a lock to be locked as many times as needed within the same thread, greatly reducing the lock management when a state changing function calls another state changing function. This safety measure helps users by always guaranteeing the environment is locked when calling global level environment functions like creating new bodies or loading scenes, regardless if the user has locked it. However, directly accessing the bodies and robots is dangerous without having the environment lock acquired.

7.2.2 Simulation Thread

Every environment has an internal time and a simulation thread directly attached to a physics engine. The thread is always running in the background and periodically steps the simulation time by a small delta for the physics engine and on all the simulation-enabled interfaces. By default, the thread is always running and can always potentially modify the environment state; therefore, users always need to explicitly lock the environment whenever playing with the internal state like modifying bodies by setting joint values or link transformations. If not careful, the controller or physics engine will overwrite them. By default, the simulation thread just sets the object positions depending on their controller inputs, but a physics engine can be attached to integrate velocities, accelerations, forces, and torques.

The simulation thread can feel like a nuisance at first, but its division of robot control into control input computation and execution greatly helps users only concentrate on feeding commands to the robot without worrying about the simulation loop. It also allows a world update to happen in one one discrete time step.

7.2.3 Cloning

One of the strengths of OpenRAVE is in allowing multiple environments work simultaneously in the same process. Environment cloning allows OpenRAVE to become truly parallel by managing multiple environments and running simultaneous planners on top of them.

One of the strengths of OpenRAVE is in allowing multiple environments to work simultaneously in the same process. Environment cloning allows OpenRAVE to become truly parallel by managing multiple environments and running simultaneous planners on top of them. Because there is no shared state across the clone and the original environment, it is not possible to use an interface created from one environment in another For example, if a planner is created in one environment, it should be used only by objects in that environment. It is not possible to set a planner to plan for objects belonging to a different environment. This is because a planner will lock the environment and expect the objects it controls to be exclusively under its control.

Creating a clone is simple, in C++ just type:

```
EnvironmentBasePtr pNewEnvironment = GetEnv()->CloneSelf(Clone_Bodies)
```

to create a clone that copies all the existing bodies (with attachments and grabbed bodies) and their current states. Basically the clone can perform any operations that would have been done with the original enviornment.

Because the environment state is very complex, the cloning process can control how much of it gets transferred to the new clone. For example, all existing bodies and robots can be cloned, their attached controllers can be cloned, the attached viewer can be cloned, the collision checker state can be cloned, and the simulation state can be cloned. Basically the clone should be able to perform any operations that can be done with the original environment without any modification in the input parameters.

When cloning real robots, one extremely important feature that OpenRAVE cloning offers is the ability to maintain a real-time view of the world that sensors continuously update with new information. When a planner is instantiated, it should make a copy of the environment that it can exclusively control without interfering with the updating operations. Furthermore, the real-world environment possibly has robot controllers connected to real robots, having a clone gives the ability to set simulation controllers guarantees robot safety while planning; commands from a cloned environment would not accidentally send commands to the real robot.

7.2.4 Validating Plugins

Every plugin needs to export several functions to notify the core what interfaces it has and to instantiate the interfaces. When a plugin is first loaded, it is validated by the environment and its interface information is queried so the core can register the names.

There are many mechanisms in the validation process to prevent old plugins to be loaded by the core. OpenRAVE is updated frequently and all user plugins are not necessarily recompiled when the OpenRAVE API changes. Therefore, we will encounter many cases when a plugin exports the correct functions, but does not implement the correct API. Using interfaces from plugins compiled with a mismatching The API can lead to unexpected crashes that are very difficult to debug, so it is absolutely necessary to detect this condition. One possible solution is to add version numbers to the API to enforce checking before an interface is returned from the plugin to the environment, but this method is brittle. It forces to keep track of a version number for every interface along with a global version number. Furthermore, every developer has to remember to increment the version when something even small changes, which can be easily forgotten and lead to serious errors later on.

We solve interface validation by computing a unique hash of the interface functions and members by running each interface through a C++ lexer, gathering the tokens that affect the C++ code structure, and then creating a 128bit unique MD5 hash. We create a hash for each interface definition and the environment. The hashes are hard coded into the C++ header files and can be queried by two methods: a static function returning the hash of the program calling the function, and a virtual function returning the hash the interface was compiled with. An interface is only valid if its virtual hash is equivalent to the static hash of the core environment. For a plugin to be loaded correctly, first the environment hashes have to match. If they do, then the individual interfaces checked and only matching interfaces are returned to the core, and from there dispatched to other plugins. Such consistency checks ensure that stale plugins will never be loaded.

7.2.5 Parallel Execution

Being able to execute a planner in multiple threads is important for applications that require speed and solution quality Because there is always a trade-off between solution quality and time of computation, some applications like industrial robots require the quickest and smoothest past to their destinations. Fortunately, environment cloning allows planners to create an independent environment for every thread they create, which enables them to call kinematics and collision functions in each respective thread without worrying about data corruption. Growing an RRT tree in a multi-threaded environment just requires one copy of the kd-tree structure to be maintained. The query operations mostly work with Euclidean distance on the configuration space, so are really fast. Furthermore, adding a new point takes O(log) time, so it shouldn't be a bottleneck in the search process compared to collision checking. Finally, environment locking allows threads to gain exclusive access to the environment. The rule of thumb is that any interface belonging or added to the environment requires an environment lock before any of its methods can be called.

7.3 Dual Simulation/Control Nature

OpenRAVE can be simultaneously used as a simulation, a controller, or both at the same time. Here are a couple of things to keep in mind:

- It can be used as a simulator by attaching a physics engine and setting torques to the joints and applying forces to the links.
- The physics engine directly reflects the internal openrave state.
- A controller can be set that sets torques/velocities/positions to the physics engine every time step. Physics simulation time steps are constantly called in an internal "openrave thread" if simulations are set to true (default)
- The default physics engine doesn't touch the openrave state, nor does it simulate velocities or dynamics
- The default controller just sets positions at the specified times

This is why users need to explicitly lock the environment mutex whenever playing with the internal openrave state like setting joint values or link transformations (in planners for example). Otherwise, the controller or physics engine will overwrite them.

7.4 Exception Handling

By using the C++ Standard and Boost libraries, OpenRAVE can recover from almost all errors that a user can experience without causing the program to shutdown on the spot. Invalid pointer and out-of-range accesses are extremely dangerous because they can modify unrelated memory, which causes the program to crash at a place completely unrelated to the root cause of the problem. Avoiding such problems has been one of the the highest priorities for the design. The core always surrounds any user code coming from plugins and callbacks with try/catch blocks, this allows the core to properly handle the error and notify the user of a problem without tearing down the environment.

Because exception handling is slow, there is a fine balance of when a function should return an error code and what it should throw an exception. In OpenRAVE, exceptions should never occur in normal operation of the program, they should only be for unexpected events of the program. For example, planners failing is an expected event dependent on the current environment, so planners should return an error code with the cause of the failure rather than throw an exception. In other words, exceptions convey the structural errors of the program that point to places in the code that should be fixed by the user. The following operations should throw exceptions in OpenRAVE:

- invalid plugin or interface hashes,
- invalid commands being sent to interfaces,
- invalid arguments passed to functions,
- invalid pointers or out-of-range parts of lists are accessed,
- environment is not locked when it should be
- a resource is present when it should be,
- a math operation is not consistent with the rest of the environment,
- environment naming constraints are not maintained,
- unrecognized enumerated types are given, and
- instantiation order is not maintained.

Any type of boost error, or null pointer access throws an openrave_exception. This greatly reduces the amount of error checking code people do. For example, C code usually has this pattern:

```
bool somefun(KinBodyPtr pbody)
{
    if( !pbody )
        return false;
    pbody->GetTransform();
    ...
}
```

or

```
bool somefun(KinBodyPtr pbody)
{
    assert( !!pbody );
    pbody->GetTransform();
    ...
}
```

If these checks are not done, the code would segfault. However, these checks can really clutter the code. In openrave, it is safe to get away with:

```
bool somefun(KinBodyPtr pbody)
{
    pbody->GetTransform();
    ...
}
```

then for handling errors (for example in the most top-level script), do

```
try {
    ...
    somefun(pbody)
    ...
}
catch(const openrave_exception& ex) {
    RAVELOG_WARN("exception caught: %s\n",ex.what());
    if( ex.GetCode() == ORE_EnvironmentNotLocked ) {
        RAVELOG_WARN("user forgot to lock environment!\n");
    }
    ...
}
```

When using openravepy in python, such unhandled C++ errors throw a python exception, which can be safely caught and processed there.

7.5 Hashes for Body Structure

A new concept that came out of OpenRAVE is the idea of creating unique hashes of a body's structure. Every body has an online state that includes:

- names of the body, its links, its joints,
- link transformations, velocities, and accelerations in the world,
- and attached bodies.

All other information is independent of the environment and can be categorized into the kinematics, geometry, and dynamics of the body. Furthermore, robots have categories for attached sensors and manipulators. The planning knowledge-base stores all cached information about a body and a robot, so it needs an consistent way of indexing this information. Indexing by robot names is not reliable because it is very difficult to remind a user to change the name every time the body structure is changed. Therefore, OpenRAVE provides functionality to serialize the different categories of a body and create a 128-bit MD5 hash. Each of the models in the planning knowledge-base relies on different categories of the robot. For example:

- inverse kinematics generation only uses the kinematics of a sub-chain of the robot defined by the manipulator and the grasp coordinate system,
- kinematic reachability cares about the robot geometry of the manipulator because it implicitly stores self-collision results,
- inverse reachability further uses the links connecting the base robot link to the base manipulator link,
- grasping cares about the geometry of the target body and the kinematics and geometry of the gripper,
- convex decompositions only care about the link geometry, and
- inverse dynamics cares only about the dynamics properties of each link and the kinematics.

There are several challenges to developing a consistent index across all operating systems and compilers since floating point errors could creep in when normalizing floating-point values. However, the idea of such an index could greatly help in developing a worldwide robot database that anyone can use.

7.6 Resource File Formats

OpenRAVE defines its own format that allows instantiation of any OpenRAVE interface and quick builing of robots and and kinematics structures. The rigid body geometries resources can be specified in virtually any 3D file format. For example:

- iv, vrml, wrl, stl, blend, 3ds, ase, obj, ply, dxf, lwo, lxo, ac, ms3d, x, mesh.xml, irrmesh, irr, nff, off, raw

These files can be used inside the <geom> tags, or can be read directly into any of the environment ReadRobotX and ReadKinBodyX methods to create a single world body.

OpenRAVE also supports the `COLLADA` international standard on 3D geometry and modeling. COLLADA is augmented with these `OpenRAVE robot-specific extensions`.

`More information here.`

8 Base Interface Concepts

New interfaces are provided by plugins and are dynamically loaded into OpenRAVE. All interfaces are derived from the OpenRAVE::InterfaceBase class and contain basic information such as the type, the owning environment, setting user data, cloning, and allowing custom string commands to be sent.

Every instantiated interface belongs to only one environment. Interfaces can be cloned using OpenRAVE::InterfaceBase::Clone.

Every interface can have its own custom commands. Sending **help** will return a list of all the commands the interface supports (think of it as a command-line way of sending commands to the interface). The GetDescription() returns a string briefly explaining the functionality, the authors, and the license of the plugin.

Ability to register custom xml reader interfaces.

- Collision Checker Concepts
- Controller Concepts
- Inverse Kinematics Solver Concepts
- Kinematics Body Concepts
- Physics Engine Concepts
- Planner Concepts
- Module Concepts
- Robot Concepts
- Sensor Concepts
- Sensor System Concepts
- SpaceSampler Concepts
- `Trajectory Concepts`
- Viewer Concepts

8.1 Collision Checker Concepts

Reference: OpenRAVE::CollisionCheckerBase.

All **CheckCollision** functions accept an optional pointer to a OpenRAVE::CollisionReport struct, which gets filled with information about the collision that takes place. Usually requesting more precise information like distance to obstacles is computationally expensive; therefore to save computation, the user can specify what collision information should be filled in the OpenRAVE::CollisionReport with the SetCollisionOptions function.

OpenRAVE is not tied to a particular collision checker. Collision checkers can be changed with SetCollisionChecker. In order to add a new collision checker, derive a class from OpenRAVE::CollisionCheckerBase and fill all the methods it provides. Then register it in 'src/environment.cpp' and CollisionCheckers. All collision checking is done through the overloaded EnvironmentBase::CheckCollision.

8.2 Controller Concepts

Reference: OpenRAVE::ControllerBase

In order for openrave to control certain robot hardware, a OpenRAVE::ControllerBase controller has to be created that will interface with the hardware-specific libraries. This controller interface then has to be created through the environment and set onto an existing robot. All commands given to the robot are first filtered through the controller, then translated to joint commands. Different controllers can have different path inputs (ie: a robot walking on a floor might just have x,y,angle), but the default is the DOF joint values.

8.2.1 Writing and Using Controllers

Assuming that there exists a plugin with a controller interface named **MyController**, here are some ways to set an openrave robot to use it:

- XML
  - add a <controller> tag in the openrave robot XML file like this:

    ```
    <robot file="robots/schunk-lwa3.robot.xml">
      <controller type="MyController controller arguments here"></controller>
    </robot>
    ```

  - It is also possible to set a controller outside of the robot definition by specifying the robot's name. For example:

    ```
    <environment>
      <robot name="schunk-lwa3" file="robots/schunk-lwa3.robot.xml">
      </robot>
      <controller type="MyController" robot="schunk-lwa3 controller arguments here"></controll
      </environment>
    ```

- C++

  ```
  RobotBasePtr probot = GetEnv()->GetRobot("schunk-lwa3");
  ControllerBasePtr pcontroller = RaveCreateController(GetEnv(),"MyController contr
        oller arguments here");
  vector<int> dofindices(probot->GetDOF());
  ```

```
for(int i = 0; i < probot->GetDOF(); ++i) {
    dofindices[i] = i;
}
int nControlTransformation = 1;
probot->SetController(pcontroller,dofindices,nControlTransformation);
```

- Python

```
robot = env.GetRobot('schunk-lwa3')
controller = RaveCreateController(env,'MyController controller arguments here')
robot.SetController(controller,range(robot.GetDOF()),controltransform=1)
```

- Octave/MATLAB

```
robotid = orEnvGetBody('schunk-lwa3');
orRobotControllerSet(robotid,'MyController','controller arguments here')
```

8.3 Inverse Kinematics Solver Concepts

Reference: OpenRAVE::IkSolverBase

Each IK solver is defined on a subset of joints of a Robot specified by the robot's manipulator. Given the position in the 3D workspace that an end effector should go to, an IK solver will find the joint configuration to take that end-effector there. Because it is common for an IK solution to have a null space, the IK solver give functionality to expose the free parameters to move the joints in null space.

8.4 Kinematics Body Concepts

Reference: OpenRAVE::KinBody

Each KinBody can be thought of as the basic rigid body element in OpenRAVE. It is composed of a collection of links (rigid bodies) connected with joints. The KinBody class provides a lot of functionality (from a planning perspective) needed to perform complex tasks:

- Setting and getting joint values
- Setting and getting the transformations of all links
- Getting the velocities of each joint (or link)
- Self collision detection functions
- Kinematic hierarchy querying - The underlying structure of KinBody is a list of links, not a tree. However, after some careful analysis, the parent and child links of a particular link can be extracted.
- Jacobian calculation - both translational and rotational.

- Attaching bodies online - a necessary function for manipulation planning; is called, for example, when an object is rigidly grasped by a hand

8.4.1 Loading Options

This is the set of loading options passed as a AttributesList into functions like OpenRAVE::EnvironmentBase::ReadKinBodyXMLFile:

- **prefix**: prefix link, joint, manipulator, and sensor names with a string
- **skipgeometry**: if 1 or true, will skip loading all geometry of the links

8.5 Physics Engine Concepts

Reference: OpenRAVE::PhysicsEngineBase

The physics engine for the environment can be set through OpenRAVE::EnvironmentBase::SetPhysicsEngine.

8.6 Planner Concepts

Reference: OpenRAVE::PlannerBase

8.6.1 Introduction

In OpenRAVE, the basic purpose of a planner is to find a trajectory starting at some initial configuration that reaches a goal condition while satisfying various navigation constraints. All planners are assumed to be geometric in nature (ie, not planning in the space of policies that depend on sensor data). Planners can have any configuration space defined by using the OpenRAVE::PlannerBase::PlannerParameters structure. A planner should never use the raw joint values functions defined in KinBody.

The usage of a planner is simple:

- Acquire its pointer from RaveCreatePlanner.
- Fill a PlannerParameters structure defining the instance of the problem. The structure has many fields for describing planning entities like start position, goal condition, and the distance metric. Try to use these fields as much as possible. Later on, this will allow users to easily swap planners without having to change the PlannerBase::PlannerParameters structure much.
- Call InitPlan passing in the robot and planner parameters. This also resets any previous information the planner had stored.

- Call PlanPath passing in a trajectory (and optionally an output stream) to start planning. If the function returns true, then the Trajectory will be filled with the geometric solution in the active DOF configuration space of the robot. By calling SetParameters, then PlanPath again, it could be possible to preserve the previous search space for the planner while changing the goal conditions.

8.6.2 Planning Details

8.6.2.1 Planner Parameters - Calling a Planner

All the information defining a planning problem should be specified in PlannerBase::PlannerParameters. `PlannerParameters` tries to cover most of the common data like distance metrics, sampling distributions, initial and goal configurations. However there are many different types of inputs to a planner, so it is impossible to cover everything with one class. Instead, `PlannerParameters` has a very flexible and safe way to extend its parameters without destroying compatibility with a particular planner or user of the planner. This is enabled by the serialization to XML capabilities of `PlannerParameters`

```
PlannerBase::PlannerParametersPtr params(new PlannerBase::PlannerParameters());
params->vinitialconfig.push_back(2);
ostream os;
os << *params;
```

will produce something in the form of

```
<PlannerParameters>
  <initialconfig>2</initialconfig>
</PlannerParameters>
```

Furthermore PlannerParameters can read such an XML file given an input stream

```
istream is;
is >> *params;
```

Using XML as a medium, it is easy to exchange data across different derivations of PlannerParameters without much effort. To add new parameters for planners to take advantage of

- make a derived class from PlannerParameters
- overload the PlannerParameters, startElement, endElement, and characters functions to process the new variables.

As long as the user of the planner passes a `PlannerParameters` that can serialize to the same format of data that the planner expects, the data will be passed. This allows the planner and the caller of PlanPath to use different `PlannerParameters`. definitions without any conflicts.

8.6.2.2 Basic Usage

This is a simple call to a birrt planner, let **activegoal** hold the goal configuration and **activejoints** hold indices to the robot joints interested to plan for.

```
PlannerBase::PlannerParametersPtr params(new PlannerBase::PlannerParameters);
params->SetRobotActiveJoints(robot); // sets the active joint indices
robot->GetActiveDOFValues(params.vinitialconfig); // set initial config (use curr
      ent robot configuration)
params.vgoalconfig = activegoal;

// set other params values like

PlannerBasePtr rrtplanner = RaveCreatePlanner(GetEnv(),"rBiRRT");
TrajectoryBasePtr ptraj = RaveCreateTrajectory(GetEnv(),robot->GetActiveDOF());
if( !rrtplanner->InitPlan(robot, params) ) {
    return false;
}

PlannerStatus status = rrtplanner->Plan(ptraj);
if( status & PS_HasSolution ) {
    robot->SetActiveMotion(ptraj); // trajectory is done, execute on the robot
}
```

In order to speed up computations further, planners can use the CO_ActiveDOFs collision checker option, which only focuses collision on the currently moving links in the robot. If using the robot active DOF, before calling the planner, the user should insert this statement:

```
CollisionOptionsStateSaver optionstate(GetEnv()->GetCollisionChecker(),GetEnv()->
      GetCollisionChecker()->GetCollisionOptions()|CO_ActiveDOFs,false);
```

8.6.2.3 Defining Extra Planner Parameters

Here is how to derive from a PlannerParameters class in order to introduce new parameters.

```
class BasicRRTParameters : public PlannerBase::PlannerParameters
{
public:
BasicRRTParameters() : _fGoalBiasProb(0.05f), _bProcessing(false) {
        _vXMLParameters.push_back("goalbias");
    }

    dReal _fGoalBiasProb;

 protected:
    bool _bProcessing;
    virtual bool serialize(std::ostream& O) const
    {
        if( !PlannerParameters::serialize(O) )
```

```
            return false;
        O << "<goalbias>" << _fGoalBiasProb << "</goalbias>" << endl;
        return !!O;
    }

    ProcessElement startElement(const std::string& name, const std::list<std::pai
      r<std::string,std::string> >& atts)
    {
        if( _bProcessing )
            return PE_Ignore;
        switch( PlannerBase::PlannerParameters::startElement(name,atts) ) {
            case PE_Pass: break;
            case PE_Support: return PE_Support;
            case PE_Ignore: return PE_Ignore;
        }

        _bProcessing = name=="goalbias";
        return _bProcessing ? PE_Support : PE_Pass;
    }

    virtual bool endElement(const string& name)
    {
        if( _bProcessing ) {
            if( name == "goalbias")
                _ss >> _fGoalBiasProb;
            else
                RAVELOG_WARN(str(boost::format("unknown tag %s\n")%name));
            _bProcessing = false;
            return false;
        }

        // give a chance for the default parameters to get processed
        return PlannerParameters::endElement(name);
    }
};
```

8.6.2.4 Planner Development

Most planners do their computation iteratively, and they take lots of computation time. It is very frequent for a user to want to early-terminate the planner, or tell it to return the best solution it has founds immediately. Users might also want to visualize the planning process without getting into the internals of the planner. In order to do this, OpenRAVE allows users to register callbacks via OpenRAVE::PlannerBase::RegisterPlanCallback. Planner developers should **always** call OpenRAVE::PlannerBase::_CallCallbacks inside their planning loop and process the input correctly.

8.6.3 Planner Examples

Examples of planners are:

- Manipulation - manipulable objects need to be specified. Objects like doors should be special cases that planners knows about.

- Following - Goal easily changes. Attributes can change.
- Path Smoothing - uses the input trajectory
- Trajectory Re-timing - uses the input trajectory
- Object Building - Need to describe how parts of object fit together into a bigger part.
- Dish Washing - Specific goals are not specified, just a condition that all plates need to be inside.
- Foot step planning - Need discrete footsteps and other capabilities from robot.

Planner should be able to query sensor information from the Robot like its current camera image etc. Planner should be compatible with Robot presented; some handshaking should happen between the two during InitPlan function.

8.6.4 Path Optimization

Path smoothing/optimization can be regarded as a post-processing step to planners. "Path optimization" algorithms take in an existing trajectory and filter it using the existing constraints of the planner. In fact, functionality there is no difference between a "path optimization" planner and a regular planner besides the fact that a trajectory is used as input. Because PlannerBase::PlanPath already has a trajectory as an argument, this does not cause any major API changes to the infrastructure.

However, the PlannerParameters structure had to reflect what 'path optimization' algorithm to use for post processing the trajectory. This is now reflected in the PlannerParameters::_sPostProcessingPlanner and PlannerParameters::_sPostProcessingParameters arguments. By default, this is the default "linear shortcut" path optimizer. There is also a helper function in PlannerBase to help users easily call the post-processing step:

```
_ProcessPostPlanners(RobotBasePtr probot, TrajectoryBasePtr ptraj);
```

Please take a look at how the default RRT algorithms are now structured.

Planner post-processing actually allows users to chain planners in the same way that filters are chained, all through specifying planner parameters. Of course, users can continue to smooth in planners without relying on this framework. However, explicit control of path smoothing allows custom parameter to be easily specified.

8.7 Module Concepts

Reference: OpenRAVE::ModuleBase

Base class for modules the user might want to instantiate. A module registers itself with OpenRAVE's SimulateStep calls and can accept commands from the server or other plugins via SendCommand. A module stops receiving commands when it is destroyed. Modules are an easy way for developers to run and test their own code.

8.8 Robot Concepts

Reference: OpenRAVE::RobotBase

Robots are a special type of KinBody that need higher level functionality for their control and movement in the environment. There are a couple of differences between a Robot and a regular KinBody.

8.8.1 Manipulators

Every robot supports a list of Manipulator objects that describe the links the robot should use when manipulating parts of the environment. Usually manipulators are serial chains with a Base link and an End Effector link. Each manipulator is also decomposed into two parts: the arm and the hand. The hand usually makes contact with the objects while the arm transfers the hand to its destination. The Manipulator class also has an optional pointer to a IkSolverBase class providing inverse kinematics functionality. The IK solver used by a Manipulator can be changed by Manipulator::SetIKSolver, so plugins can provide and set their own IK solvers.

8.8.2 Active Degrees of Freedom

When controlling and planning for a robot, it is possible to set the degrees of freedom that should be used. For example, consider a humanoid robot. There should be in easy way to specify to planners that only the right hand of the robot should be taken into consideration when planning; the rest of the joints should be ignored. Or consider the case where we care about navigation of the humanoid robot. Here we would want to control the translation of the robot on the plane and its orientation. Perhaps we want to do footstep planning and also care about controlling the two legs. All this is possible with the Active Degrees of Freedom feature provided by OpenRAVE. First call RobotBase::SetActiveDOFs to set the degrees of freedom of the robot; it is also possible to set translation about the XYZ axes or the angle around a rotation axis as a degree of freedom. Each RobotBase function with the word Active expects the active DOF values to be specified. Basically, for any function in KinBody that deals with Joints, there is a corresponding active function in RobotBase.

8.8.3 Grabbing Bodies

It is possible for a robot to attach a KinBody onto one of its links so that when the link moves, the KinBody also moves. Because collision detection will stop being checked between the robot and the KinBody, you could say that the KinBody becomes a part of the robot temporarily. This functionality is necessary for manipulation planning. Whenever the robot is carrying a body, all collisions between the robot and that item should be ignored once the body has been grasped.

8.8.4 Attaching Sensors

Can attach any number of sensors to the robot's links through the AttachedSensor class. The sensor transformation will be completely owned by the robot. A robot can be attached with any number of sensors on any number of links. As the robot link moves, the sensor moves with it preserving its relative transformation.

AttachedSensor object holds a SensorBase object that contains the actual object gathering and publishing data.

8.8.5 Loading Options

This is the set of loading options passed as a AttributesList into functions like OpenRAVE::EnvironmentBase::ReadRobotXMLFile.

KinBody Loading Options is also valid.

8.9 Sensor Concepts

Reference: OpenRAVE::SensorBase

A sensor measures physical properties from the environment and converts them to data. Each sensor is associated with a particular position in space, has a geometry with properties defining the type of sensor, and can be queried for sensor data. Available sensor types are specified by SensorType.

By default, all the sensors start with power off, meaning that the sensor does not gather data. The power can be turned on by using OpenRAVE::SensorBase::Configure and sending the SensorBase::CC_PowerOn command. All programs should manually turn sensor power on before using the sensors.

The sensor has two different rendering options:.

- **Geometry Rendering** - Renders a small icon that represents where the laser is placed in the environment, the icon's image should only be dependent on the geometry parameters of the sensor, and not the actual sensor data. Geometry rendering should be turned on by default. To configure this, use the SensorBase::CC_RenderDataX commands.

- **Data Rendering** - Data rendering shows the measured sensor data coming out of the GetSensorData(). Usually the data can be very heavy, especially when a sensor's update rate is high, so data rendering is turned off by default. If the power is off, data should not be rendered. To configure this, use the SensorBase::CC_-RenderGeometryX commands.

Check out the `basesensors` plugin for an example of how to implement a basic laser range and camera sensors.

8.10 Sensor System Concepts

Reference: OpenRAVE::SensorSystemBase New objects can be created, existing objects can be updated. Every managed object should set the kinbody's Manager pointer

8.11 SpaceSampler Concepts

Reference: OpenRAVE::SpaceSamplerBase

Space samplers are responsible for generating samples in spaces like R^n, SO(3), SE(3), etc. The samples can be randomized or deterministic.

Each sampler can support returning the values in a floating-point precision or unsigned integer format. There are sampling calls for each version. The samplers could choose to implement only one of the types of both, this should be clear in the Supports function.

Each sampler has a state and can be configured with different dimensions and seeds.

8.12 Viewer Concepts

Reference: OpenRAVE::ViewerBase Viewer is responsible only for the environment it is attached to.

9 Module Documentation

9.1 Plugin Export Functions

Classes

- class PLUGININFO

  Holds all the OpenRAVE-specific information provided by a plugin.

Typedefs

- typedef InterfaceBasePtr(∗ PluginExportFn_OpenRAVECreateInterface)(InterfaceType type, const std::string &name, const char ∗pluginhash, const char ∗envhash, EnvironmentBasePtr penv)

  Create the interfaces, see CreateInterfaceValidated.

- typedef bool(∗ PluginExportFn_OpenRAVEGetPluginAttributes)(PLUGININFO ∗pinfo, int size, const char ∗infohash)

  Called to fill information about the plugin, see GetPluginAttributesValidated.

- typedef void(∗ PluginExportFn_DestroyPlugin)()

  Called before plugin is unloaded from openrave. See DestroyPlugin.

Functions

- OpenRAVE::InterfaceBasePtr CreateInterfaceValidated (OpenRAVE::InterfaceType type, const std::string &name, std::istream &sinput, OpenRAVE::EnvironmentBasePtr penv)

  [helper] *Validated function callback for creating an interface function. No checks need to be made on the parmaeters.*

- void GetPluginAttributesValidated (OpenRAVE::PLUGININFO &info)

  [helper] *Validated function callback for returning a plugin's information. No checks need to be made on the parmaeters.*

- OPENRAVE_PLUGIN_API OpenRAVE::InterfaceBasePtr OpenRAVECreateInterface (OpenRAVE::InterfaceType type, const std::string &name, const char ∗interfacehash, const char ∗envhash, OpenRAVE::EnvironmentBasePtr penv)

  [export] *Definition of a plugin export. Requires CreateInterfaceValidated to be defined.*

- OPENRAVE_PLUGIN_API void OpenRAVEGetPluginAttributes (OpenRAVE::PLUGININFO ∗pinfo, int size, const char ∗infohash)

  [export] *Definition of a plugin export. Requires GetPluginAttributesValidated to be defined.*

- OPENRAVE_PLUGIN_API void DestroyPlugin ()

  [export] *Stub function to be defined by plugin that includes rave/pluginh.*

9.1.1 Detailed Description

Every plugin needs to export these functions

9.1.2 Function Documentation

9.1.2.1 OpenRAVE::InterfaceBasePtr CreateInterfaceValidated (OpenRAVE::InterfaceType *type*, const std::string & *name*, std::istream & *sinput*, OpenRAVE::EnvironmentBasePtr *penv*)

[helper] Validated function callback for creating an interface function. No checks need to be made on the parmaeters.

If possible, always returns a valid pointer regardless of initialization failure since the actual interface pointer stores documentation information and is used in introspection. Only use when rave/pluginh is included.

Parameters

← ***type*** the interface type

← ***name*** the lowercase letters of the interface name

← ***sinput*** a stream to the rest of the input args to OpenRAVECreateInterface

← ***penv*** the environment pointer

Returns

a pointer to the interface if one could have been created.

Examples:

customreader.cpp, and plugincpp.cpp.

9.1.2.2 void GetPluginAttributesValidated (OpenRAVE::PLUGININFO & *info*)

[helper] Validated function callback for returning a plugin's information. No checks need to be made on the parmaeters.

This function is called only once initially to determine what the plugin offers. It should be the safest funcdtion and should not create any static resources for the plugin. Only use when rave/pluginh is included.

Parameters

→ ***info*** Holds information on what services this plugin provides.

9.2 Base Interface Classes

Classes

- class CollisionCheckerBase

  [interface] *Responsible for all collision checking queries of the environment.* ***If not specified, method is not multi-thread safe.*** *See Collision Checker Concepts.*

- class ControllerBase

  [interface] *Abstract base class to encapsulate a local controller.* ***If not specified, method is not multi-thread safe.*** *See Controller Concepts.*

- class IkSolverBase

  [interface] *Base class for all Inverse Kinematic solvers.* ***If not specified, method is not multi-thread safe.*** *See Inverse Kinematics Solver Concepts.*

- class InterfaceBase

  [interface] *Base class for all interfaces that OpenRAVE provides. See Base Interface Concepts.*

- class KinBody

  [interface] *A kinematic body of links and joints.* ***If not specified, method is not multi-thread safe.*** *See Kinematics Body Concepts.*

- class ModuleBase

  [interface] *A loadable module of user code meant to solve a specific domain.* ***If not specified, method is not multi-thread safe.*** *See Module Concepts.*

- class PhysicsEngineBase

  [interface] *The physics engine interfaces supporting simulations and dynamics. See Physics Engine Concepts.*

- class PlannerBase

  [interface] *Planner interface that generates trajectories for target objects to follow through the environment.* ***If not specified, method is not multi-thread safe.*** *See Planner Concepts.*

- class RobotBase

  [interface] *A robot is a kinematic body that has attached manipulators, sensors, and controllers.* ***If not specified, method is not multi-thread safe.*** *See Robot Concepts.*

- class SensorBase

  [interface] *A sensor measures physical properties from the environment.* ***If not specified, method is not multi-thread safe.*** *See Sensor Concepts.*

- class SensorSystemBase

  [interface] *Used to manage the creation and destruction of bodies. See Sensor System Concepts.*

- class SpaceSamplerBase

  [interface] *Contains space samplers commonly used in planners.* ***If not specified, method is not multi-thread safe.*** *See SpaceSampler Concepts.*

- class TrajectoryBase

  [interface] *Encapsulate a time-parameterized trajectories of robot configurations.* ***If not specified, method is not multi-thread safe.*** `Trajectory Concepts`

- class ViewerBase

  [interface] *Base class for the graphics and gui engine that renders the environment and provides visual sensor information.* ***If not specified, method is not multi-thread safe.*** *See Viewer Concepts.*

9.2.1 Detailed Description

A list of the OpenRAVE interface templates. See Base Interface Concepts.

9.3 Geometric Primitives

Classes

- class ray< T >

  A ray defined by an origin and a direction.

- class aabb< T >

  An axis aligned bounding box.

- class obb< T >

  An oriented bounding box.

- class triangle< T >

  A triangle defined by 3 points.

- class frustum< T >

  A pyramid with its vertex clipped.

Functions

- template<typename T >
  int insideQuadrilateral (const RaveVector< T > &v, const RaveVector< T > &verts)

  Tests a point inside a 3D quadrilateral.

- template<typename T >
  int insideTriangle (const RaveVector< T > v, const triangle< T > &tri)

  Tests a point insdie a 3D triangle.

- template<typename T >
  bool RayAABBTest (const ray< T > &r, const aabb< T > &ab)

  Test collision of a ray with an axis aligned bounding box.

- template<typename T >
  bool RayOBBTest (const ray< T > &r, const obb< T > &o)

  Test collision of a ray and an oriented bounding box.

- template<typename T >
  bool IsOBBinFrustum (const obb< T > &o, const frustum< T > &fr)

  Test collision of an oriented bounding box and a frustum.

- template<typename T , typename U >
  bool IsOBBinConvexHull (const obb< T > &o, const U &vplanes)

  Tests if an oriented bounding box is inside a 3D convex hull.

- template<typename T >
  bool TriTriCollision (const RaveVector< T > &u1, const RaveVector< T > &u2, const RaveVector< T > &u3, const RaveVector< T > &v1, const

RaveVector< T > &v2, const RaveVector< T > &v3, RaveVector< T > &contactpos, RaveVector< T > &contactnorm)

Test collision if two 3D triangles.
Assuming triangle vertices are declared counter-clockwise!!

- template<typename T >
  obb< T > OBBFromAABB (const aabb< T > &ab, const RaveTransformMatrix< T > &t)

  Transform an axis aligned bounding box to an oriented bounding box.

- template<typename T >
  obb< T > OBBFromAABB (const aabb< T > &ab, const RaveTransform< T > &t)

  Transform an axis aligned bounding box to an oriented bounding box.

- template<typename T >
  obb< T > TransformOBB (const RaveTransform< T > &t, const obb< T > &o)

  Transforms an oriented bounding box.

- template<typename T >
  obb< T > TransformOBB (const RaveTransformMatrix< T > &t, const obb< T > &o)

  Transforms an oriented bounding box.

- template<typename T >
  bool AABBCollision (const aabb< T > &ab1, const aabb< T > &ab2)

  projects an obb along the world axes

Distnace functions.

- template<typename T >
  T DistVertexOBBSq (const RaveVector< T > &v, const obb< T > &o)

  The minimum distance form the vertex to the oriented bounding box.

9.3.1 Detailed Description

A set of geometric primitives and functions offering collision detection and other distance measurement capabilities.

9.3.2 Function Documentation

9.3.2.1 bool OpenRAVE::geometry::AABBCollision (const aabb< T > & *ab1*, const aabb< T > & *ab2*)

projects an obb along the world axes

Test collision between two axis-aligned bounding boxes.

9.3.2.2 bool OpenRAVE::geometry::IsOBBinConvexHull (const obb< T > & *o*, const U & *vplanes*)

Tests if an oriented bounding box is inside a 3D convex hull.

Parameters

vplanes the plane normals of the convex hull, normals should be facing inside.

9.3.2.3 obb<T> OpenRAVE::geometry::OBBFromAABB (const aabb< T > & *ab*, const RaveTransform< T > & *t*)

Transform an axis aligned bounding box to an oriented bounding box.

Parameters

← ***t*** transformation used to set the coordinate system of ab.

9.3.2.4 obb<T> OpenRAVE::geometry::OBBFromAABB (const aabb< T > & *ab*, const RaveTransformMatrix< T > & *t*)

Transform an axis aligned bounding box to an oriented bounding box.

Parameters

← ***t*** transformation used to set the coordinate system of ab.

9.3.2.5 obb<T> OpenRAVE::geometry::TransformOBB (const RaveTransformMatrix< T > & *t*, const obb< T > & *o*)

Transforms an oriented bounding box.

Parameters

← ***t*** transformation used to set the coordinate system of o.

9.3.2.6 obb<T> OpenRAVE::geometry::TransformOBB (const RaveTransform< T > & *t*, const obb< T > & *o*)

Transforms an oriented bounding box.

Parameters

← ***t*** transformation used to set the coordinate system of o.

9.3.2.7 bool OpenRAVE::geometry::TriTriCollision (const RaveVector< T > & *u1*, const RaveVector< T > & *u2*, const RaveVector< T > & *u3*, const RaveVector< T > & *v1*, const RaveVector< T > & *v2*, const RaveVector< T > & *v3*, RaveVector< T > & *contactpos*, RaveVector< T > & *contactnorm*)

Test collision if two 3D triangles.

Assuming triangle vertices are declared counter-clockwise!!

Parameters

→ ***contactnorm*** if triangles collide, then filled with the normal of the second triangle

Returns

true if triangles collide.

9.4 Affine Math

Classes

- class RaveVector< T >

Vector class containing 4 dimensions.

- class RaveTransform< T >

  Affine transformation parameterized with quaterions.

- class RaveTransformMatrix< T >

  Affine transformation parameterized with rotation matrices. Scales and shears are not supported.

Functions

- template<typename T >
  RaveVector< T > quatFromAxisAngle (const RaveVector< T > &axis, T angle)

  Converts an axis-angle rotation into a quaternion.

- template<typename T >
  RaveVector< T > quatFromAxisAngle (const RaveVector< T > &axisangle)

  Converts an axis-angle rotation into a quaternion.

- template<typename T >
  RaveVector< T > quatFromMatrix (const RaveTransformMatrix< T > &rotation)

  Converts the rotation of a matrix into a quaternion.

- template<typename T >
  RaveTransformMatrix< T > matrixFromQuat (const RaveVector< T > &quat)

  Converts a quaternion to a 3x3 matrix.

- template<typename T >
  void matrixFromQuat (RaveTransformMatrix< T > &rotation, const RaveVector< T > &quat)

  Converts a quaternion to a 3x3 matrix.

- template<typename T >
  RaveTransformMatrix< T > matrixFromAxisAngle (const RaveVector< T > &axis, T angle)

  Converts an axis-angle rotation to a 3x3 matrix.

- template<typename T >
  RaveTransformMatrix< T > matrixFromAxisAngle (const RaveVector< T > &axisangle)

Converts an axis-angle rotation to a 3x3 matrix.

- template<typename T >
RaveVector< T > quatMultiply (const RaveVector< T > &quat0, const RaveVector< T > &quat1)

  Multiply two quaternions.

- template<typename T >
RaveVector< T > quatInverse (const RaveVector< T > &quat)

  Inverted a quaternion rotation.

- template<typename T >
RaveVector< T > quatSlerp (const RaveVector< T > &quat0, const RaveVector< T > &quat1, T t)

  Sphereical linear interpolation between two quaternions.

- template<typename T >
RaveVector< T > quatRotate (const RaveVector< T > &q, const RaveVector< T > &t)

  transform a vector by a quaternion

- template<typename T >
RaveVector< T > quatRotateDirection (const RaveVector< T > &sourcedir, const RaveVector< T > &targetdir)

  Return the minimal quaternion that orients sourcedir to targetdir.

- template<typename T >
std::pair< T, RaveVector< T > > normalizeAxisRotation (const RaveVector< T > &axis, const RaveVector< T > &quat)

  Find the rotation theta around axis such that rot(axis,theta) $*$ q is closest to the identity rotation.

- template<typename T >
RaveVector< T > axisAngleFromQuat (const RaveVector< T > &quat)

  Converts a quaternion into the axis-angle representation.

- template<typename T >
RaveVector< T > axisAngleFromMatrix (const RaveTransformMatrix< T > &rotation)

  Converts the rotation of a matrix into axis-angle representation.

- template<typename T >
RaveTransformMatrix< T > transformLookat (const RaveVector< T > &vlookat, const RaveVector< T > &vcamerapos, const RaveVector< T > &vcameraup)

Returns a camera matrix that looks along a ray with a desired up vector.

- OPENRAVE_API dReal RaveExp (dReal f)

  exponential

- OPENRAVE_API dReal RaveLog (dReal f)

  logarithm

- OPENRAVE_API dReal RaveCos (dReal f)

  cosine

- OPENRAVE_API dReal RaveSin (dReal f)

  sine

- OPENRAVE_API dReal RaveTan (dReal f)

  tangent

- OPENRAVE_API dReal RaveLog2 (dReal f)

  base 2 logarithm

- OPENRAVE_API dReal RaveLog10 (dReal f)

  base 10 logarithm

- OPENRAVE_API dReal RaveAcos (dReal f)

  arccosine

- OPENRAVE_API dReal RaveAsin (dReal f)

  arcsine

- OPENRAVE_API dReal RaveAtan2 (dReal fy, dReal fx)

  arctangent2 covering entire circle

- OPENRAVE_API dReal RavePow (dReal fx, dReal fy)

  power x^y

- OPENRAVE_API dReal RaveSqrt (dReal f)

  square-root

- OPENRAVE_API dReal RaveFabs (dReal f)

  absolute value

9.4.1 Detailed Description

A set of classes and functions that provide the basic affine math operations with vectors, matrices, and quaternions.

9.4.2 Function Documentation

9.4.2.1 RaveVector<T> OpenRAVE::geometry::axisAngleFromMatrix (const RaveTransformMatrix< T > & *rotation*)

Converts the rotation of a matrix into axis-angle representation.

Parameters

rotation 3x3 rotation matrix

9.4.2.2 RaveVector<T> OpenRAVE::geometry::axisAngleFromQuat (const RaveVector< T > & *quat*)

Converts a quaternion into the axis-angle representation.

Parameters

quat quaternion, (s,vx,vy,vz)

9.4.2.3 RaveTransformMatrix<T> OpenRAVE::geometry::matrixFromAxisAngle (const RaveVector< T > & *axisangle*)

Converts an axis-angle rotation to a 3x3 matrix.

Parameters

axis unit axis $*$ rotation angle (radians), 3 values

9.4.2.4 RaveTransformMatrix<T> OpenRAVE::geometry::matrixFromAxisAngle (const RaveVector< T > & *axis*, T *angle*)

Converts an axis-angle rotation to a 3x3 matrix.

Parameters

axis unit axis, 3 values

angle rotation angle (radians)

9.4.2.5 void OpenRAVE::geometry::matrixFromQuat (RaveTransformMatrix< T > & *rotation*, const RaveVector< T > & *quat*)

Converts a quaternion to a 3x3 matrix.

Parameters

$\rightarrow$ ***rotation***

$\leftarrow$ ***quat*** quaternion, (s,vx,vy,vz)

9.4.2.6 RaveTransformMatrix<T> OpenRAVE::geometry::matrixFromQuat (const RaveVector< T > & *quat*)

Converts a quaternion to a 3x3 matrix.

Parameters

$\leftarrow$ ***quat*** quaternion, (s,vx,vy,vz)

9.4.2.7 std::pair<T, RaveVector<T> > OpenRAVE::geometry::normalizeAxisRotation (const RaveVector< T > & *axis*, const RaveVector< T > & *quat*)

Find the rotation theta around axis such that rot(axis,theta) $*$ q is closest to the identity rotation.

Parameters

← ***axis*** axis to minimize rotation about
← ***quat*** input

Returns

The angle that minimizes the rotation along with the normalized rotation rot(axis,theta)∗q

9.4.2.8 RaveVector<T> OpenRAVE::geometry::quatFromAxisAngle (const RaveVector< T > & *axisangle*)

Converts an axis-angle rotation into a quaternion.

Parameters

axisangle unit axis ∗ rotation angle (radians), 3 values

9.4.2.9 RaveVector<T> OpenRAVE::geometry::quatFromAxisAngle (const RaveVector< T > & *axis*, T *angle*)

Converts an axis-angle rotation into a quaternion.

Parameters

axis unit axis, 3 values
angle rotation angle (radians)

Examples:

orplanning_ik.cpp.

9.4.2.10 RaveVector<T> OpenRAVE::geometry::quatFromMatrix (const RaveTransformMatrix< T > & *rotation*)

Converts the rotation of a matrix into a quaternion.

Parameters

t transform for extracting the 3x3 rotation.

9.4.2.11 RaveVector<T> OpenRAVE::geometry::quatInverse (const RaveVector< T > & *quat*)

Inverted a quaternion rotation.

Parameters

quat quaternion, (s,vx,vy,vz)

9.4.2.12 RaveVector<T> OpenRAVE::geometry::quatMultiply (const RaveVector< T > & *quat0*, const RaveVector< T > & *quat1*)

Multiply two quaternions.

Parameters

quat0 quaternion, (s,vx,vy,vz)

quat1 quaternion, (s,vx,vy,vz)

Examples:

orplanning_ik.cpp.

9.4.2.13 RaveVector<T> OpenRAVE::geometry::quatRotate (const RaveVector< T > & *q*, const RaveVector< T > & *t*)

transform a vector by a quaternion

Parameters

9.4.2.14 RaveVector<T> OpenRAVE::geometry::quatRotateDirection (const RaveVector< T > & *sourcedir*, const RaveVector< T > & *targetdir*)

Return the minimal quaternion that orients sourcedir to targetdir.

Parameters

- ***sourcedir*** direction of the original vector, 3 values
- ***targetdir*** new direction, 3 values

9.4.2.15 RaveVector<T> OpenRAVE::geometry::quatSlerp (const RaveVector< T > & *quat0*, const RaveVector< T > & *quat1*, T *t*)

Sphereical linear interpolation between two quaternions.

Parameters

- ***quat0*** quaternion, (s,vx,vy,vz)
- ***quat1*** quaternion, (s,vx,vy,vz)
- ***t*** real value in [0,1]. 0 returns quat1, 1 returns quat2

9.4.2.16 OPENRAVE_API dReal OpenRAVE::RaveExp (dReal *f*)

exponential

Wrappers of common basic math functions, allows OpenRAVE to control the precision requirements.

9.4.2.17 RaveTransformMatrix<T> OpenRAVE::geometry::transformLookat (const RaveVector< T > & *vlookat*, const RaveVector< T > & *vcamerapos*, const RaveVector< T > & *vcameraup*)

Returns a camera matrix that looks along a ray with a desired up vector.

Parameters

- ← ***vlookat*** the point space to look at, the camera will rotation and zoom around this point
- ← ***vcampos*** the position of the camera in space
- ← ***vcamup*** vector from the camera

10 Namespace Documentation

10.1 boost Namespace Reference

Modifications controlling boost library behavior.

10.1.1 Detailed Description

Modifications controlling boost library behavior.

10.2 OpenRAVE Namespace Reference

The entire OpenRAVE library.

Namespaces

- namespace geometry

  Templated math and geometric functions.

- namespace mathextra

  Extra math routines that are useful to have but don't really belong anywhere.

Classes

- class CollisionReport

  Holds information about a particular collision that occured.

- class CollisionCheckerBase

  ***[interface]** Responsible for all collision checking queries of the environment. **If not specified, method is not multi-thread safe.** See Collision Checker Concepts.*

- class CollisionOptionsStateSaver

  Helper class to save and restore the collision options. If options are not supported and required is true, throws an exception.

- class ControllerBase

  ***[interface]** Abstract base class to encapsulate a local controller. **If not specified, method is not multi-thread safe.** See Controller Concepts.*

- class MultiController

controller that manage multiple controllers, allows users to easily set multiple controllers for one robot.

- class EnvironmentBase

  Maintains a world state, which serves as the gateway to all functions offered through OpenRAVE. See Environment Concepts.

- class IkSolverBase

  ***[interface]** Base class for all Inverse Kinematic solvers. **If not specified, method is not multi-thread safe.** See Inverse Kinematics Solver Concepts.*

- class InterfaceBase

  ***[interface]** Base class for all interfaces that OpenRAVE provides. See Base Interface Concepts.*

- class KinBody

  ***[interface]** A kinematic body of links and joints. **If not specified, method is not multi-thread safe.** See Kinematics Body Concepts.*

- class ModuleBase

  ***[interface]** A loadable module of user code meant to solve a specific domain. **If not specified, method is not multi-thread safe.** See Module Concepts.*

- class openrave_exception

  Exception that all OpenRAVE internal methods throw; the error codes are held in OpenRAVEErrorCode.

- class UserData

  base class for all user data

- class XMLReadable

  base class for readable interfaces

- class BaseXMLReader
- class DummyXMLReader

  reads until the tag ends

- class ConfigurationSpecification

  A configuration specification references values in the environment that then define a configuration-space which can be searched for.

- class IkParameterization

  Parameterization of basic primitives for querying inverse-kinematics solutions.

- class PhysicsEngineBase

  [interface] *The physics engine interfaces supporting simulations and dynamics. See Physics Engine Concepts.*

- class PlannerBase

  [interface] *Planner interface that generates trajectories for target objects to follow through the environment.* ***If not specified, method is not multi-thread safe.*** *See Planner Concepts.*

- class PLUGININFO

  Holds all the OpenRAVE-specific information provided by a plugin.

- class RobotBase

  [interface] *A robot is a kinematic body that has attached manipulators, sensors, and controllers.* ***If not specified, method is not multi-thread safe.*** *See Robot Concepts.*

- class SensorBase

  [interface] *A sensor measures physical properties from the environment.* ***If not specified, method is not multi-thread safe.*** *See Sensor Concepts.*

- class SensorSystemBase

  [interface] *Used to manage the creation and destruction of bodies. See Sensor System Concepts.*

- class SimpleSensorSystem

  A very simple sensor system example that manages raw detection data.

- class SpaceSamplerBase

  [interface] *Contains space samplers commonly used in planners.* ***If not specified, method is not multi-thread safe.*** *See SpaceSampler Concepts.*

- class TrajectoryBase

  [interface] *Encapsulate a time-parameterized trajectories of robot configurations.* ***If not specified, method is not multi-thread safe.*** `Trajectory Concepts`

- class GraphHandle

  Handle holding the plot from the viewers. The plot will continue to be drawn as long as a reference to this handle is held.

- class ViewerBase

  [interface] *Base class for the graphics and gui engine that renders the environment and provides visual sensor information.* ***If not specified, method is not multi-thread safe.*** *See Viewer Concepts.*

Typedefs

- typedef CollisionReport COLLISIONREPORT RAVE_DEPRECATED
- typedef boost::weak_ptr< EnvironmentBase > EnvironmentBaseWeakPtr

  Cloning Options for interfaces and environments.

- typedef InterfaceBasePtr(∗ PluginExportFn_OpenRAVECreateInterface)(InterfaceType type, const std::string &name, const char ∗pluginhash, const char ∗envhash, EnvironmentBasePtr penv)

  Create the interfaces, see CreateInterfaceValidated.

- typedef bool(∗ PluginExportFn_OpenRAVEGetPluginAttributes)(PLUGININFO ∗pinfo, int size, const char ∗infohash)

  Called to fill information about the plugin, see GetPluginAttributesValidated.

- typedef void(∗ PluginExportFn_DestroyPlugin)()

  Called before plugin is unloaded from openrave. See DestroyPlugin.

- typedef InterfaceBasePtr(∗ PluginExportFn_CreateInterface)(InterfaceType type, const std::string &name, const char ∗pluginhash, EnvironmentBasePtr penv)
- typedef bool(∗ PluginExportFn_GetPluginAttributes)(PLUGININFO ∗pinfo, int size)

Enumerations

- enum CollisionOptions {

  CO_Distance = 1, CO_UseTolerance = 2, CO_Contacts = 4, CO_RayAnyHit = 8,

  CO_ActiveDOFs = 16 }

  options for collision checker

- enum CollisionAction { CA_DefaultAction = 0, CA_Ignore = 1 }

  action to perform whenever a collision is detected between objects

- enum IkFilterReturn { IKFR_Success = 0, IKFR_Reject = 1, IKFR_Quit = 2 }

  Return value for the ik filter that can be optionally set on an ik solver.

- enum IkFilterOptions {

  IKFO_CheckEnvCollisions = 1, IKFO_IgnoreSelfCollisions = 2, IKFO_IgnoreJointLimits = 4, IKFO_IgnoreCustomFilters = 8,

  IKFO_IgnoreEndEffectorCollisions = 16 }

Controls what information gets validated when searching for an inverse kinematics solution.

- enum SerializationOptions {

  SO_Kinematics = 0x01, SO_Dynamics = 0x02, SO_BodyState = 0x04, SO_-NamesAndFiles = 0x08,

  SO_RobotManipulators = 0x10, SO_RobotSensors = 0x20, SO_Geometry = 0x40 }

  serialization options for interfaces

- enum OpenRAVEErrorCode { ,

  ORE_CommandNotSupported = 3 , ORE_InvalidPlugin = 5, ORE_-InvalidInterfaceHash = 6, ORE_NotImplemented = 7,

  ORE_InconsistentConstraints = 8 }

  OpenRAVE error codes

- enum DebugLevel { , Level_VerifyPlans = 0x80000000 }
- enum CloningOptions {

  Clone_Bodies = 1, Clone_Viewer = 2, Clone_Simulation = 4, Clone_-RealControllers = 8,

  Clone_Sensors = 16 }
- enum IkParameterizationType { ,

  IKP_Transform6D = 0x67000001, IKP_Rotation3D = 0x34000002, IKP_-Translation3D = 0x33000003, IKP_Direction3D = 0x23000004,

  IKP_Ray4D = 0x46000005, IKP_Lookat3D = 0x23000006, IKP_-TranslationDirection5D = 0x56000007, IKP_TranslationXY2D = 0x22000008,

  IKP_TranslationXYOrientation3D = 0x33000009, IKP_-TranslationLocalGlobal6D = 0x3600000a, IKP_NumberOfParameterizations = 10 }

  The types of inverse kinematics parameterizations supported.

- enum DOFAffine { ,

  DOF_X = 1, DOF_Y = 2, DOF_Z = 4, DOF_XYZ = DOF_X|DOF_Y|DOF_Z,

  DOF_RotationAxis = 8, DOF_Rotation3D = 16, DOF_RotationQuat = 32, DOF_RotationMask = (DOF_RotationAxis|DOF_Rotation3D|DOF_-RotationQuat),

  DOF_Transform = (DOF_XYZ|DOF_RotationQuat) }

  Selects which DOFs of the affine transformation to include in the active configuration.

- enum PhysicsEngineOptions { PEO_SelfCollisions = 1 }

  basic options for physics engine

- enum PlannerStatus { PS_Failed = 0, PS_HasSolution = 1, PS_Interrupted = 2 }

  the status of the PlanPath method. Used when PlanPath can be called multiple times to resume planning.

- enum PlannerAction { PA_None = 0, PA_Interrupt = 1, PA_ReturnWithAnySolution = 2 }

  action to send to the planner while it is planning. This is usually done by the user-specified planner callback function

- enum IntervalType { IT_Open = 0, IT_OpenStart = 1, IT_OpenEnd = 2, IT_Closed = 3 }

  Specifies the boundary conditions of intervals for sampling.

Functions

- std::string ChangeTextColor (int attribute, int fg, int bg)

  Change the text color (on either stdout or stderr) with an attr:fg:bg (thanks to Radu Rusu for the code).

- std::string ChangeTextColor (int attribute, int fg)

  Change the text color (on either stdout or stderr) with an attr:fg (thanks to Radu Rusu for the code).

- std::string ResetTextColor ()

  Reset the text color (on either stdout or stderr) to its original state (thanks to Radu Rusu for the code).

- OPENRAVE_API void RaveSetDebugLevel (int level)

  Sets the global openrave debug level. A combination of DebugLevel.

- OPENRAVE_API int RaveGetDebugLevel ()

  Returns the openrave debug level.

- const char ∗ RaveGetSourceFilename (const char ∗pfilename)

  extracts only the filename

- DefineRavePrintfW (_FATALLEVEL) DefineRavePrintfW(_ERRORLEVEL) DefineRavePrintfW(_WARNLEVEL) DefineRavePrintfW(_DEBUGLEVEL) DefineRavePrintfW(_VERBOSELEVEL) DefineRavePrintfA(_FATALLEVEL) DefineRavePrintfA(_ERRORLEVEL) DefineRavePrintfA(_WARNLEVEL) DefineRavePrintfA(_DEBUGLEVEL) DefineRavePrintfA(_VERBOSELEVEL) enum InterfaceType *adds the function name and line number to an openrave exception*

- OPENRAVE_API int RaveGetIndexFromAffineDOF (int affinedofs, DOFAffine dof)

  Given a mask of affine dofs and a dof inside that mask, returns the index where the value could be found.

- OPENRAVE_API DOFAffine RaveGetAffineDOFFromIndex (int affinedofs, int index)

  Given a mask of affine dofs and an index into the array, returns the affine dof that is being referenced.

- OPENRAVE_API int RaveGetAffineDOF (int affinedofs)

  Returns the degrees of freedom needed to represent all the values in the affine dof mask.

- OPENRAVE_API void RaveGetAffineDOFValuesFromTransform (std::vector< dReal >::iterator itvalues, const Transform &t, int affinedofs, const Vector &vActvAffineRotationAxis=Vector(0, 0, 1))

  Converts the transformation matrix into the specified affine values format.

- OPENRAVE_API void RaveGetTransformFromAffineDOFValues (Transform &t, std::vector< dReal >::const_iterator itvalues, int affinedofs, const Vector &vActvAffineRotationAxis=Vector(0, 0, 1))

  Converts affine dof values into a transform.

- OPENRAVE_API void RaveInitRandomGeneration (uint32_t seed)
- OPENRAVE_API uint32_t RaveRandomInt ()
- OPENRAVE_API float RaveRandomFloat (IntervalType interval=IT_Closed)
- OPENRAVE_API double RaveRandomDouble (IntervalType interval=IT_Closed)
- bool RaveParseDirectories (const char *pdirs, std::vector< std::string > &vdirs)

  separates the directories from a string and returns them in a vector

- OPENRAVE_API std::ostream & operator<< (std::ostream &O, const PlannerBase::PlannerParameters &v)
- OPENRAVE_API std::istream & operator>> (std::istream &I, PlannerBase::PlannerParameters &v)
- OPENRAVE_CORE_API EnvironmentBasePtr RaveCreateEnvironment ()
- OPENRAVE_CORE_API EnvironmentBasePtr CreateEnvironment (bool bLoadAllPlugins=true) RAVE_DEPRECATED

  - OPENRAVE_API dReal RaveExp (dReal f)

exponential

- OPENRAVE_API dReal RaveLog (dReal f)
  logarithm
- OPENRAVE_API dReal RaveCos (dReal f)
  cosine
- OPENRAVE_API dReal RaveSin (dReal f)
  sine
- OPENRAVE_API dReal RaveTan (dReal f)
  tangent
- OPENRAVE_API dReal RaveLog2 (dReal f)
  base 2 logarithm
- OPENRAVE_API dReal RaveLog10 (dReal f)
  base 10 logarithm
- OPENRAVE_API dReal RaveAcos (dReal f)
  arccosine
- OPENRAVE_API dReal RaveAsin (dReal f)
  arcsine
- OPENRAVE_API dReal RaveAtan2 (dReal fy, dReal fx)
  arctangent2 covering entire circle
- OPENRAVE_API dReal RavePow (dReal fx, dReal fy)
  power x^y
- OPENRAVE_API dReal RaveSqrt (dReal f)
  square-root
- OPENRAVE_API dReal RaveFabs (dReal f)
  absolute value

Global Functionality - Interface Creation, Plugin Management, Logging

- const char * RaveGetInterfaceHash (InterfaceType type)
  Returns the a 16 character null-terminated string specifying a hash of the interfaces used for checking changes.

- template< typename T >
  boost::shared_ptr< T > RaveInterfaceCast (InterfaceBasePtr pinterface)

  Safely casts from the base interface class to an openrave interface using static_-pointer_cast.

- template< typename T >
  boost::shared_ptr< T const > RaveInterfaceConstCast (InterfaceBaseConstPtr pinterface)

  Safely casts from the base interface class to an openrave interface using static_-pointer_cast.

- OPENRAVE_API const std::map< InterfaceType, std::string > & RaveGetInterfaceNamesMap ()

  returns a lower case string of the interface type

- OPENRAVE_API const std::string & **RaveGetInterfaceName** (InterfaceType type)
- OPENRAVE_API const std::map< IkParameterizationType, std::string > & RaveGetIkParameterizationMap ()

  returns a string of the ik parameterization type names (can include upper case in order to match IkParameterizationType)

- OPENRAVE_API std::string RaveGetHomeDirectory ()

  Returns the openrave home directory where settings, cache, and other files are stored.

- OPENRAVE_API std::string RaveFindDatabaseFile (const std::string &filename, bool bRead=true)

  Searches for a filename in the database and returns a full path/URL to it.

- OPENRAVE_API int RaveInitialize (bool bLoadAllPlugins=true, int level=Level_Info)

  Explicitly initializes the global OpenRAVE state (optional).

- OPENRAVE_API void RaveInitializeFromState (UserDataPtr globalstate)

  Initializes the global state from an already loaded OpenRAVE environment.

- OPENRAVE_API UserDataPtr RaveGlobalState ()

  A pointer to the global openrave state.

- OPENRAVE_API void RaveDestroy ()

  Destroys the entire OpenRAVE state and all loaded environments.

- OPENRAVE_API void RaveGetPluginInfo (std::list< std::pair< std::string, PLUGININFO > > &plugins)

  Get all the loaded plugins and the interfaces they support.

- OPENRAVE_API void RaveGetLoadedInterfaces (std::map< InterfaceType, std::vector< std::string > > &interfacenames)

  Get a list of all the loaded interfaces.

- OPENRAVE_API void RaveReloadPlugins ()

  Reloads all the plugins.

- OPENRAVE_API bool RaveLoadPlugin (const std::string &libraryname)

  Load a plugin and its interfaces.

- OPENRAVE_API bool RaveHasInterface (InterfaceType type, const std::string &interfacename)

  Returns true if interface can be created, otherwise false.

- OPENRAVE_API InterfaceBasePtr **RaveCreateInterface** (EnvironmentBasePtr penv, InterfaceType type, const std::string &interfacename)
- OPENRAVE_API RobotBasePtr **RaveCreateRobot** (EnvironmentBasePtr penv, const std::string &name="")
- OPENRAVE_API PlannerBasePtr **RaveCreatePlanner** (EnvironmentBasePtr penv, const std::string &name)
- OPENRAVE_API SensorSystemBasePtr **RaveCreateSensorSystem** (EnvironmentBasePtr penv, const std::string &name)
- OPENRAVE_API ControllerBasePtr **RaveCreateController** (EnvironmentBasePtr penv, const std::string &name)
- OPENRAVE_API ModuleBasePtr **RaveCreateModule** (EnvironmentBasePtr penv, const std::string &name)
- OPENRAVE_API ModuleBasePtr **RaveCreateProblem** (EnvironmentBasePtr penv, const std::string &name)
- OPENRAVE_API ModuleBasePtr **RaveCreateProblemInstance** (EnvironmentBasePtr penv, const std::string &name)
- OPENRAVE_API IkSolverBasePtr **RaveCreateIkSolver** (EnvironmentBasePtr penv, const std::string &name)
- OPENRAVE_API PhysicsEngineBasePtr **RaveCreatePhysicsEngine** (EnvironmentBasePtr penv, const std::string &name)
- OPENRAVE_API SensorBasePtr **RaveCreateSensor** (EnvironmentBasePtr penv, const std::string &name)
- OPENRAVE_API CollisionCheckerBasePtr **RaveCreateCollisionChecker** (EnvironmentBasePtr penv, const std::string &name)
- OPENRAVE_API ViewerBasePtr **RaveCreateViewer** (EnvironmentBasePtr penv, const std::string &name)
- OPENRAVE_API SpaceSamplerBasePtr **RaveCreateSpaceSampler** (EnvironmentBasePtr penv, const std::string &name)
- OPENRAVE_API KinBodyPtr **RaveCreateKinBody** (EnvironmentBasePtr penv, const std::string &name="")
- OPENRAVE_API TrajectoryBasePtr RaveCreateTrajectory (EnvironmentBasePtr penv, const std::string &name="")

  Return an empty trajectory instance.

- OPENRAVE_API TrajectoryBasePtr **RaveCreateTrajectory** (EnvironmentBasePtr penv, int dof) RAVE_DEPRECATED
- OPENRAVE_API UserDataPtr RaveRegisterInterface (InterfaceType type, const std::string &name, const char *interfacehash, const char *envhash, const boost::function< InterfaceBasePtr(EnvironmentBasePtr, std::istream &)> &createfn)

    Registers a function to create an interface, this allows the interface to be created by other modules.

- OPENRAVE_API UserDataPtr RaveRegisterXMLReader (InterfaceType type, const std::string &xmltag, const CreateXMLReaderFn &fn)

    Registers a custom xml reader for a particular interface.

- OPENRAVE_API int RaveGetEnvironmentId (EnvironmentBasePtr penv)

    return the environment's unique id, returns 0 if environment could not be found or not registered

- OPENRAVE_API EnvironmentBasePtr RaveGetEnvironment (int id)

    get the environment from its unique id

- OPENRAVE_API void RaveGetEnvironments (std::list< EnvironmentBasePtr > &listenvironments)

    Return all the created OpenRAVE environments.

- OPENRAVE_API BaseXMLReaderPtr RaveCallXMLReader (InterfaceType type, const std::string &xmltag, InterfaceBasePtr pinterface, const AttributesList &atts)

    Returns the current registered reader for the interface type/xmlid.

Variables

- static const dReal PI = (dReal)3.14159265358979323846

    openrave constant for PI, could be replaced by accurate precision number depending on choice of dReal.

10.2.1 Detailed Description

The entire OpenRAVE library.

10.2.2 Typedef Documentation

10.2.2.1 typedef InterfaceBasePtr(* PluginExportFn_CreateInterface)(InterfaceType type, const std::string &name, const char *pluginhash, EnvironmentBasePtr penv)

10.2.2.2 typedef bool(* PluginExportFn_GetPluginAttributes)(PLUGININFO *pinfo, int size)

10.2.2.3 typedef TrajectoryBase Trajectory RAVE_DEPRECATED

10.2.3 Enumeration Type Documentation

10.2.3.1 enum CloningOptions

Enumerator:

Clone_Bodies clone all the bodies/robots of the environment, exclude attached interfaces like sensors/controllers

Clone_Viewer clone the viewer type, although figures won't be copied, new viewer does try to match views

Clone_Simulation clone the physics engine and simulation state (ie, timesteps, gravity)

Clone_RealControllers if specified, will clone the real controllers of all the robots, otherwise each robot gets ideal controller

Clone_Sensors if specified, will clone the sensors attached to the robot and added to the environment

10.2.3.2 enum CollisionAction

action to perform whenever a collision is detected between objects

Enumerator:

CA_DefaultAction let the physics/collision engine resolve the action

CA_Ignore do nothing

10.2.3.3 enum CollisionOptions

options for collision checker

Enumerator:

CO_Distance Compute distance measurements, this is usually slow and not all checkers support it.

CO_UseTolerance not used

CO_Contacts Return the contact points of the collision in the CollisionReport. Note that this takes longer to compute.

CO_RayAnyHit When performing collision with rays, if this is set, algorithm just returns any hit instead of the closest (can be faster).

CO_ActiveDOFs Allows planners to greatly reduce redundant collision checks. If set and the target object is a robot, then only the links controlled by the currently set active DOFs and their attached bodies will be checked for collisions.

The things that ∗∗will not be∗∗ checked for collision are:

- links that do not remove with respect to each other as a result of moving the active dofs.

10.2.3.4 enum DebugLevel

Enumerator:

Level_VerifyPlans if set, should verify every plan returned. the verification is left up to the planners or the modules calling the planners. See planningutils::ValidateTrajectory

10.2.3.5 enum DOFAffine

Selects which DOFs of the affine transformation to include in the active configuration.

Enumerator:

DOF_X can move in the x direction

DOF_Y can move in the y direction

DOF_Z can move in the z direction

DOF_XYZ moves in xyz direction

DOF_RotationAxis can rotate around an axis (1 dof)

DOF_Rotation3D can rotate freely (3 dof), the parameterization is theta $*$ v, where v is the rotation axis and theta is the angle about that axis

DOF_RotationQuat can rotate freely (4 dof), parameterization is a quaternion. In order for limits to work correctly, the quaternion is in the space of _vRotationQuatLimitStart. _vRotationQuatLimitStart is always left-multiplied before setting the transform!

DOF_RotationMask mask for all bits representing 3D rotations

DOF_Transform translate and rotate freely in 3D space

10.2.3.6 enum IkFilterOptions

Controls what information gets validated when searching for an inverse kinematics solution.

Enumerator:

IKFO_CheckEnvCollisions will check environment collisions with the robot (not checked by default)

IKFO_IgnoreSelfCollisions will not check the self-collision of the robot (checked by default)

IKFO_IgnoreJointLimits will not check the joint limits of the robot (checked by default)

IKFO_IgnoreCustomFilters will not use the custom filter, even if one is set

IKFO_IgnoreEndEffectorCollisions will not check collision with the environment and the end effector links and bodies attached to the end effector links. The end effector links are defined by RobotBase::Manipulator::GetChildLinks. Use this option when RobotBase::Manipulator::CheckEndEffectorCollision has already been called, or it is ok for the end effector to collide given the IK constraints. Self-collisions between the moving links and end effector are still checked.

10.2.3.7 enum IkFilterReturn

Return value for the ik filter that can be optionally set on an ik solver.

Enumerator:

IKFR_Success the ik solution is good

IKFR_Reject reject the ik solution

IKFR_Quit the ik solution is rejected and the ik call itself should quit with failure

10.2.3.8 enum IkParameterizationType

The types of inverse kinematics parameterizations supported.

The minimum degree of freedoms required is set in the upper 4 bits of each type. The number of values used to represent the parameterization (>= dof) is the next 4 bits. The lower bits contain a unique id of the type.

Enumerator:

IKP_Transform6D end effector reaches desired 6D transformation

IKP_Rotation3D end effector reaches desired 3D rotation

IKP_Translation3D end effector origin reaches desired 3D translation

IKP_Direction3D direction on end effector coordinate system reaches desired direction

IKP_Ray4D ray on end effector coordinate system reaches desired global ray

IKP_Lookat3D direction on end effector coordinate system points to desired 3D position

IKP_TranslationDirection5D end effector origin and direction reaches desired 3D translation and direction. Can be thought of as Ray IK where the origin of the ray must coincide.

IKP_TranslationXY2D 2D translation along XY plane

IKP_TranslationXYOrientation3D 2D translation along XY plane and 1D rotation around Z axis. The offset of the rotation is measured starting at +X, so at +X is it 0, at +Y it is pi/2.

IKP_TranslationLocalGlobal6D local point on end effector origin reaches desired 3D global point

IKP_NumberOfParameterizations number of parameterizations (does not count IKP_None)

10.2.3.9 enum IntervalType

Specifies the boundary conditions of intervals for sampling.

Enumerator:

IT_Open (a,b)

IT_OpenStart (a,b]

IT_OpenEnd [a,b)

IT_Closed [a,b]

10.2.3.10 enum OpenRAVEErrorCode

OpenRAVE error codes

Enumerator:

ORE_CommandNotSupported string command could not be parsed or is not supported

ORE_InvalidPlugin shared object is not a valid plugin

ORE_InvalidInterfaceHash interface hashes do not match between plugins

ORE_NotImplemented function is not implemented by the interface.

ORE_InconsistentConstraints return solutions or trajectories do not follow the constraints of the planner/module

10.2.3.11 enum PhysicsEngineOptions

basic options for physics engine

Enumerator:

PEO_SelfCollisions if set, physics engine will use contact forces from self-collisions

10.2.3.12 enum PlannerAction

action to send to the planner while it is planning. This is usually done by the user-specified planner callback function

Enumerator:

PA_None no action

PA_Interrupt interrupt the planner and return to user

PA_ReturnWithAnySolution return quickly with any path

10.2.3.13 enum PlannerStatus

the status of the PlanPath method. Used when PlanPath can be called multiple times to resume planning.

Enumerator:

PS_Failed planner failed

PS_HasSolution planner succeeded

PS_Interrupted planning was interrupted, but can be resumed by calling PlanPath again

10.2.3.14 enum SerializationOptions

serialization options for interfaces

Enumerator:

SO_Kinematics kinematics information

SO_Dynamics dynamics information

SO_BodyState state of the body

SO_NamesAndFiles resource files and names

SO_RobotManipulators serialize robot manipulators

SO_RobotSensors serialize robot sensors

SO_Geometry geometry information (for collision detection)

10.2.4 Function Documentation

10.2.4.1 OPENRAVE_CORE_API EnvironmentBasePtr OpenRAVE::CreateEnvironment (bool *bLoadAllPlugins* = `true`)

10.2.4.2 OpenRAVE::DefineRavePrintfW (_FATALLEVEL)

adds the function name and line number to an openrave exception

Enumeration of all the interfaces.

10.2.4.3 OPENRAVE_API std::ostream& OpenRAVE::operator<< (std::ostream & *O*, const PlannerBase::PlannerParameters & *v*)

`<PlannerParameters>`

tags

10.2.4.4 OPENRAVE_API std::istream& OpenRAVE::operator>> (std::istream & *I*, PlannerBase::PlannerParameters & *v*)

`<PlannerParameters>`

to be the first token. Parses stream until

`</PlannerParameters>`

reached

10.2.4.5 OPENRAVE_API BaseXMLReaderPtr OpenRAVE::RaveCallXMLReader (InterfaceType *type*, const std::string & *xmltag*, InterfaceBasePtr *pinterface*, const AttributesList & *atts*)

Returns the current registered reader for the interface type/xmlid.

Exceptions

openrave_exception Will throw with ORE_InvalidArguments if registered function could not be found.

10.2.4.6 OPENRAVE_CORE_API EnvironmentBasePtr OpenRAVE::RaveCreateEnvironment ()

Creates an OpenRAVE environment.

Parameters

bLoadAllPlugins passed into RaveInitialize

10.2.4.7 OPENRAVE_API void OpenRAVE::RaveDestroy ()

Destroys the entire OpenRAVE state and all loaded environments.

This functions should be always called before program shutdown in order to assure all resources are relased appropriately.

10.2.4.8 OPENRAVE_API std::string OpenRAVE::RaveFindDatabaseFile (const std::string & *filename*, bool *bRead* = `true`)

Searches for a filename in the database and returns a full path/URL to it.

Parameters

filename the relative filename in the database

bRead if true will only return a file if it exists. If false, will return the filename of the first valid database directory.

Returns

a non-empty string if a file could be found.

10.2.4.9 OPENRAVE_API int OpenRAVE::RaveGetAffineDOF (int *affinedofs*)

Returns the degrees of freedom needed to represent all the values in the affine dof mask.

Exceptions

openrave_exception throws if

10.2.4.10 OPENRAVE_API DOFAffine OpenRAVE::RaveGetAffineDOFFromIndex (int *affinedofs*, int *index*)

Given a mask of affine dofs and an index into the array, returns the affine dof that is being referenced.

Parameters

affinedofs a mask of DOFAffine values

index an index into the affine dof array

Exceptions

openrave_exception throws if dof if index is out of bounds

10.2.4.11 OPENRAVE_API void OpenRAVE::RaveGetAffineDOFValuesFromTransform (std::vector< dReal >::iterator *itvalues*, const Transform & *t*, int *affinedofs*, const Vector & *vActvAffineRotationAxis* = `Vector(0, 0, 1)`)

Converts the transformation matrix into the specified affine values format.

Parameters

→ *itvalues* an iterator to the vector to write the values to. Will write exactly RaveGetAffineDOF(affinedofs) values.

← *the* affine transformation to convert

← *affinedofs* the affine format to output values in

← *vActvAffineRotationAxis* optional rotation axis if affinedofs specified DOF_-RotationAxis

10.2.4.12 OPENRAVE_API EnvironmentBasePtr OpenRAVE::RaveGetEnvironment (int *id*)

get the environment from its unique id

Parameters

id the unique environment id returned by RaveGetEnvironmentId

10.2.4.13 OPENRAVE_API std::string OpenRAVE::RaveGetHomeDirectory ()

Returns the openrave home directory where settings, cache, and other files are stored.

On Linux/Unix systems, this is usually $HOME/.openrave, on Windows this is $HOMEPATH/.openrave

10.2.4.14 OPENRAVE_API int OpenRAVE::RaveGetIndexFromAffineDOF (int *affinedofs*, DOFAffine *dof*)

Given a mask of affine dofs and a dof inside that mask, returns the index where the value could be found.

Parameters

affinedofs a mask of DOFAffine values

dof a set of values inside affinedofs, the index of the first value is returned

Exceptions

openrave_exception throws if dof is not present in affinedofs

10.2.4.15 OPENRAVE_API void OpenRAVE::RaveGetPluginInfo (std::list< std::pair< std::string, PLUGININFO > > & *plugins*)

Get all the loaded plugins and the interfaces they support.

Parameters

plugins A list of plugins. Each entry has the plugin name and the interfaces it supports

10.2.4.16 OPENRAVE_API void OpenRAVE::RaveGetTransformFromAffineDOFValues (Transform & *t*, std::vector< dReal >::const_iterator *itvalues*, int *affinedofs*, const Vector & *vActvAffineRotationAxis* = `Vector(0, 0, 1)`)

Converts affine dof values into a transform.

Note that depending on what the dof values holds, only a part of the transform will be updated.

Parameters

→ ***t*** the output transform

← ***itvalues*** the start iterator of the affine dof values

← ***affinedofs*** the affine dof mask

← ***vActvAffineRotationAxis*** optional rotation axis if affinedofs specified DOF_RotationAxis

10.2.4.17 OPENRAVE_API UserDataPtr OpenRAVE::RaveGlobalState ()

A pointer to the global openrave state.

Returns

a managed pointer to the state.

10.2.4.18 OPENRAVE_API int OpenRAVE::RaveInitialize (bool *bLoadAllPlugins* = `true`, int *level* = `Level_Info`)

Explicitly initializes the global OpenRAVE state (optional).

Optional function to initialize openrave plugins and logging. Although environment creation will automatically make sure this function is called, users might want explicit control of when this happens.

Parameters

bLoadAllPlugins If true will load all the openrave plugins automatically that can be found in the OPENRAVE_PLUGINS environment path

Returns

0 if successful, otherwise an error code

10.2.4.19 OPENRAVE_API void OpenRAVE::RaveInitializeFromState (UserDataPtr *globalstate*)

Initializes the global state from an already loaded OpenRAVE environment.

Because of shared object boundaries, it is necessary to pass the global state pointer around. If using plugin.h, this function is automatically called by CreateInterfaceValidated. It is also called by and every InterfaceBase constructor.

Parameters

← ***globalstate***

10.2.4.20 OPENRAVE_API void OpenRAVE::RaveInitRandomGeneration (uint32_t *seed*)

10.2.4.21 boost::shared_ptr<T> OpenRAVE::RaveInterfaceCast (InterfaceBasePtr *pinterface*)

Safely casts from the base interface class to an openrave interface using static_pointer_cast.

The reason why dynamic_pointer_cast cannot be used is because interfaces might be created by different plugins, and the runtime type information will be different.

10.2.4.22 boost::shared_ptr<T const> OpenRAVE::RaveInterfaceConstCast (InterfaceBaseConstPtr *pinterface*)

Safely casts from the base interface class to an openrave interface using static_pointer_cast.

The reason why dynamic_pointer_cast cannot be used is because interfaces might be created by different plugins, and the runtime type information will be different.

10.2.4.23 OPENRAVE_API bool OpenRAVE::RaveLoadPlugin (const std::string & *libraryname*)

Load a plugin and its interfaces.

If the plugin is already loaded, will reload it.

Parameters

name the filename of the plugin to load

10.2.4.24 OPENRAVE_API double OpenRAVE::RaveRandomDouble (IntervalType *interval* = `IT_Closed`)

10.2.4.25 OPENRAVE_API float OpenRAVE::RaveRandomFloat (IntervalType *interval* = `IT_Closed`)

10.2.4.26 OPENRAVE_API uint32_t OpenRAVE::RaveRandomInt ()

10.2.4.27 OPENRAVE_API UserDataPtr OpenRAVE::RaveRegisterInterface (InterfaceType *type*, const std::string & *name*, const char * *interfacehash*, const char * *envhash*, const boost::function< InterfaceBasePtr(EnvironmentBasePtr, std::istream &)> & *createfn*)

Registers a function to create an interface, this allows the interface to be created by other modules.

Parameters

type interface type

name interface name

interfacehash the hash of the interface being created (use the global defines OPENRAVE_X_HASH)

envhash the hash of the environment (use the global define OPENRAVE_-ENVIRONMENT_HASH)

createfn functions to create the interface it takes two parameters: the environment and an istream to the rest of the interface creation arguments.

Returns

a handle if function is successfully registered. By destroying the handle, the interface will be automatically unregistered.

Exceptions

openrave_exception Will throw with ORE_InvalidInterfaceHash if hashes do not match

10.2.4.28 OPENRAVE_API UserDataPtr OpenRAVE::RaveRegisterXMLReader (InterfaceType *type*, const std::string & *xmltag*, const CreateXMLReaderFn & *fn*)

Registers a custom xml reader for a particular interface.

Once registered, anytime an interface is created through XML and the xmltag is seen, the function CreateXMLReaderFn will be called to get a reader for that tag

Parameters

xmltag the tag specified in xmltag is seen in the interface, the the custom reader will be created.

fn CreateXMLReaderFn(pinterface,atts) - passed in the pointer to the interface where the tag was seen along with the list of attributes

Returns

a pointer holding the registration, releasing the pointer will unregister the XML reader

10.2.4.29 OPENRAVE_API void OpenRAVE::RaveReloadPlugins ()

Reloads all the plugins.

The interfaces currently created remain will continue using the old plugins, so this function is safe in that plugins currently loaded remain loaded until the last interface that uses them is released.

10.3 OpenRAVE::geometry Namespace Reference

Templated math and geometric functions.

Classes

- class RaveVector

  Vector class containing 4 dimensions.

- class RaveTransform

  Affine transformation parameterized with quaterions.

- class RaveTransformMatrix

  Affine transformation parameterized with rotation matrices. Scales and shears are not supported.

- class ray

  A ray defined by an origin and a direction.

- class aabb

  An axis aligned bounding box.

- class obb

  An oriented bounding box.

- class triangle

  A triangle defined by 3 points.

- class frustum

  A pyramid with its vertex clipped.

- class RaveCameraIntrinsics

  intrinsic parameters for a camera.

Functions

- template< typename T >
  RaveVector< T > quatFromAxisAngle (const RaveVector< T > &axis, T angle)

  Converts an axis-angle rotation into a quaternion.

- template< typename T >
  RaveVector< T > quatFromAxisAngle (const RaveVector< T > &axisangle)

  Converts an axis-angle rotation into a quaternion.

- template< typename T >
  RaveVector< T > quatFromMatrix (const RaveTransformMatrix< T > &rotation)

  Converts the rotation of a matrix into a quaternion.

- template< typename T >
  RaveTransformMatrix< T > matrixFromQuat (const RaveVector< T > &quat)

  Converts a quaternion to a 3x3 matrix.

- template< typename T >
  void matrixFromQuat (RaveTransformMatrix< T > &rotation, const RaveVector< T > &quat)

  Converts a quaternion to a 3x3 matrix.

- template< typename T >
  RaveTransformMatrix< T > matrixFromAxisAngle (const RaveVector< T > &axis, T angle)

  Converts an axis-angle rotation to a 3x3 matrix.

- template< typename T >
  RaveTransformMatrix< T > matrixFromAxisAngle (const RaveVector< T > &axisangle)

  Converts an axis-angle rotation to a 3x3 matrix.

- template< typename T >
  RaveVector< T > quatMultiply (const RaveVector< T > &quat0, const RaveVector< T > &quat1)

  Multiply two quaternions.

- template< typename T >
  RaveVector< T > quatInverse (const RaveVector< T > &quat)

  Inverted a quaternion rotation.

- template< typename T >
  RaveVector< T > quatSlerp (const RaveVector< T > &quat0, const RaveVector< T > &quat1, T t)

  Sphereical linear interpolation between two quaternions.

- template<typename T >
RaveVector< T > quatRotate (const RaveVector< T > &q, const RaveVector< T > &t)

    transform a vector by a quaternion

- template<typename T >
RaveVector< T > quatRotateDirection (const RaveVector< T > &sourcedir, const RaveVector< T > &targetdir)

    Return the minimal quaternion that orients sourcedir to targetdir.

- template<typename T >
std::pair< T, RaveVector< T > > normalizeAxisRotation (const RaveVector< T > &axis, const RaveVector< T > &quat)

    Find the rotation theta around axis such that rot(axis,theta) $*$ q is closest to the identity rotation.

- template<typename T >
RaveVector< T > axisAngleFromQuat (const RaveVector< T > &quat)

    Converts a quaternion into the axis-angle representation.

- template<typename T >
RaveVector< T > axisAngleFromMatrix (const RaveTransformMatrix< T > &rotation)

    Converts the rotation of a matrix into axis-angle representation.

- template<typename T >
RaveTransformMatrix< T > transformLookat (const RaveVector< T > &vlookat, const RaveVector< T > &vcamerapos, const RaveVector< T > &vcameraup)

    Returns a camera matrix that looks along a ray with a desired up vector.

- template<typename T >
int insideQuadrilateral (const RaveVector< T > &v, const RaveVector< T > &verts)

    Tests a point inside a 3D quadrilateral.

- template<typename T >
int insideTriangle (const RaveVector< T > v, const triangle< T > &tri)

    Tests a point insdie a 3D triangle.

- template<typename T >
bool RayAABBTest (const ray< T > &r, const aabb< T > &ab)

    Test collision of a ray with an axis aligned bounding box.

- template<typename T >
  bool RayOBBTest (const ray< T > &r, const obb< T > &o)

  Test collision of a ray and an oriented bounding box.

- template<typename T >
  bool IsOBBinFrustum (const obb< T > &o, const frustum< T > &fr)

  Test collision of an oriented bounding box and a frustum.

- template<typename T , typename U >
  bool IsOBBinConvexHull (const obb< T > &o, const U &vplanes)

  Tests if an oriented bounding box is inside a 3D convex hull.

- template<typename T >
  bool TriTriCollision (const RaveVector< T > &u1, const RaveVector< T > &u2, const RaveVector< T > &u3, const RaveVector< T > &v1, const RaveVector< T > &v2, const RaveVector< T > &v3, RaveVector< T > &contactpos, RaveVector< T > &contactnorm)

  Test collision if two 3D triangles.
  Assuming triangle vertices are declared counter-clockwise!!

- template<typename T >
  obb< T > OBBFromAABB (const aabb< T > &ab, const RaveTransformMatrix< T > &t)

  Transform an axis aligned bounding box to an oriented bounding box.

- template<typename T >
  obb< T > OBBFromAABB (const aabb< T > &ab, const RaveTransform< T > &t)

  Transform an axis aligned bounding box to an oriented bounding box.

- template<typename T >
  obb< T > TransformOBB (const RaveTransform< T > &t, const obb< T > &o)

  Transforms an oriented bounding box.

- template<typename T >
  obb< T > TransformOBB (const RaveTransformMatrix< T > &t, const obb< T > &o)

  Transforms an oriented bounding box.

- template<typename T >
  bool AABBCollision (const aabb< T > &ab1, const aabb< T > &ab2)

  projects an obb along the world axes

Primitive Serialization functions.

Don't add new lines to the output << operators. Some applications use it to serialize the data to send across the network.

- template< typename U >
  std::ostream & **operator**<< (std::ostream &O, const RaveVector< U > &v)
- template< typename U >
  std::istream & **operator**>> (std::istream &I, RaveVector< U > &v)
- template< typename U >
  std::ostream & **operator**<< (std::ostream &O, const RaveTransform< U > &v)
- template< typename U >
  std::istream & **operator**>> (std::istream &I, RaveTransform< U > &v)
- template< typename U >
  std::ostream & **operator**<< (std::ostream &O, const ray< U > &r)
- template< typename U >
  std::istream & **operator**>> (std::istream &I, ray< U > &r)
- template< typename U >
  std::ostream & operator<< (std::ostream &O, const RaveTransformMatrix< U > &v)

  serialize in column order! This is the format transformations are passed across the network

- template< typename U >
  std::istream & operator>> (std::istream &I, RaveTransformMatrix< U > &v)

  de-serialize in column order! This is the format transformations are passed across the network

Distnace functions.

- template< typename T >
  T DistVertexOBBSq (const RaveVector< T > &v, const obb< T > &o)

  The minimum distance form the vertex to the oriented bounding box.

10.3.1 Detailed Description

Templated math and geometric functions.

10.4 OpenRAVE::mathextra Namespace Reference

Extra math routines that are useful to have but don't really belong anywhere.

Functions

- template<typename T >
  bool eig2 (const T *pfmat, T *peigs, T &fv1x, T &fv1y, T &fv2x, T &fv2y)
- template<typename T >
  void GetCovarBasisVectors (const T fCovariance[3][3], T vbasis[3][3])
- template<typename T >
  void svd3 (const T *A, T *U, T *D, T *V)
- template<typename T >
  bool inv2 (T *pf, T *pfres)

  takes the inverse of the 2x2 matrix pf and stores it into pfres, returns true if matrix is invertible
- template<typename T >
  T * _mult3_s4 (T *pfres, const T *pf1, const T *pf2)

  mult3 with a 3x3 matrix whose row stride is 16 bytes
- template<typename T >
  T * _mult3_s3 (T *pfres, const T *pf1, const T *pf2)

  mult3 with a 3x3 matrix whose row stride is 12 bytes
- template<typename T >
  T matrixdet3 (const T *pf, int stride)

  Compute the determinant of a 3x3 matrix whose row stride stride elements.
- template<typename T >
  T * _inv3 (const T *pf, T *pfres, T *pfdet, int stride)

  3x3 matrix inverse.
- template<typename T >
  T * _inv4 (const T *pf, T *pfres)

  4x4 matrix inverse.
- template<typename T >
  T * _transpose3 (const T *pf, T *pfres)

  Transpose a 3x3 matrix.
- template<typename T >
  T * _transpose4 (const T *pf, T *pfres)

  Transpose a 4x4 matrix.

10.4.1 Detailed Description

Extra math routines that are useful to have but don't really belong anywhere.

10.4.2 Function Documentation

10.4.2.1 T* OpenRAVE::mathextra::_inv3 (const T * *pf*, T * *pfres*, T * *pfdet*, int *stride*)

3x3 matrix inverse.

Parameters

- $\leftarrow$ ***pf*** the input 3x3 matrix
- $\rightarrow$ ***pf*** the result of the operation, can be the same matrix as pf
- $\rightarrow$ ***pfdet*** if not NULL, fills it with the determinant of the source matrix
- $\leftarrow$ ***stride*** the stride in elements between elements.

10.4.2.2 bool eig2 (const T * *pfmat*, T * *peigs*, T & *fv1x*, T & *fv1y*, T & *fv2x*, T & *fv2y*)

extract eigen values and vectors from a 2x2 matrix and returns true if all values are real returned eigen vectors are normalized

10.4.2.3 void OpenRAVE::mathextra::GetCovarBasisVectors (const T *fCovariance*[3][3], T *vbasis*[3][3])

Computes the eigenvectors of the covariance matrix and forms a basis

Parameters

- $\leftarrow$ ***fCovariance*** a symmetric 3x3 matrix.
- $\rightarrow$ ***vbasis*** the basis vectors extracted (form a right hand coordinate system).

10.4.2.4 void svd3 (const T * *A*, T * *U*, T * *D*, T * *V*)

SVD of a 3x3 matrix A such that A = U*diag(D)*V' The row stride for all matrices is 3*sizeof(T) bytes

Parameters

$\leftarrow$ ***A*** 3x3 matrix

$\rightarrow$ ***U*** 3x3 matrix

$\rightarrow$ ***D*** 3x1 matrix

$\rightarrow$ ***V*** 3x3 matrix

11 Class Documentation

11.1 aabb$<$ T $>$ Class Template Reference

An axis aligned bounding box.

11.1.1 Detailed Description

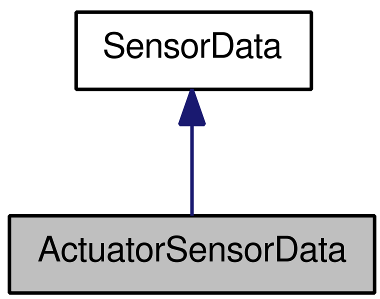
template$<$typename T$>$ class OpenRAVE::geometry::aabb$<$ T $>$

An axis aligned bounding box.

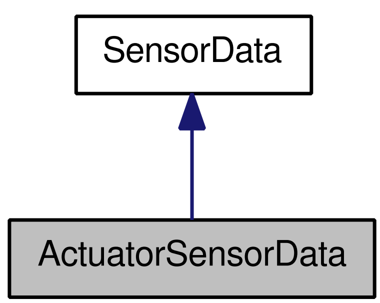
11.2 ActuatorSensorData Class Reference

An actuator for modeling motors and other mechanisms that produce torque/force. The actuator has only one degree of freedom.

Inheritance diagram for ActuatorSensorData:



Collaboration diagram for ActuatorSensorData:



Public Types

- enum ActuatorState {
  AS_Undefined = 0, AS_Idle = 1, AS_Moving = 2, AS_Stalled = 3,
  AS_Braked = 4 }

  the state of the actuator

Public Attributes

- dReal measuredcurrent

  measured current from the actuator

- dReal measuredtemperature

  measured temperature from the actuator

- dReal appliedcurrent

  current sent to the actuator

11.2.1 Detailed Description

An actuator for modeling motors and other mechanisms that produce torque/force. The actuator has only one degree of freedom.

11.2.2 Member Enumeration Documentation

11.2.2.1 enum ActuatorState

the state of the actuator

Enumerator:

AS_Undefined returned when no state is defined

AS_Idle this actuator is idle

AS_Moving this actuator is in motion from previous commands

AS_Stalled the actuator is stalled, needs to be unstalled by sending a ready signal

AS_Braked the actuator is braked

11.3 AttachedSensor Class Reference

Attaches a sensor to a link on the robot.

Public Member Functions

- virtual SensorBase::SensorDataPtr GetData () const
  retrieves the current data from the sensor
- virtual const std::string & GetStructureHash () const

11.3.1 Detailed Description

Attaches a sensor to a link on the robot.

11.3.2 Member Function Documentation

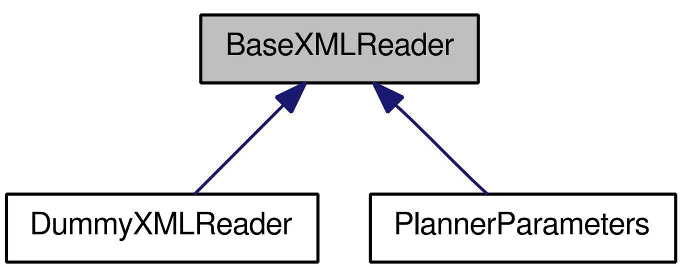
11.3.2.1 virtual const std::string& GetStructureHash () const [virtual]

Returns

hash of the sensor definition

11.4 BaseXMLReader Class Reference

Inheritance diagram for BaseXMLReader:



Public Types

- enum ProcessElement { PE_Pass = 0, PE_Support = 1, PE_Ignore = 2 }

Public Member Functions

- virtual XMLReadablePtr GetReadable ()
- virtual ProcessElement startElement (const std::string &name, const AttributesList &atts)=0
- virtual bool endElement (const std::string &name)=0
- virtual void characters (const std::string &ch)=0

Public Attributes

- std::string _filename

  XML filename/resource used for this class (can be empty).

11.4.1 Detailed Description

base class for all xml readers. XMLReaders are used to process data from xml files. Custom readers can be registered through EnvironmentBase. By default it can record all data that is encountered inside the xml reader

Examples:

customreader.cpp.

11.4.2 Member Enumeration Documentation

11.4.2.1 enum ProcessElement

Enumerator:

PE_Pass current tag was not supported, so pass onto another class

PE_Support current tag will be processed by this class

PE_Ignore current tag and all its children should be ignored

11.4.3 Member Function Documentation

11.4.3.1 virtual void characters (const std::string & *ch*) `[pure virtual]`

gets called for all data in between tags.

Parameters

ch a string to the data

11.4.3.2 virtual bool endElement (const std::string & *name*) `[pure virtual]`

Gets called at the end of each "</type>" expression. In this case, name is "type"

Parameters

name of the tag, will be always lower case

Returns

true if XMLReader has finished parsing (one condition is that name==_fieldname) , otherwise false

11.4.3.3 virtual XMLReadablePtr GetReadable () `[virtual]`

a readable interface that stores the information processsed for the current tag This pointer is used to the InterfaceBase class registered readers

11.4.3.4 virtual ProcessElement startElement (const std::string & *name*, const AttributesList & *atts*) `[pure virtual]`

Gets called in the beginning of each "<type>" expression. In this case, name is "type"

Parameters

name of the tag, will be always lower case

atts string of attributes where the first std::string is the attribute name and second is the value

Returns

true if tag is accepted and this class will process it, otherwise false

11.5 BodyState Class Reference

Stores the state of the current body that is published in a thread safe way from the environment without requiring locking the environment.

Public Attributes

- std::string strname

  name of the body

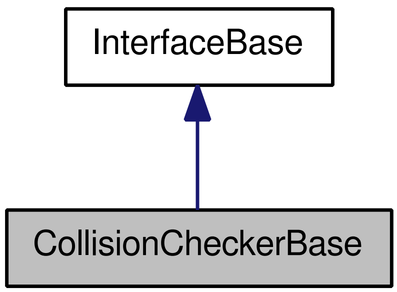
11.5.1 Detailed Description

Stores the state of the current body that is published in a thread safe way from the environment without requiring locking the environment.

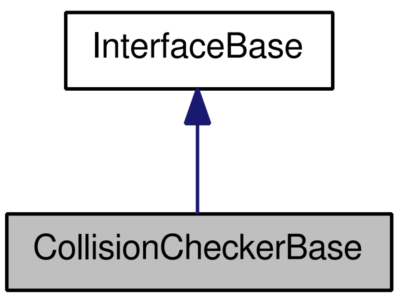
11.6 CollisionCheckerBase Class Reference

[interface] Responsible for all collision checking queries of the environment. **If not specified, method is not multi-thread safe.** See Collision Checker Concepts.

Inheritance diagram for CollisionCheckerBase:



Collaboration diagram for CollisionCheckerBase:



Public Member Functions

- virtual bool SetCollisionOptions (int collisionoptions)=0

  Set basic collision options using the CollisionOptions enum.

- virtual int GetCollisionOptions () const =0

  get the current collision options

- virtual bool InitKinBody (KinBodyPtr pbody)=0

  notified when a new body has been initialized in the environment

- virtual bool Enable (KinBodyConstPtr pbody, bool bEnable) RAVE_-DEPRECATED=0
- virtual bool EnableLink (KinBody::LinkConstPtr pbody, bool bEnable)=0

Static Public Member Functions

- static InterfaceType GetInterfaceTypeStatic ()

  return the static interface type this class points to (used for safe casting)

Collision specific functions.

Each function takes an optional pointer to a CollisionReport structure and returns true if collision occurs.

- virtual bool CheckCollision (KinBodyConstPtr pbody1, CollisionReportPtr report=CollisionReportPtr())=0

  checks collision of a body and a scene. Attached bodies are respected. If CO_-ActiveDOFs is set, will only check affected links of the body.

- virtual bool CheckCollision (KinBodyConstPtr pbody1, KinBodyConstPtr pbody2, CollisionReportPtr report=CollisionReportPtr())=0

  checks collision between two bodies. Attached bodies are respected. If CO_-ActiveDOFs is set, will only check affected links of the pbody1.

- virtual bool CheckCollision (KinBody::LinkConstPtr plink, CollisionReportPtr report=CollisionReportPtr())=0

  checks collision of a link and a scene. Attached bodies are ignored. CO_ActiveDOFs option is ignored.

- virtual bool CheckCollision (KinBody::LinkConstPtr plink1, KinBody::LinkConstPtr plink2, CollisionReportPtr report=CollisionReportPtr())=0

  checks collision of two links. Attached bodies are ignored. CO_ActiveDOFs option is ignored.

- virtual bool CheckCollision (KinBody::LinkConstPtr plink, KinBodyConstPtr pbody, CollisionReportPtr report=CollisionReportPtr())=0 *checks collision of a link and a body. Attached bodies for pbody are respected. CO_-ActiveDOFs option is ignored.*

- virtual bool CheckCollision (KinBody::LinkConstPtr plink, const std::vector< KinBodyConstPtr > &vbodyexcluded, const std::vector< KinBody::LinkConstPtr > &vlinkexcluded, CollisionReportPtr report=CollisionReportPtr())=0

  checks collision of a link and a scene. Attached bodies are ignored. CO_ActiveDOFs option is ignored.

- virtual bool CheckCollision (KinBodyConstPtr pbody, const std::vector< KinBodyConstPtr > &vbodyexcluded, const std::vector< KinBody::LinkConstPtr > &vlinkexcluded, CollisionReportPtr report=CollisionReportPtr())=0

  checks collision of a body and a scene. Attached bodies are respected. If CO_-ActiveDOFs is set, will only check affected links of pbody.

- virtual bool CheckCollision (const RAY &ray, KinBody::LinkConstPtr plink, CollisionReportPtr report=CollisionReportPtr())=0

  Check collision with a link and a ray with a specified length. CO_ActiveDOFs option is ignored.

- virtual bool CheckCollision (const RAY &ray, KinBodyConstPtr pbody, CollisionReportPtr report=CollisionReportPtr())=0

  Check collision with a link and a ray with a specified length.

- virtual bool CheckCollision (const RAY &ray, CollisionReportPtr report=CollisionReportPtr())=0

  Check collision with a body and a ray with a specified length. CO_ActiveDOFs option is ignored.

- virtual bool InitEnvironment ()=0
- virtual void DestroyEnvironment ()=0
- virtual bool CheckSelfCollision (KinBodyConstPtr pbody, CollisionReportPtr report=CollisionReportPtr())=0

  Checks self collision only with the links of the passed in body.

- virtual void **SetCollisionData** (KinBodyPtr pbody, UserDataPtr data)
- CollisionCheckerBasePtr **shared_collisionchecker** ()
- CollisionCheckerBaseConstPtr **shared_collisionchecker_const** () const

11.6.1 Detailed Description

[interface] Responsible for all collision checking queries of the environment. **If not specified, method is not multi-thread safe.** See Collision Checker Concepts.

11.6.2 Member Function Documentation

11.6.2.1 virtual bool CheckCollision (const RAY & *ray*, CollisionReportPtr *report* = `CollisionReportPtr()`) `[pure virtual]`

Check collision with a body and a ray with a specified length. CO_ActiveDOFs option is ignored.

Parameters

- ***ray*** holds the origin and direction. The length of the ray is the length of the direction.
- ***pbody*** the kinbody to look for collisions
- → ***report*** [optional] collision report to be filled with data about the collision. If a body was hit, CollisionReport::plink1 contains the hit link pointer.

11.6.2.2 virtual bool CheckCollision (const RAY & *ray*, KinBodyConstPtr *pbody*, CollisionReportPtr *report* = `CollisionReportPtr()`) `[pure virtual]`

Check collision with a link and a ray with a specified length.

Parameters

- ***ray*** holds the origin and direction. The length of the ray is the length of the direction.
- ***pbody*** the link to collide with. If CO_ActiveDOFs is set, will only check affected links of the body.
- → ***report*** [optional] collision report to be filled with data about the collision. If a body was hit, CollisionReport::plink1 contains the hit link pointer.

11.6.2.3 virtual bool CheckCollision (const RAY & *ray*, KinBody::LinkConstPtr *plink*, CollisionReportPtr *report* = `CollisionReportPtr()`) `[pure virtual]`

Check collision with a link and a ray with a specified length. CO_ActiveDOFs option is ignored.

Parameters

ray holds the origin and direction. The length of the ray is the length of the direction.

plink the link to collide with

→ ***report*** [optional] collision report to be filled with data about the collision. If a body was hit, CollisionReport::plink1 contains the hit link pointer.

11.6.2.4 virtual bool CheckSelfCollision (KinBodyConstPtr *pbody*, CollisionReportPtr *report* = `CollisionReportPtr()`) `[protected, pure virtual]`

Checks self collision only with the links of the passed in body.

Links that are joined together are ignored.

11.6.2.5 virtual void DestroyEnvironment () `[protected, pure virtual]`

called when environment switches to a different collision checker engine has to clear/deallocate any memory associated with KinBody::_pCollisionData

11.6.2.6 virtual bool Enable (KinBodyConstPtr *pbody*, bool *bEnable*) `[pure virtual]`

See also

EnableLink

11.6.2.7 virtual bool EnableLink (KinBody::LinkConstPtr *pbody*, bool *bEnable*) `[pure virtual]`

enables or disables a link from being considered in collisions

Returns

true if operation succeeded

11.6.2.8 virtual bool InitEnvironment () `[protected, pure virtual]`

called when environment sets this collision checker, checker assumes responsibility for KinBody::_pCollisionData checker should also gather all current bodies in the environment and put them in its collision space

11.7 CollisionOptionsStateSaver Class Reference

Helper class to save and restore the collision options. If options are not supported and required is true, throws an exception.

11.7.1 Detailed Description

Helper class to save and restore the collision options. If options are not supported and required is true, throws an exception.

11.8 CollisionReport Class Reference

Holds information about a particular collision that occured.

Public Attributes

- int options

  the options that the CollisionReport was called with

- KinBody::LinkConstPtr plink2

  the colliding links if a collision involves a bodies. Collisions do not always occur with 2 bodies like ray collisions, so these fields can be empty.

- int numCols

  this is the number of objects that collide with the object of interest

- std::vector< KinBody::LinkConstPtr > vLinkColliding

  objects colliding with this object

- dReal minDistance

  minimum distance from last query, filled if CO_Distance option is set

- int numWithinTol

  number of objects within tolerance of this object, filled if CO_UseTolerance option is set

- std::vector< CONTACT > contacts

  the convention is that the normal will be "out" of plink1's surface. Filled if CO_-UseContacts option is set.

11.8.1 Detailed Description

Holds information about a particular collision that occured.

Examples:

orcollision.cpp.

11.9 ConfigurationSpecification Class Reference

A configuration specification references values in the environment that then define a configuration-space which can be searched for.

Classes

- class Group

  A group referencing the values of one body in the environment.

Public Member Functions

- virtual int GetDOF () const

  return the dimension of the configuraiton space (degrees of freedom)

- virtual bool IsValid () const

  check if the groups form a contingous space

- virtual std::vector< Group >::const_iterator FindCompatibleGroup (const Group &g, bool exactmatch=false) const

  finds the most compatible group to the given group

- virtual std::vector< Group >::const_iterator FindTimeDerivativeGroup (const Group &g, bool exactmatch=false) const

  Return the most compatible group that represents the time-derivative data of the group.

- virtual void AddVelocityGroups (bool adddeltatime)

  adds a velocity group for every position group.

- virtual ConfigurationSpecification ConvertToVelocitySpecification () const

  converts all the groups to the corresponding velocity groups and returns the specification

- virtual ConfigurationSpecification GetTimeDerivativeSpecification (int timederivative) const

  returns a new specification of just particular time-derivative groups.

- virtual void ResetGroupOffsets ()

  set the offsets of each group in order to get a contiguous configuration space

- virtual int AddDeltaTime ()

  adds the deltatime tag to the end if one doesn't exist and returns the index into the configuration space

- virtual bool ExtractTransform (Transform &t, std::vector< dReal >::const_iterator itdata, KinBodyConstPtr pbody) const

  extracts an affine transform given the start of a configuration space point

- virtual bool ExtractIkParameterization (IkParameterization &ikparam, std::vector< dReal >::const_iterator itdata, int timederivative=0) const

  extracts an ikparameterization given the start of a configuration space point

- virtual bool ExtractAffineValues (std::vector< dReal >::iterator itvalues, std::vector< dReal >::const_iterator itdata, KinBodyConstPtr pbody, int affinedofs, int timederivative=0) const

  extracts the affine values

- virtual bool ExtractJointValues (std::vector< dReal >::iterator itvalues, std::vector< dReal >::const_iterator itdata, KinBodyConstPtr pbody, const std::vector< int > &indices, int timederivative=0) const

  extracts a body's joint values given the start of a configuration space point

- virtual bool ExtractDeltaTime (dReal &deltatime, std::vector< dReal >::const_iterator itdata) const

  extracts the delta time from the configuration if one exists

- virtual bool InsertJointValues (std::vector< dReal >::iterator itdata, std::vector< dReal >::const_iterator itvalues, KinBodyConstPtr pbody, const std::vector< int > &indices, int timederivative=0) const *inserts a set of joint values into a configuration space point*

- virtual bool InsertDeltaTime (std::vector< dReal >::iterator itdata, dReal deltatime)

  sets the deltatime field of the data if one exists

Static Public Member Functions

- static void ConvertGroupData (std::vector< dReal >::iterator ittargetdata, size_t targetstride, const Group &gtarget, std::vector< dReal >::const_iterator itsourcedata, size_t sourcestride, const Group &gsource, size_t numpoints, EnvironmentBaseConstPtr penv)

  given two compatible groups, convers data represented in the source group to data represented in the target group

- static void ConvertData (std::vector< dReal >::iterator ittargetdata, const ConfigurationSpecification &targetspec, std::vector< dReal >::const_iterator itsourcedata, const ConfigurationSpecification &sourcespec, size_t numpoints, EnvironmentBaseConstPtr penv, bool filluninitialized=true)

  Converts from one specification to another.

11.9.1 Detailed Description

A configuration specification references values in the environment that then define a configuration-space which can be searched for. It is composed of several groups targetting values for individual bodies. It is serialized into XML. The XML syntax is as follows:

```
<configuration>
  <group name="string" offset="#OFF1" dof="#D1" interpolation="string"/>
  <group name="string" offset="#OFF2" dof="#D2" interpolation="string"/>
</configuration>
```

Examples:

ormulticontrol.cpp.

11.9.2 Member Function Documentation

11.9.2.1 virtual void AddVelocityGroups (bool *adddeltatime*) [virtual]

adds a velocity group for every position group.

If velocities groups already exist, they are checked for and/or modified. Note that the configuration space might be re-ordered as a result of this function call.

Parameters

adddeltatime If true will add the 'deltatime' tag, which is necessary for trajectory sampling

11.9.2.2 static void ConvertData (std::vector< dReal >::iterator *ittargetdata*, const ConfigurationSpecification & *targetspec*, std::vector< dReal >::const_iterator *itsourcedata*, const ConfigurationSpecification & *sourcespec*, size_t *numpoints*, EnvironmentBaseConstPtr *penv*, bool *filluninitialized* = `true`) `[static]`

Converts from one specification to another.

Parameters

ittargetdata iterator pointing to start of target group data that should be overwritten

targetspec the target configuration specification

itsourcedata iterator pointing to start of source group data that should be read

sourcespec the source configuration specification

numpoints the number of points to convert. The target and source strides are gtarget.dof and gsource.dof

penv [optional] The environment which might be needed to fill in unknown data. Assumes environment is locked.

filluninitialized If there exists target groups that cannot be initialized, then will set default values to them.

11.9.2.3 static void ConvertGroupData (std::vector< dReal >::iterator *ittargetdata*, size_t *targetstride*, const Group & *gtarget*, std::vector< dReal >::const_iterator *itsourcedata*, size_t *sourcestride*, const Group & *gsource*, size_t *numpoints*, EnvironmentBaseConstPtr *penv*) `[static]`

given two compatible groups, convers data represented in the source group to data represented in the target group

Parameters

ittargetdata iterator pointing to start of target group data that should be overwritten

targetstride the number of elements that to go from the next target point. Necessary if numpoints > 1.

gtarget the target configuration group

itsourcedata iterator pointing to start of source group data that should be read

sourcestride the number of elements that to go from the next source point. Necessary if numpoints > 1.

gsource the source configuration group

numpoints the number of points to convert. The target and source strides are gtarget.dof and gsource.dof

penv [optional] The environment which might be needed to fill in unknown data. Assumes environment is locked.

Exceptions

openrave_exception throw f groups are incompatible

11.9.2.4 virtual ConfigurationSpecification ConvertToVelocitySpecification () const `[virtual]`

converts all the groups to the corresponding velocity groups and returns the specification

The velocity configuration space will have a one-to-one correspondence with the

11.9.2.5 virtual bool ExtractAffineValues (std::vector< dReal >::iterator *itvalues*, std::vector< dReal >::const_iterator *itdata*, KinBodyConstPtr *pbody*, int *affinedofs*, int *timederivative* = 0) const `[virtual]`

extracts the affine values

Looks for 'affine_X' groups. If pbody is not initialized, will choose the first affine_X found.

Parameters

inout] itvalues iterator to vector that holds the default values and will be overwritten with the new values. must be initialized

← ***itdata*** data in the format of this configuration specification.

← ***affinedofs*** the format of the affine dofs requested

← ***timederivative*** the time derivative of the data to extract

Returns

true if at least one group was found for extracting

11.9.2.6 virtual bool ExtractIkParameterization (IkParameterization & *ikparam*, std::vector< dReal >::const_iterator *itdata*, int *timederivative* = 0) const [virtual]

extracts an ikparameterization given the start of a configuration space point

Looks for 'ikparam' groups.

Parameters

inout] ikparam filled with ikparameterization (if found)

← ***itdata*** data in the format of this configuration specification

Returns

true if at least one group was found for extracting

11.9.2.7 virtual bool ExtractJointValues (std::vector< dReal >::iterator *itvalues*, std::vector< dReal >::const_iterator *itdata*, KinBodyConstPtr *pbody*, const std::vector< int > & *indices*, int *timederivative* = 0) const [virtual]

extracts a body's joint values given the start of a configuration space point

Looks for 'joint_X' groups. If pbody is not initialized, will choose the first joint_X found.

Parameters

inout] itvalues iterator to vector that holds the default values and will be overwritten with the new values. must be initialized

← ***itdata*** data in the format of this configuration specification.

← ***indices*** the set of DOF indices of the body to extract and write into itvalues.

← ***timederivative*** the time derivative of the data to extract

Returns

true if at least one group was found for extracting

11.9.2.8 virtual bool ExtractTransform (Transform & *t*, std::vector< dReal >::const_iterator *itdata*, KinBodyConstPtr *pbody*) const [virtual]

extracts an affine transform given the start of a configuration space point

Looks for 'affine_transform' groups. If pbody is not initialized, will choose the first affine_transform found.

Parameters

inout] t the transform holding the default values, which will be overwritten with the new values.

← ***itdata*** data in the format of this configuration specification.

Returns

true if at least one group was found for extracting

11.9.2.9 virtual std::vector<Group>::const_iterator FindCompatibleGroup (const Group & *g*, bool *exactmatch* = false) const [virtual]

finds the most compatible group to the given group

Parameters

g the group to query, only the Group::name and Group::dof values are used

exactmatch if true, will only return groups whose name exactly matches with g.name

Returns

an iterator part of _vgroups that represents the most compatible group. If no group is found, will return _vgroups.end()

11.9.2.10 virtual std::vector<Group>::const_iterator FindTimeDerivativeGroup (const Group & *g*, bool *exactmatch* = `false`) const `[virtual]`

Return the most compatible group that represents the time-derivative data of the group.

For example given a 'joint_values' group, this will return the closest 'joint_velocities' group.

Parameters

g the group to query, only the Group::name and Group::dof values are used

exactmatch if true, will only return groups whose name exactly matches with g.name

Returns

an iterator part of _vgroups that represents the most compatible group. If no group is found, will return _vgroups.end()

11.9.2.11 virtual ConfigurationSpecification GetTimeDerivativeSpecification (int *timederivative*) const `[virtual]`

returns a new specification of just particular time-derivative groups.

Parameters

timederivative the time derivative to query groups from. 0 is positions/joint values, 1 is velocities, 2 is accelerations, etc

11.9.2.12 virtual bool InsertDeltaTime (std::vector< dReal >::iterator *itdata*, dReal *deltatime*) `[virtual]`

sets the deltatime field of the data if one exists

Parameters

inout] itdata data in the format of this configuration specification.

← ***deltatime*** the delta time of the time stamp (from previous point)

Returns

true if at least one group was found for inserting

11.9.2.13 virtual bool InsertJointValues (std::vector< dReal >::iterator *itdata*, std::vector< dReal >::const_iterator *itvalues*, KinBodyConstPtr *pbody*, const std::vector< int > & *indices*, int *timederivative* = 0) const [virtual]

inserts a set of joint values into a configuration space point

Looks for 'joint_X' groups. If pbody is not initialized, will use the first joint_X found.

Parameters

inout] itdata data in the format of this configuration specification.
← ***itvalues*** iterator to joint values to write
← ***indices*** the set of DOF indices that itvalues represents.
← ***timederivative*** the time derivative of the data to insert

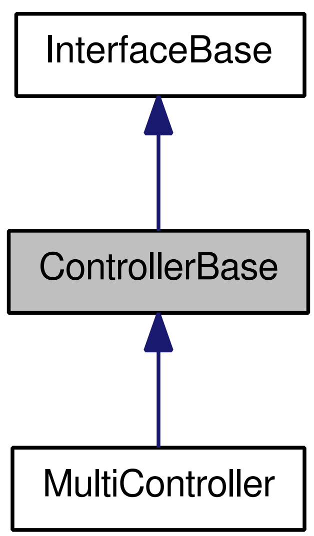
Returns

true if at least one group was found for inserting

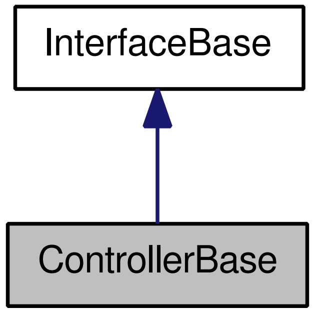
11.10 ControllerBase Class Reference

[interface] Abstract base class to encapsulate a local controller. **If not specified, method is not multi-thread safe.** See Controller Concepts.

Inheritance diagram for ControllerBase:



Collaboration diagram for ControllerBase:



Public Member Functions

- virtual bool Init (RobotBasePtr robot, const std::string &args) RAVE_-DEPRECATED

  Initializes the controller.

- virtual bool Init (RobotBasePtr robot, const std::vector< int > &dofindices, int nControlTransformation)=0

  initializes the controller and specifies the controlled dof

- virtual const std::vector< int > & GetControlDOFIndices () const =0

  returns the dof indices controlled

- virtual int IsControlTransformation () const =0

  returns non-zero value if base affine transformation is controlled.

- virtual void Reset (int options)=0

  Resets the current controller trajectories and any other state associated with the robot.

- virtual bool SetDesired (const std::vector< dReal > &values, TransformConstPtr trans=TransformConstPtr())=0

  go to a specific position in configuration space. ***[multi-thread safe]***

- virtual bool SetPath (TrajectoryBaseConstPtr ptraj)=0

  Follow a path in configuration space, adds to the queue of trajectories already in execution. ***[multi-thread safe]***

- virtual void SimulationStep (dReal fTimeElapsed)=0

Simulate one step forward for controllers running in the simulation environment.

- virtual bool IsDone ()=0

  Return true when goal reached.

- virtual dReal GetTime () const OPENRAVE_DUMMY_IMPLEMENTATION

  return the time along the current command

- virtual void GetVelocity (std::vector< dReal > &vel) const OPENRAVE_-DUMMY_IMPLEMENTATION

  get velocity of the controlled DOFs

- virtual void GetTorque (std::vector< dReal > &torque) const OPENRAVE_-DUMMY_IMPLEMENTATION

Static Public Member Functions

- static InterfaceType GetInterfaceTypeStatic ()

  return the static interface type this class points to (used for safe casting)

11.10.1 Detailed Description

[interface] Abstract base class to encapsulate a local controller. **If not specified, method is not multi-thread safe.** See Controller Concepts.

Examples:

customreader.cpp.

11.10.2 Member Function Documentation

11.10.2.1 virtual void GetTorque (std::vector< dReal > & *torque*) const `[virtual]`

get torque/current/strain values

Parameters

torque [out] - returns the current torque/current/strain exerted by each of the dofs from outside forces. The feedforward and friction terms should be subtracted out already

Reimplemented in MultiController.

11.10.2.2 virtual void GetVelocity (std::vector< dReal > & *vel*) const `[virtual]`

get velocity of the controlled DOFs

Parameters

vel [out] - current velocity of robot from the dof

11.10.2.3 virtual bool Init (RobotBasePtr *robot*, const std::vector< int > & *dofindices*, int *nControlTransformation*) `[pure virtual]`

initializes the controller and specifies the controlled dof

Parameters

robot the robot that uses the controller

dofindices the indices that controller will have exclusive access to

nControlTransformation

See also

IsControlTransformation

Returns

true on successful initialization

Implemented in MultiController.

11.10.2.4 virtual bool Init (RobotBasePtr *robot*, const std::string & *args*) `[virtual]`

Initializes the controller.

Parameters

robot the robot that uses the controller

args extra arguments that the controller takes.

Returns

true on successful initialization

Examples:

customreader.cpp.

11.10.2.5 virtual int IsControlTransformation () const [pure virtual]

returns non-zero value if base affine transformation is controlled.

Only one controller can modify translation and orientation per robot. For now, the two cannot be divided.

Examples:

customreader.cpp.

11.10.2.6 virtual bool IsDone () [pure virtual]

Return true when goal reached.

If a trajectory was set, return only when trajectory is done. If SetDesired was called, return only when robot is is at the desired location. If SendCommand sent, returns true when the command was completed by the hand.

Implemented in MultiController.

Examples:

customreader.cpp.

11.10.2.7 virtual void Reset (int *options*) [pure virtual]

Resets the current controller trajectories and any other state associated with the robot.

Parameters

options - specific options that can be used to control what to reset

Examples:

customreader.cpp.

11.10.2.8 virtual bool SetDesired (const std::vector< dReal > & *values*, TransformConstPtr *trans* = `TransformConstPtr()`) `[pure virtual]`

go to a specific position in configuration space. **[multi-thread safe]**

Parameters

values the final configuration in the control dofs

trans the transformation of the base. If not specified will use the current robot transformation. Ignored if controller does not use it

Returns

true if position operation successful.

Examples:

customreader.cpp.

11.10.2.9 virtual bool SetPath (TrajectoryBaseConstPtr *ptraj*) `[pure virtual]`

Follow a path in configuration space, adds to the queue of trajectories already in execution. **[multi-thread safe]**

Parameters

ptraj - the trajectory

Returns

true if trajectory operation successful

Examples:

customreader.cpp.

11.10.2.10 virtual void SimulationStep (dReal *fTimeElapsed*) `[pure virtual]`

Simulate one step forward for controllers running in the simulation environment.

Parameters

fTimeElapsed - time elapsed in simulation environment since last frame

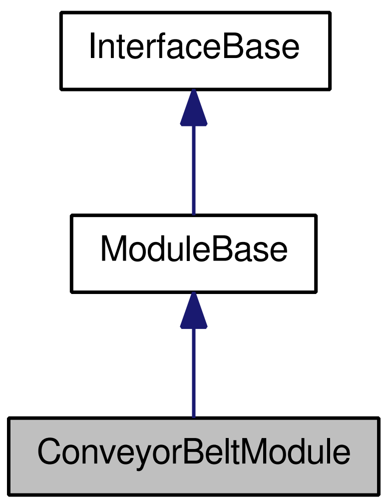
Examples:

customreader.cpp.

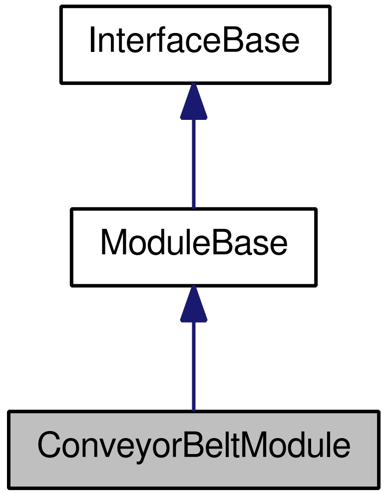
11.11 ConveyorBeltModule Class Reference

Holds a registered set of bodies, at every time step creates new bodies and moves them along a trajectory.

Inheritance diagram for ConveyorBeltModule:



Collaboration diagram for ConveyorBeltModule:



11.11.1 Detailed Description

Holds a registered set of bodies, at every time step creates new bodies and moves them along a trajectory.

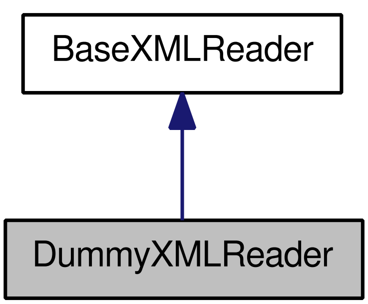
Examples:

orconveyormovement.cpp.

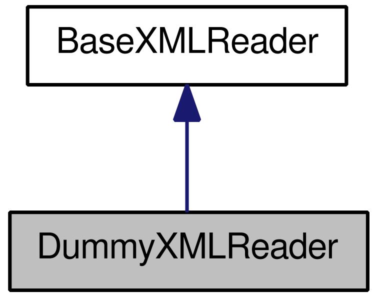
11.12 DummyXMLReader Class Reference

reads until the tag ends

Inheritance diagram for DummyXMLReader:



Collaboration diagram for DummyXMLReader:



11.12.1 Detailed Description

reads until the tag ends

11.13 EnvironmentBase Class Reference

Maintains a world state, which serves as the gateway to all functions offered through OpenRAVE. See Environment Concepts.

Public Member Functions

- virtual void Destroy ()=0

  Releases all environment resources, should be always called when environment stops being used.

- virtual void Reset ()=0

  Resets all objects of the scene (preserves all problems, planners). ***[multi-thread safe]***

- virtual void SetUserData (UserDataPtr data)

  set user data

- virtual UserDataPtr GetUserData () const

  return the user custom data

- virtual UserDataPtr GlobalState ()=0

  Returns the OpenRAVE global state, used for initializing plugins.

- virtual void OwnInterface (InterfaceBasePtr pinterface)=0

Environment will own the interface until EnvironmentBase::Destroy is called.

- virtual void DisownInterface (InterfaceBasePtr pinterface)=0

  Remove ownership of the interface.

- virtual EnvironmentBasePtr CloneSelf (int options)=0

  Create and return a clone of the current environment.

- virtual int AddModule (ModuleBasePtr module, const std::string &cmdargs)=0

  Load a new module, need to Lock if calling outside simulation thread.

- virtual bool RemoveProblem (ModuleBasePtr prob) RAVE_DEPRECATED=0
- virtual boost::shared_ptr< void > GetModules (std::list< ModuleBasePtr > &listModules) const =0

  Returns a list of loaded problems with a pointer to a lock preventing the list from being modified.

- virtual EnvironmentMutex & GetMutex () const =0

  Return the global environment mutex used to protect environment information access in multi-threaded environments.

- virtual const std::string & GetHomeDirectory () const RAVE_DEPRECATED=0

Physics and Simulation

- virtual bool SetPhysicsEngine (PhysicsEngineBasePtr physics)=0
- virtual PhysicsEngineBasePtr **GetPhysicsEngine** () const =0
- virtual void StepSimulation (dReal timeStep)=0

  Makes one simulation time step. ***[multi-thread safe]***

- virtual void StartSimulation (dReal fDeltaTime, bool bRealTime=true)=0

  Start the internal simulation thread. ***[multi-thread safe]***

- virtual void StopSimulation ()=0

  Stops the internal physics loop, stops calling SimulateStep for all modules. ***[multi-thread safe]***

- virtual bool IsSimulationRunning () const =0

  Return true if inner simulation loop is executing. ***[multi-thread safe]***

- virtual uint64_t GetSimulationTime ()=0

  Return simulation time since the start of the environment (in microseconds). ***[multi-thread safe]***

Object Setting and Querying

- virtual void AddKinBody (KinBodyPtr body, bool bAnonymous=false)=0
  Add a body to the environment.
- virtual void AddRobot (RobotBasePtr robot, bool bAnonymous=false)=0
  add a robot to the environment
- virtual void AddSensor (SensorBasePtr sensor, bool bAnonymous=false)=0
  registers the sensor with the environment and turns its power on.
- virtual void GetSensors (std::vector< SensorBasePtr > &sensors) const =0
  Fill an array with all sensors loaded in the environment. ***[multi-thread safe]***
- virtual bool RemoveKinBody (KinBodyPtr body) RAVE_DEPRECATED=0
- virtual bool Remove (InterfaceBasePtr obj)=0
  Removes a currently loaded interface from the environment. ***[multi-thread safe]***
- virtual KinBodyPtr GetKinBody (const std::string &name) const =0
  Query a body from its name. ***[multi-thread safe]***
- virtual SensorBasePtr GetSensor (const std::string &name) const =0
  Query a sensor from its name. ***[multi-thread safe]***
- virtual RobotBasePtr GetRobot (const std::string &name) const =0
  Query a robot from its name. ***[multi-thread safe]***
- virtual void GetBodies (std::vector< KinBodyPtr > &bodies) const =0
  Get all bodies loaded in the environment (including robots). ***[multi-thread safe]***
- virtual void GetRobots (std::vector< RobotBasePtr > &robots) const =0
  Fill an array with all robots loaded in the environment. ***[multi-thread safe]***
- virtual void AddViewer (ViewerBasePtr pviewer)=0
  adds a viewer to the environment
- virtual bool AttachViewer (ViewerBasePtr pnewviewer) RAVE_-DEPRECATED
- virtual ViewerBasePtr GetViewer (const std::string &name="") const =0
  Return a viewer with a particular name.
- virtual boost::shared_ptr< boost::mutex::scoped_lock > GetViewers (std::list< ViewerBasePtr > &listViewers) const =0
  Returns a list of loaded viewers with a pointer to a lock preventing the list from being modified.
- virtual void GetPublishedBodies (std::vector< KinBody::BodyState > &vbodies)=0
  Retrieve published bodies, completes even if environment is locked. ***[multi-thread safe]***

- virtual void UpdatePublishedBodies ()=0
- virtual KinBodyPtr GetBodyFromEnvironmentId (int id)=0

  Get the corresponding body from its unique network id.

- virtual void Triangulate (KinBody::Link::TRIMESH &trimesh, KinBodyConstPtr pbody)=0

  Triangulation of the body including its current transformation. trimesh will be appended the new data. ***[multi-thread safe]***

- virtual void TriangulateScene (KinBody::Link::TRIMESH &trimesh, SelectionOptions options, const std::string &selectname)=0

  General triangulation of the whole scene. ***[multi-thread safe]***

- virtual void SetDebugLevel (uint32_t level)=0

File Loading and Parsing

- enum SelectionOptions {
  SO_NoRobots = 1, TO_Obstacles = 1, SO_Robots = 2, TO_Robots = 2,
  SO_Everything = 3, TO_Everything = 3, SO_Body = 4, TO_Body = 4,
  SO_AllExceptBody = 5, TO_AllExceptBody = 5, SO_BodyList = 6 }

  A set of options used to select particular parts of the scene.

- typedef SelectionOptions **TriangulateOptions**
- virtual bool Load (const std::string &filename, const AttributesList &atts=AttributesList())=0

  Loads a scene from a file and adds all objects in the environment. ***[multi-thread safe]***

- virtual bool LoadData (const std::string &data, const AttributesList &atts=AttributesList())=0

  Loads a scene from in-memory data and adds all objects in the environment. ***[multi-thread safe]***

- virtual bool **LoadXMLData** (const std::string &data, const AttributesList &atts=AttributesList())
- virtual void Save (const std::string &filename, SelectionOptions options=SO_Everything, const std::string &selectname="")=0

  Saves a scene depending on the filename extension. Default is in COLLADA format.

- virtual RobotBasePtr ReadRobotURI (RobotBasePtr robot, const std::string &filename, const AttributesList &atts=AttributesList())=0

Initializes a robot from a resource file. The robot is not added to the environment when calling this function. ***[multi-thread safe]***

- virtual RobotBasePtr **ReadRobotXMLFile** (RobotBasePtr robot, const std::string &filename, const AttributesList &atts=AttributesList())
- virtual RobotBasePtr ReadRobotURI (const std::string &filename)

  Creates a new robot from a file with no extra load options specified. ***[multi-thread safe]***

- virtual RobotBasePtr **ReadRobotXMLFile** (const std::string &filename)
- virtual RobotBasePtr ReadRobotData (RobotBasePtr robot, const std::string &data, const AttributesList &atts=AttributesList())=0

  Initialize a robot from in-memory data. ***[multi-thread safe]***

- virtual RobotBasePtr **ReadRobotXMLData** (RobotBasePtr robot, const std::string &data, const AttributesList &atts=AttributesList())
- virtual KinBodyPtr ReadKinBodyURI (KinBodyPtr body, const std::string &filename, const AttributesList &atts=AttributesList())=0

  Initializes a kinematic body from a resource file. The body is not added to the environment when calling this function. ***[multi-thread safe]***

- virtual KinBodyPtr **ReadKinBodyXMLFile** (KinBodyPtr body, const std::string &filename, const AttributesList &atts=AttributesList())
- virtual KinBodyPtr ReadKinBodyURI (const std::string &filename)

  Creates a new kinbody from an XML file with no extra load options specified. ***[multi-thread safe]***

- virtual KinBodyPtr **ReadKinBodyXMLFile** (const std::string &filename)
- virtual KinBodyPtr ReadKinBodyData (KinBodyPtr body, const std::string &data, const AttributesList &atts=AttributesList())=0

  Initializes a kinematic body from in-memory data. ***[multi-thread safe]***

- virtual KinBodyPtr **ReadKinBodyXMLData** (KinBodyPtr body, const std::string &data, const AttributesList &atts=AttributesList())
- virtual InterfaceBasePtr ReadInterfaceURI (InterfaceBasePtr pinterface, InterfaceType type, const std::string &filename, const AttributesList &atts=AttributesList())=0

  Initializes an interface from a resource file. ***[multi-thread safe]***

- virtual InterfaceBasePtr **ReadInterfaceXMLFile** (InterfaceBasePtr pinterface, InterfaceType type, const std::string &filename, const AttributesList &atts=AttributesList())
- virtual InterfaceBasePtr **ReadInterfaceURI** (const std::string &filename, const AttributesList &atts=AttributesList())=0

- virtual InterfaceBasePtr **ReadInterfaceXMLFile** (const std::string &filename, const AttributesList &atts=AttributesList())
- virtual InterfaceBasePtr ReadInterfaceData (InterfaceBasePtr pinterface, InterfaceType type, const std::string &data, const AttributesList &atts=AttributesList())=0

  Initializes an interface from in-memory data. ***[multi-thread safe]***

- virtual InterfaceBasePtr **ReadInterfaceXMLData** (InterfaceBasePtr pinterface, InterfaceType type, const std::string &data, const AttributesList &atts=AttributesList())
- virtual boost::shared_ptr< KinBody::Link::TRIMESH > ReadTrimeshURI (boost::shared_ptr< KinBody::Link::TRIMESH > ptrimesh, const std::string &filename, const AttributesList &atts=AttributesList())=0

  reads in the rigid geometry of a resource file into a TRIMESH structure

- virtual boost::shared_ptr< KinBody::Link::TRIMESH > **ReadTrimeshFile** (boost::shared_ptr< KinBody::Link::TRIMESH > ptrimesh, const std::string &filename, const AttributesList &atts=AttributesList())
- virtual UserDataPtr RegisterXMLReader (InterfaceType type, const std::string &xmltag, const CreateXMLReaderFn &fn) RAVE_DEPRECATED=0
- virtual bool ParseXMLFile (BaseXMLReaderPtr preader, const std::string &filename) RAVE_DEPRECATED=0

  Parses a file for OpenRAVE XML formatted data.

- virtual bool ParseXMLData (BaseXMLReaderPtr preader, const std::string &data) RAVE_DEPRECATED=0

  Parses a data file for XML data.

Collision specific functions.

Each function takes an optional pointer to a CollisionReport structure and returns true if collision occurs. **[multi-thread safe]**

- typedef boost::function< CollisionAction(CollisionReportPtr, bool)> **CollisionCallbackFn**
- virtual bool SetCollisionChecker (CollisionCheckerBasePtr pchecker)=0

  set the global environment collision checker

- virtual CollisionCheckerBasePtr **GetCollisionChecker** () const =0
- virtual bool CheckCollision (KinBodyConstPtr pbody1, CollisionReportPtr report=CollisionReportPtr())=0

- virtual bool CheckCollision (KinBodyConstPtr pbody1, KinBodyConstPtr pbody2, CollisionReportPtr report=CollisionReportPtr())=0
- virtual bool CheckCollision (KinBody::LinkConstPtr plink, CollisionReportPtr report=CollisionReportPtr())=0
- virtual bool CheckCollision (KinBody::LinkConstPtr plink1, KinBody::LinkConstPtr plink2, CollisionReportPtr report=CollisionReportPtr())=0
- virtual bool CheckCollision (KinBody::LinkConstPtr plink, KinBodyConstPtr pbody, CollisionReportPtr report=CollisionReportPtr())=0
- virtual bool CheckCollision (KinBody::LinkConstPtr plink, const std::vector< KinBodyConstPtr > &vbodyexcluded, const std::vector< KinBody::LinkConstPtr > &vlinkexcluded, CollisionReportPtr report=CollisionReportPtr())=0
- virtual bool CheckCollision (KinBodyConstPtr pbody, const std::vector< KinBodyConstPtr > &vbodyexcluded, const std::vector< KinBody::LinkConstPtr > &vlinkexcluded, CollisionReportPtr report=CollisionReportPtr())=0
- virtual bool CheckCollision (const RAY &ray, KinBody::LinkConstPtr plink, CollisionReportPtr report=CollisionReportPtr())=0
- virtual bool CheckCollision (const RAY &ray, KinBodyConstPtr pbody, CollisionReportPtr report=CollisionReportPtr())=0
- virtual bool CheckCollision (const RAY &ray, CollisionReportPtr report=CollisionReportPtr())=0
- virtual bool CheckSelfCollision (KinBodyConstPtr pbody, CollisionReportPtr report=CollisionReportPtr())=0
- virtual boost::shared_ptr< void > RegisterCollisionCallback (const CollisionCallbackFn &callback)=0
- virtual bool **HasRegisteredCollisionCallbacks** () const =0
- virtual void **GetRegisteredCollisionCallbacks** (std::list< CollisionCallbackFn > &) const =0

3D plotting methods.

- typedef OpenRAVE::GraphHandlePtr GraphHandlePtr RAVE_DEPRECATED
- virtual OpenRAVE::GraphHandlePtr plot3 (const float ∗ppoints, int numPoints, int stride, float fPointSize, const RaveVector< float > &color=RaveVector< float >(1, 0.5, 0.5, 1), int drawstyle=0)=0

  Plot a point cloud with one color. ***[multi-thread safe]***

- virtual OpenRAVE::GraphHandlePtr plot3 (const float ∗ppoints, int numPoints, int stride, float fPointSize, const float ∗colors, int drawstyle=0, bool bhasalpha=false)=0

  . Plots 3D points with individual colors. ***[multi-thread safe]***

- virtual OpenRAVE::GraphHandlePtr drawlinestrip (const float *ppoints, int numPoints, int stride, float fwidth, const RaveVector< float > &color=RaveVector< float >(1, 0.5, 0.5, 1))=0

  Draws a series of connected lines with one color. ***[multi-thread safe]***

- virtual OpenRAVE::GraphHandlePtr drawlinestrip (const float *ppoints, int numPoints, int stride, float fwidth, const float *colors)=0

  Draws a series of connected lines with individual colors. ***[multi-thread safe]***

- virtual OpenRAVE::GraphHandlePtr drawlinelist (const float *ppoints, int numPoints, int stride, float fwidth, const RaveVector< float > &color=RaveVector< float >(1, 0.5, 0.5, 1))=0

  Draws a list of individual lines, each specified by a succeeding pair of points. ***[multi-thread safe]***

- virtual OpenRAVE::GraphHandlePtr drawlinelist (const float *ppoints, int numPoints, int stride, float fwidth, const float *colors)=0

  Draws a list of individual lines, each specified by a succeeding pair of points. ***[multi-thread safe]***

- virtual OpenRAVE::GraphHandlePtr drawarrow (const RaveVector< float > &p1, const RaveVector< float > &p2, float fwidth, const RaveVector< float > &color=RaveVector< float >(1, 0.5, 0.5, 1))=0

  Draws an arrow p1 is start, p2 is finish. ***[multi-thread safe]***

- virtual OpenRAVE::GraphHandlePtr drawbox (const RaveVector< float > &vpos, const RaveVector< float > &vextents)=0

  Draws a box. ***[multi-thread safe]***

- virtual OpenRAVE::GraphHandlePtr drawplane (const RaveTransform< float > &tplane, const RaveVector< float > &vextents, const boost::multi_array< float, 3 > &vtexture)=0

  Draws a textured plane. ***[multi-thread safe]***

- virtual OpenRAVE::GraphHandlePtr **drawtrimesh** (const float *ppoints, int stride, const int *pIndices, int numTriangles, const RaveVector< float > &color)=0
- virtual OpenRAVE::GraphHandlePtr drawtrimesh (const float *ppoints, int stride, const int *pIndices, int numTriangles, const boost::multi_array< float, 2 > &colors)=0

  Draws a triangle mesh, each vertices of each triangle should be counter-clockwise. ***[multi-thread safe]***

11.13.1 Detailed Description

Maintains a world state, which serves as the gateway to all functions offered through OpenRAVE. See Environment Concepts.

11.13.2 Member Typedef Documentation

11.13.2.1 typedef OpenRAVE::GraphHandlePtr GraphHandlePtr RAVE_DEPRECATED

11.13.3 Member Enumeration Documentation

11.13.3.1 enum SelectionOptions

A set of options used to select particular parts of the scene.

Enumerator:

SO_NoRobots everything but robots

TO_Obstacles everything but robots

SO_Robots all robots

TO_Robots all robots

SO_Everything all bodies and robots everything

TO_Everything all bodies and robots everything

SO_Body only triangulate robot/kinbody

TO_Body only triangulate robot/kinbody

SO_AllExceptBody select everything but the robot/kinbody

TO_AllExceptBody select everything but the robot/kinbody

SO_BodyList provide a list of body names

11.13.4 Member Function Documentation

11.13.4.1 virtual void AddKinBody (KinBodyPtr *body*, bool *bAnonymous* = `false`) `[pure virtual]`

Add a body to the environment.

Parameters

← ***body*** the pointer to an initialized body

← ***bAnonymous*** if true and there exists a body/robot with the same name, will make body's name unique

Exceptions

openrave_exception Throw if body is invalid or already added

11.13.4.2 virtual void AddRobot (RobotBasePtr *robot*, bool *bAnonymous* = false) [pure virtual]

add a robot to the environment

Parameters

← ***robot*** the pointer to an initialized robot

← ***bAnonymous*** if true and there exists a body/robot with the same name, will make robot's name unique

Exceptions

openrave_exception Throw if robot is invalid or already added

11.13.4.3 virtual void AddSensor (SensorBasePtr *sensor*, bool *bAnonymous* = false) [pure virtual]

registers the sensor with the environment and turns its power on.

Parameters

← ***sensor*** the pointer to an initialized sensor

← ***bAnonymous*** if true and there exists a sensor with the same name, will make sensor's name unique

Exceptions

openrave_exception Throw if sensor is invalid or already added

11.13.4.4 virtual void AddViewer (ViewerBasePtr *pviewer*) [pure virtual]

adds a viewer to the environment

Exceptions

openrave_exception Throw if body is invalid or already added

11.13.4.5 virtual bool AttachViewer (ViewerBasePtr *pnewviewer*) [virtual]

11.13.4.6 virtual bool CheckCollision (const RAY & *ray*, CollisionReportPtr *report* = CollisionReportPtr()) [pure virtual]

See also

CollisionCheckerBase::CheckCollision(const RAY&,CollisionReportPtr)

11.13.4.7 virtual bool CheckCollision (const RAY & *ray*, KinBodyConstPtr *pbody*, CollisionReportPtr *report* = CollisionReportPtr()) [pure virtual]

See also

CollisionCheckerBase::CheckCollision(const RAY&,KinBodyConstPtr,CollisionReportPtr)

11.13.4.8 virtual bool CheckCollision (const RAY & *ray*, KinBody::LinkConstPtr *plink*, CollisionReportPtr *report* = CollisionReportPtr()) [pure virtual]

See also

CollisionCheckerBase::CheckCollision(const RAY&,KinBody::LinkConstPtr,CollisionReportPtr)

11.13.4.9 virtual bool CheckCollision (KinBodyConstPtr *pbody*, const std::vector< KinBodyConstPtr > & *vbodyexcluded*, const std::vector< KinBody::LinkConstPtr > & *vlinkexcluded*, CollisionReportPtr *report* = `CollisionReportPtr()`) `[pure virtual]`

See also

CollisionCheckerBase::CheckCollision(KinBodyConstPtr,const std::vector<KinBodyConstPtr>&,const std::vector<KinBody::LinkConstPtr>&,CollisionReportPtr)

11.13.4.10 virtual bool CheckCollision (KinBody::LinkConstPtr *plink*, const std::vector< KinBodyConstPtr > & *vbodyexcluded*, const std::vector< KinBody::LinkConstPtr > & *vlinkexcluded*, CollisionReportPtr *report* = `CollisionReportPtr()`) `[pure virtual]`

See also

CollisionCheckerBase::CheckCollision(KinBody::LinkConstPtr,const std::vector<KinBodyConstPtr>&,const std::vector<KinBody::LinkConstPtr>&,CollisionReportPtr)

11.13.4.11 virtual bool CheckCollision (KinBody::LinkConstPtr *plink*, KinBodyConstPtr *pbody*, CollisionReportPtr *report* = `CollisionReportPtr()`) `[pure virtual]`

See also

CollisionCheckerBase::CheckCollision(KinBody::LinkConstPtr,KinBodyConstPtr,CollisionReportPtr)

11.13.4.12 virtual bool CheckCollision (KinBody::LinkConstPtr *plink1*, KinBody::LinkConstPtr *plink2*, CollisionReportPtr *report* = `CollisionReportPtr()`) `[pure virtual]`

See also

CollisionCheckerBase::CheckCollision(KinBody::LinkConstPtr,KinBody::LinkConstPtr,CollisionReportPtr)

11.13.4.13 virtual bool CheckCollision (KinBody::LinkConstPtr *plink*, CollisionReportPtr *report* = `CollisionReportPtr()`) `[pure virtual]`

See also

CollisionCheckerBase::CheckCollision(KinBody::LinkConstPtr,CollisionReportPtr)

11.13.4.14 virtual bool CheckCollision (KinBodyConstPtr *pbody1*, KinBodyConstPtr *pbody2*, CollisionReportPtr *report* = `CollisionReportPtr()`) `[pure virtual]`

See also

CollisionCheckerBase::CheckCollision(KinBodyConstPtr,KinBodyConstPtr,CollisionReportPtr)

11.13.4.15 virtual bool CheckCollision (KinBodyConstPtr *pbody1*, CollisionReportPtr *report* = `CollisionReportPtr()`) `[pure virtual]`

See also

CollisionCheckerBase::CheckCollision(KinBodyConstPtr,CollisionReportPtr)

11.13.4.16 virtual bool CheckSelfCollision (KinBodyConstPtr *pbody*, CollisionReportPtr *report* = `CollisionReportPtr()`) `[pure virtual]`

See also

CollisionCheckerBase::CheckSelfCollision

11.13.4.17 virtual EnvironmentBasePtr CloneSelf (int *options*) `[pure virtual]`

Create and return a clone of the current environment.

Clones do not share any memory or resource between each other. or their parent making them ideal for performing separte planning experiments while keeping the parent environment unchanged. By default a clone only copies the collision checkers and physics engine. When bodies are cloned, the unique ids are preserved across environments (each body can be referenced with its id in both environments). The attached and grabbed bodies of each body/robot are also copied to the new environment.

Parameters

options A set of CloningOptions describing what is actually cloned.

Returns

An environment of the same type as this environment containing the copied information.

11.13.4.18 virtual void Destroy () `[pure virtual]`

Releases all environment resources, should be always called when environment stops being used.

Removing all environment pointer might not be enough to destroy the environment resources.

11.13.4.19 virtual OpenRAVE::GraphHandlePtr drawarrow (const RaveVector< float > & *p1*, const RaveVector< float > & *p2*, float *fwidth*, const RaveVector< float > & *color* = `RaveVector< float >(1, 0.5, 0.5, 1)) [pure virtual]`

Draws an arrow p1 is start, p2 is finish. **[multi-thread safe]**

Parameters

color the rgb color of the point. The last component of the color is used for alpha blending.

Returns

handle to plotted points, graph is removed when handle is destroyed (goes out of scope). This requires the user to always store the handle in a persistent variable if the plotted graphics are to remain on the viewer.

11.13.4.20 virtual OpenRAVE::GraphHandlePtr drawbox (const RaveVector< float > & *vpos*, const RaveVector< float > & *vextents*) [pure virtual]

Draws a box. **[multi-thread safe]**

extents are half the width, height, and depth of the box

Returns

handle to plotted points, graph is removed when handle is destroyed (goes out of scope). This requires the user to always store the handle in a persistent variable if the plotted graphics are to remain on the viewer.

11.13.4.21 virtual OpenRAVE::GraphHandlePtr drawlinelist (const float * *ppoints*, int *numPoints*, int *stride*, float *fwidth*, const float * *colors*) [pure virtual]

Draws a list of individual lines, each specified by a succeeding pair of points. **[multi-thread safe]**

Parameters

stride stride in bytes to next point, ie: nextpoint = (float*)((char*)ppoints+stride)

Returns

handle to plotted points, graph is removed when handle is destroyed (goes out of scope). This requires the user to always store the handle in a persistent variable if the plotted graphics are to remain on the viewer.

11.13.4.22 virtual OpenRAVE::GraphHandlePtr drawlinelist (const float * *ppoints*, int *numPoints*, int *stride*, float *fwidth*, const RaveVector< float > & *color* = RaveVector< float >(1, 0.5, 0.5, 1)) [pure virtual]

Draws a list of individual lines, each specified by a succeeding pair of points. **[multi-thread safe]**

Parameters

stride stride in bytes to next point, ie: nextpoint = (float*)((char*)ppoints+stride)

color the rgb color of the point. The last component of the color is used for alpha blending.

Returns

handle to plotted points, graph is removed when handle is destroyed (goes out of scope). This requires the user to always store the handle in a persistent variable if the plotted graphics are to remain on the viewer.

11.13.4.23 virtual OpenRAVE::GraphHandlePtr drawlinestrip (const float * *ppoints*, int *numPoints*, int *stride*, float *fwidth*, const float * *colors*) `[pure virtual]`

Draws a series of connected lines with individual colors. **[multi-thread safe]**

Parameters

stride stride in bytes to next point, ie: nextpoint = (float*)((char*)ppoints+stride)

Returns

handle to plotted points, graph is removed when handle is destroyed (goes out of scope). This requires the user to always store the handle in a persistent variable if the plotted graphics are to remain on the viewer.

11.13.4.24 virtual OpenRAVE::GraphHandlePtr drawlinestrip (const float * *ppoints*, int *numPoints*, int *stride*, float *fwidth*, const RaveVector< float > & *color* = `RaveVector< float >(1, 0.5, 0.5, 1)`) `[pure virtual]`

Draws a series of connected lines with one color. **[multi-thread safe]**

Parameters

stride stride in bytes to next point, ie: nextpoint = (float*)((char*)ppoints+stride)

color the rgb color of the point. The last component of the color is used for alpha blending

Returns

handle to plotted points, graph is removed when handle is destroyed (goes out of scope). This requires the user to always store the handle in a persistent variable if the plotted graphics are to remain on the viewer.

11.13.4.25 virtual OpenRAVE::GraphHandlePtr drawplane (const RaveTransform< float > & *tplane*, const RaveVector< float > & *vextents*, const boost::multi_array< float, 3 > & *vtexture*) [pure virtual]

Draws a textured plane. **[multi-thread safe]**

Parameters

tplane describes the center of the plane. the zaxis of this coordinate is the normal of the plane

vextents the extents of the plane along the x and y directions (z is ignored)

vtexture a 3D array specifying height x width x color (the color dimension can be 1, 3, or 4 (for alpha blending))

Returns

handle to plotted points, graph is removed when handle is destroyed (goes out of scope). This requires the user to always store the handle in a persistent variable if the plotted graphics are to remain on the viewer.

11.13.4.26 virtual OpenRAVE::GraphHandlePtr drawtrimesh (const float ∗ *ppoints*, int *stride*, const int ∗ *pIndices*, int *numTriangles*, const boost::multi_array< float, 2 > & *colors*) [pure virtual]

Draws a triangle mesh, each vertices of each triangle should be counter-clockwise. **[multi-thread safe]**

Parameters

ppoints - array of 3D points

stride stride in bytes to next point, ie: nextpoint = (float∗)((char∗)ppoints+stride)

pIndices If not NULL, zero-based indices into the points for every triangle. pIndices should be of size numTriangles. If pIndices is NULL, ppoints is assumed to contain numTriangles∗3 points and triangles will be rendered in list order.

color The color of the triangle. The last component of the color is used for alpha blending

Returns

handle to plotted points, graph is removed when handle is destroyed (goes out of scope). This requires the user to always store the handle in a persistent variable if the plotted graphics are to remain on the viewer.

11.13.4.27 virtual void GetBodies (std::vector< KinBodyPtr > & *bodies*) const [pure virtual]

Get all bodies loaded in the environment (including robots). **[multi-thread safe]**

Parameters

→ ***bodies*** filled with all the bodies

11.13.4.28 virtual const std::string& GetHomeDirectory () const [pure virtual]

11.13.4.29 virtual KinBodyPtr GetKinBody (const std::string & *name*) const [pure virtual]

Query a body from its name. **[multi-thread safe]**

Returns

first KinBody (including robots) that matches with name

11.13.4.30 virtual boost::shared_ptr<void> GetModules (std::list< ModuleBasePtr > & *listModules*) const [pure virtual]

Returns a list of loaded problems with a pointer to a lock preventing the list from being modified.

As long as the lock is held, the problems are guaranteed to stay loaded in the environment.

Returns

returns a pointer to a Lock. Destroying the shared_ptr will release the lock

11.13.4.31 virtual EnvironmentMutex& GetMutex () const `[pure virtual]`

Return the global environment mutex used to protect environment information access in multi-threaded environments.

Accessing environment body information and adding/removing bodies or changing any type of scene property should have the environment lock acquired. Once the environment is locked, the user is guaranteed that nnothing will change in the environment.

11.13.4.32 virtual void GetPublishedBodies (std::vector< KinBody::BodyState > & *vbodies*) `[pure virtual]`

Retrieve published bodies, completes even if environment is locked. **[multi-thread safe]**

Note that the pbody pointer might become invalid as soon as GetPublishedBodies returns.

11.13.4.33 virtual RobotBasePtr GetRobot (const std::string & *name*) const `[pure virtual]`

Query a robot from its name. **[multi-thread safe]**

Returns

first Robot that matches the name

11.13.4.34 virtual SensorBasePtr GetSensor (const std::string & *name*) const `[pure virtual]`

Query a sensor from its name. **[multi-thread safe]**

Returns

first sensor that matches with name, note that sensors attached to robots have the robot name as a prefix.

11.13.4.35 virtual void GetSensors (std::vector< SensorBasePtr > & *sensors*) const [pure virtual]

Fill an array with all sensors loaded in the environment. **[multi-thread safe]**

The sensors come from the currently loaded robots and the explicitly added sensors

11.13.4.36 virtual uint64_t GetSimulationTime () [pure virtual]

Return simulation time since the start of the environment (in microseconds). **[multi-thread safe]**

See Simulation Thread for more about the simulation thread.

11.13.4.37 virtual ViewerBasePtr GetViewer (const std::string & *name* = "") const [pure virtual]

Return a viewer with a particular name.

When no name is specified, the first loaded viewer is returned.

11.13.4.38 virtual boost::shared_ptr<boost::mutex::scoped_lock> GetViewers (std::list< ViewerBasePtr > & *listViewers*) const [pure virtual]

Returns a list of loaded viewers with a pointer to a lock preventing the list from being modified.

As long as the lock is held, the problems are guaranteed to stay loaded in the environment.

Returns

returns a pointer to a Lock. Destroying the shared_ptr will release the lock

11.13.4.39 virtual bool IsSimulationRunning () const [pure virtual]

Return true if inner simulation loop is executing. **[multi-thread safe]**

See Simulation Thread for more about the simulation thread.

11.13.4.40 virtual bool ParseXMLData (BaseXMLReaderPtr *preader*, const std::string & *data*) `[pure virtual]`

Parses a data file for XML data.

Parameters

pdata The data of the buffer

len the number of bytes valid in pdata

11.13.4.41 virtual OpenRAVE::GraphHandlePtr plot3 (const float ∗ *ppoints*, int *numPoints*, int *stride*, float *fPointSize*, const float ∗ *colors*, int *drawstyle* = `0`, bool *bhasalpha* = `false`) `[pure virtual]`

. Plots 3D points with individual colors. **[multi-thread safe]**

Arguments same as plot3 with one color, except has an individual color for every point

Parameters

colors An array of rgb colors of size numPoints where each channel is in [0,1]. colors+(bhasalpha?4:3) points to the second color.

stride stride in bytes to next point, ie: nextpoint = (float∗)((char∗)ppoints+stride)

drawstyle if 0 will draw pixels. if 1, will draw 3D spherse

bhasalpha if true, then each color consists of 4 values with the last value being the alpha of the point (1 means opaque). If false, then colors is 3 values.

Returns

handle to plotted points, graph is removed when handle is destroyed (goes out of scope). This requires the user to always store the handle in a persistent variable if the plotted graphics are to remain on the viewer.

11.13.4.42 virtual OpenRAVE::GraphHandlePtr plot3 (const float ∗ *ppoints*, int *numPoints*, int *stride*, float *fPointSize*, const RaveVector< float > & *color* = `RaveVector< float >(1, 0.5, 0.5, 1)`, int *drawstyle* = `0`) `[pure virtual]`

Plot a point cloud with one color. **[multi-thread safe]**

Parameters

- ***ppoints*** array of points
- ***numPoints*** number of points to plot
- ***stride*** stride in bytes to next point, ie: nextpoint = (float∗)((char∗)ppoints+stride)
- ***fPointSize*** size of a point in pixels
- ***color*** the rgb color of the point. The last component of the color is used for alpha blending
- ***drawstyle*** if 0 will draw pixels. if 1, will draw 3D spheres

Returns

handle to plotted points, graph is removed when handle is destroyed (goes out of scope). This requires the user to always store the handle in a persistent variable if the plotted graphics are to remain on the viewer.

11.13.4.43 virtual InterfaceBasePtr ReadInterfaceData (InterfaceBasePtr *pinterface*, InterfaceType *type*, const std::string & *data*, const AttributesList & *atts* = `AttributesList()`) `[pure virtual]`

Initializes an interface from in-memory data. **[multi-thread safe]**

Parameters

- ***pinterface*** If a null pointer is passed, a new interface will be created, otherwise an existing interface will be filled
- ***data*** string containing data
- ***atts*** The attribute/value pair specifying loading options. See the individual interface descriptions at Base Interface Concepts.

11.13.4.44 virtual InterfaceBasePtr ReadInterfaceURI (InterfaceBasePtr *pinterface*, InterfaceType *type*, const std::string & *filename*, const AttributesList & *atts* = `AttributesList()`) `[pure virtual]`

Initializes an interface from a resource file. **[multi-thread safe]**

Parameters

pinterface If a null pointer is passed, a new interface will be created, otherwise an existing interface will be filled

filename the name of the resource file, its extension determines the format of the file. See Resource File Formats.

atts The attribute/value pair specifying loading options. See the individual interface descriptions at Base Interface Concepts.

11.13.4.45 virtual KinBodyPtr ReadKinBodyData (KinBodyPtr *body*, const std::string & *data*, const AttributesList & *atts* = `AttributesList()`) `[pure virtual]`

Initializes a kinematic body from in-memory data. **[multi-thread safe]**

The body should not be added to the environment when calling this function.

Parameters

body If a null pointer is passed, a new body will be created, otherwise an existing robot will be filled

atts The attribute/value pair specifying loading options. Defined in Kinematics Body Concepts.

11.13.4.46 virtual KinBodyPtr ReadKinBodyURI (KinBodyPtr *body*, const std::string & *filename*, const AttributesList & *atts* = `AttributesList()`) `[pure virtual]`

Initializes a kinematic body from a resource file. The body is not added to the environment when calling this function. **[multi-thread safe]**

Parameters

filename the name of the resource file, its extension determines the format of the file. See Resource File Formats.

body If a null pointer is passed, a new body will be created, otherwise an existing robot will be filled

atts The attribute/value pair specifying loading options. Defined in Kinematics Body Concepts.

11.13.4.47 virtual RobotBasePtr ReadRobotData (RobotBasePtr *robot*, const std::string & *data*, const AttributesList & *atts* = `AttributesList()`) `[pure virtual]`

Initialize a robot from in-memory data. **[multi-thread safe]**

The robot should not be added the environment when calling this function.

Parameters

- ***robot*** If a null pointer is passed, a new robot will be created, otherwise an existing robot will be filled
- ***atts*** The attribute/value pair specifying loading options. Defined in Robot Concepts.

11.13.4.48 virtual RobotBasePtr ReadRobotURI (RobotBasePtr *robot*, const std::string & *filename*, const AttributesList & *atts* = `AttributesList()`) `[pure virtual]`

Initializes a robot from a resource file. The robot is not added to the environment when calling this function. **[multi-thread safe]**

Parameters

- ***robot*** If a null pointer is passed, a new robot will be created, otherwise an existing robot will be filled
- ***filename*** the name of the resource file, its extension determines the format of the file. See Resource File Formats.
- ***atts*** The attribute/value pair specifying loading options. Defined in Robot Concepts.

11.13.4.49 virtual boost::shared_ptr<KinBody::Link::TRIMESH> ReadTrimeshURI (boost::shared_ptr< KinBody::Link::TRIMESH > *ptrimesh*, const std::string & *filename*, const AttributesList & *atts* = `AttributesList()`) `[pure virtual]`

reads in the rigid geometry of a resource file into a TRIMESH structure

Parameters

filename the name of the resource file, its extension determines the format of the file. Complex meshes and articulated meshes are all triangulated appropriately. See Resource File Formats.

options Options to control the parsing process.

11.13.4.50 virtual boost::shared_ptr<void> RegisterCollisionCallback (const CollisionCallbackFn & *callback*) `[pure virtual]`

Register a collision callback.

Whenever a collision is detected between between bodies during a CheckCollision call or physics simulation, the callback is called. The callback should return an action specifying how the collision should be handled: **action = callback(CollisionReport,bool IsCalledFromPhysicsEngine)**

Returns

a handle to the registration, once the handle loses scope, the callback is unregistered

11.13.4.51 virtual UserDataPtr RegisterXMLReader (InterfaceType *type*, const std::string & *xmltag*, const CreateXMLReaderFn & *fn*) `[pure virtual]`

11.13.4.52 virtual bool Remove (InterfaceBasePtr *obj*) `[pure virtual]`

Removes a currently loaded interface from the environment. **[multi-thread safe]**

The function removes currently loaded bodies, robots, sensors, problems from the actively used interfaces of the environment. This does not destroy the interface, but it does remove all references managed. Some interfaces like problems have destroy methods that are called to signal unloading. Note that the active interfaces are different from the owned interfaces.

Parameters

$\leftarrow$ ***obj*** interface to remove

Returns

true if the interface was successfully removed from the environment.

11.13.4.53 virtual bool RemoveKinBody (KinBodyPtr *body*) `[pure virtual]`

11.13.4.54 virtual bool RemoveProblem (ModuleBasePtr *prob*) `[pure virtual]`

11.13.4.55 virtual void Reset () `[pure virtual]`

Resets all objects of the scene (preserves all problems, planners). **[multi-thread safe]**

Do not call inside a SimulationStep call

11.13.4.56 virtual void Save (const std::string & *filename*, SelectionOptions *options* = `SO_Everything`, const std::string & *selectname* = `""`) `[pure virtual]`

Saves a scene depending on the filename extension. Default is in COLLADA format.

Parameters

filename the filename to save the results at

options controls what to save

selectname

Exceptions

openrave_exception Throw if failed to save anything

11.13.4.57 virtual void SetDebugLevel (uint32_t *level*) `[pure virtual]`

sets the debug level

Parameters

level 0 for no debug, 1 - to print all debug messeges. Default value for release builds is 0, for debug builds it is 1 declaring variables with stdcall can be a little complex

11.13.4.58 virtual bool SetPhysicsEngine (PhysicsEngineBasePtr *physics*) [pure virtual]

set the physics engine, disabled by default

Parameters

physics the engine to set, if NULL, environment sets an dummy physics engine

11.13.4.59 virtual void StartSimulation (dReal *fDeltaTime*, bool *bRealTime* = true) [pure virtual]

Start the internal simulation thread. **[multi-thread safe]**

Resets simulation time to 0. See Simulation Thread for more about the simulation thread.

Parameters

fDeltaTime the delta step to take in simulation

bRealTime if false will call SimulateStep as fast as possible, otherwise will time the simulate step calls so that simulation progresses with real system time.

11.13.4.60 virtual void StepSimulation (dReal *timeStep*) [pure virtual]

Makes one simulation time step. **[multi-thread safe]**

Can be called manually by the user inside planners. Keep in mind that the internal simulation thread also calls this function periodically. See Simulation Thread for more about the simulation thread.

11.13.4.61 virtual void StopSimulation () [pure virtual]

Stops the internal physics loop, stops calling SimulateStep for all modules. **[multi-thread safe]**

See Simulation Thread for more about the simulation thread.

11.13.4.62 virtual void Triangulate (KinBody::Link::TRIMESH & *trimesh*, KinBodyConstPtr *pbody*) `[pure virtual]`

Triangulation of the body including its current transformation. trimesh will be appended the new data. **[multi-thread safe]**

Parameters

→ ***trimesh*** - The output triangle mesh

← ***body*** body the triangulate

Exceptions

openrave_exception Throw if failed to add anything

11.13.4.63 virtual void TriangulateScene (KinBody::Link::TRIMESH & *trimesh*, SelectionOptions *options*, const std::string & *selectname*) `[pure virtual]`

General triangulation of the whole scene. **[multi-thread safe]**

Parameters

→ ***trimesh*** - The output triangle mesh. The new triangles are appended to the existing triangles!

← ***options*** - Controlls what to triangulate.

← ***selectname*** - name of the body used in options

Exceptions

openrave_exception Throw if failed to add anything

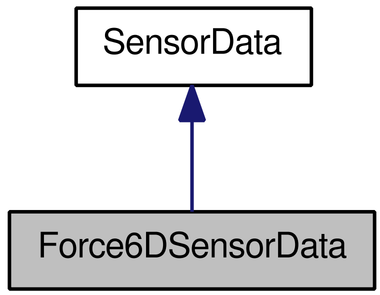
11.13.4.64 virtual void UpdatePublishedBodies () `[pure virtual]`

updates the published bodies that viewers and other programs listening in on the environment see. For example, calling this function inside a planning loop allows the viewer to update the environment reflecting the status of the planner. Assumes that the physics are locked.

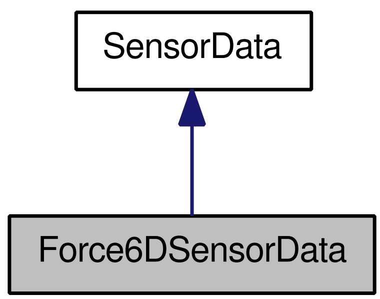
11.14 Force6DSensorData Class Reference

Stores force data.

Inheritance diagram for Force6DSensorData:



Collaboration diagram for Force6DSensorData:



Public Attributes

- Vector force

    Force in X Y Z, in newtons.

- Vector torque

    Torque in X Y Z, in newtonmeters.

11.14.1 Detailed Description

Stores force data.

11.15 Friction Struct Reference

LuGre friction model?

Public Attributes

- dReal sigma_0

  the stiffness coefficient of the contacting surfaces

- dReal sigma_1

  the friction damping coefficient.

- dReal mu_s

  static friction coefficient

- dReal mu_d

  dynamic friction coefficient

11.15.1 Detailed Description

LuGre friction model?

11.16 frustum< T > Class Template Reference

A pyramid with its vertex clipped.

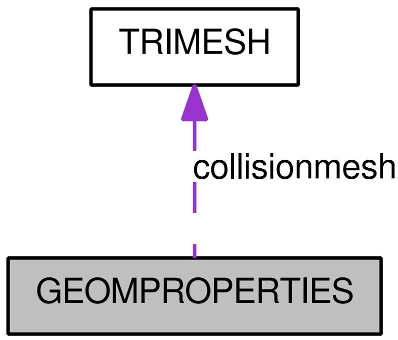
11.16.1 Detailed Description

template<typename T> class OpenRAVE::geometry::frustum< T >

A pyramid with its vertex clipped.

11.17 GEOMPROPERTIES Class Reference

Collaboration diagram for GEOMPROPERTIES:



Public Types

- enum GeomType

  The type of geometry primitive.

Public Member Functions

- const Transform & GetTransform () const

  Local transformation of the geom primitive with respect to the link's coordinate system.

- const std::string & GetRenderFilename () const

  render resource file, should be transformed by _t before rendering

- const TRIMESH & GetCollisionMesh () const

  collision data of the specific object in its local coordinate system.

- virtual AABB ComputeAABB (const Transform &trans) const

  returns an axis aligned bounding box given that the geometry is transformed by trans

- virtual void SetCollisionMesh (const TRIMESH &mesh)

  sets a new collision mesh and notifies every registered callback about it

- virtual void SetDraw (bool bDraw)

  sets a drawing and notifies every registered callback about it

- virtual void SetTransparency (float f)

  set transparency level (0 is opaque)

- virtual void SetDiffuseColor (const RaveVector< float > &color)

  override diffuse color of geometry material

- virtual void SetAmbientColor (const RaveVector< float > &color)

  override ambient color of geometry material

- virtual bool ValidateContactNormal (const Vector &position, Vector &normal) const

  validates the contact normal on the surface of the geometry and makes sure the normal faces "outside" of the shape.

- virtual void SetRenderFilename (const std::string &renderfilename)

  sets a new render filename for the geometry. This does not change the collision

Protected Member Functions

- bool InitCollisionMesh (float fTessellation=1)

Protected Attributes

- Transform _t

  see GetTransform

- Vector vGeomData
- RaveVector< float > ambientColor

  hints for how to color the meshes

- TRIMESH collisionmesh

  see GetCollisionMesh

- GeomType _type

  the type of geometry primitive

- std::string _renderfilename
- Vector vRenderScale

  render scale of the object (x,y,z)

- float ftransparency

  value from 0-1 for the transparency of the rendered object, 0 is opaque

- bool _bDraw

  if true, object is drawn as part of the 3d model (default is true)

- bool _bModifiable

  if true, object geometry can be dynamically modified (default is true)

11.17.1 Detailed Description

Describes the properties of a basic geometric primitive. Contains everything associated with a physical body along with a seprate (optional) render file.

11.17.2 Member Function Documentation

11.17.2.1 const TRIMESH& GetCollisionMesh () const

collision data of the specific object in its local coordinate system.

Should be transformed by GEOMPROPERTIES::GetTransform() before rendering. For spheres and cylinders, an appropriate discretization value is chosen.

11.17.2.2 const std::string& GetRenderFilename () const

render resource file, should be transformed by _t before rendering

If the value is "__norenderif__:x", then the viewer should not render the object if it supports ∗.x files where"x" is the file extension.

11.17.2.3 bool InitCollisionMesh (float *fTessellation* = `1`) `[protected]`

triangulates the geometry object and initializes collisionmesh. GeomTrimesh types must already be triangulated

Parameters

fTessellation to control how fine the triangles need to be. 1.0f is the default value

11.17.2.4 virtual bool ValidateContactNormal (const Vector & *position*, Vector & *normal*) const [virtual]

validates the contact normal on the surface of the geometry and makes sure the normal faces "outside" of the shape.

Parameters

position the position of the contact point specified in the link's coordinate system

normal the unit normal of the contact point specified in the link's coordinate system

Returns

true if the normal is changed to face outside of the shape

11.17.3 Member Data Documentation

11.17.3.1 std::string _renderfilename [protected]

See also

ref GetRenderFilename

11.17.3.2 Vector vGeomData [protected]

for boxes, first 3 values are extents for sphere it is radius for cylinder, first 2 values are radius and height for trimesh, none

11.18 Grabbed Class Reference

The information of a currently grabbed body.

Public Attributes

- KinBodyWeakPtr pbody

  the grabbed body

- LinkPtr plinkrobot

  robot link that is grabbing the body

- std::vector< LinkConstPtr > vNonCollidingLinks

  robot links that already collide with the body

- Transform troot

  root transform (of first link of body) relative to plinkrobot's transform. In other words, pbody->GetTransform() == plinkrobot->GetTransform()∗troot

11.18.1 Detailed Description

The information of a currently grabbed body.

11.19 GraphHandle Class Reference

Handle holding the plot from the viewers. The plot will continue to be drawn as long as a reference to this handle is held.

Public Member Functions

- virtual void SetTransform (const RaveTransform< float > &t) OPENRAVE_DUMMY_IMPLEMENTATION

  Changes the underlying transformation of the plot. ***[multi-thread safe]***

- virtual void SetShow (bool bshow) OPENRAVE_DUMMY_IMPLEMENTATION

  Shows or hides the plot without destroying its resources. ***[multi-thread safe]***

11.19.1 Detailed Description

Handle holding the plot from the viewers. The plot will continue to be drawn as long as a reference to this handle is held. Designed to be multi-thread safe and destruction and modification of the viewer plot can be done at any time. The viewers internally handle synchronization and threading issues.

11.19.2 Member Function Documentation

11.19.2.1 virtual void SetTransform (const RaveTransform< float > & *t*) `[virtual]`

Changes the underlying transformation of the plot. **[multi-thread safe]**

Parameters

t new transformation of the plot

11.20 Group Class Reference

A group referencing the values of one body in the environment.

Public Attributes

- int offset

  For each data point, the number of values to offset before data for this group starts.

- int dof

  The number of values in this group.

- std::string name

  semantic information on what part of the environment the group refers to.

- std::string interpolation

  Describes how the data should be interpolated. Common methods are:

11.20.1 Detailed Description

A group referencing the values of one body in the environment.

11.20.2 Member Data Documentation

11.20.2.1 std::string interpolation

Describes how the data should be interpolated. Common methods are:

- **previous** - the previous waypoint's value is always chosen
- **next** - the next waypoint's value is always chosen
- **linear** - linear interpolation (default)

- **quadratic** - position is piecewise-quadratic, velocity is piecewise-linear, acceleration is one of -amax, 0, or amax
- **cubic** - 3 degree polynomial
- **quadric** - 4 degree polynomial
- **quintic** - 5 degree polynomial

11.20.2.2 std::string name

semantic information on what part of the environment the group refers to.

Can be composed of multiple workds; the first word is the group type, and the words following narrow the specifics. Common types are:

- **joint_values** - The joint values of a kinbody/robot. The joint names with the name of the body can follow.
- **joint_velocities** - The joint velocities (1/second) of a kinbody/robot. The name of the body with the joint names can follow.
- **joint_accelerations** - The joint accelerations (1/second^2) of a kinbody/robot. The name of the body with the joint names can follow.
- **joint_torques** - The joint torques (Newton meter) of a kinbody/robot. The name of the body with the joint names can follow.
- **affine_transform** - An affine transformation [quaternion, translation]. The name of the body with selected affine dofs (see DOFAffine) can follow.
- **affine_velocities** - The velocity (1/second) of the affine transformation [rotation axis, translation velocity], the name of the body can follow.
- **affine_accelerations** - The velocity (1/second^2) of the affine transformation [rotation axis, translation velocity], the name of the body can follow.
- **ikparam_values** - The values of an IkParmeterization. The ikparam type is stored as the second value in name
- **ikparam_velocities** - velociti of an IkParmeterization. The ikparam type is stored as the second value in name

11.21 IkParameterization Class Reference

Parameterization of basic primitives for querying inverse-kinematics solutions.

Public Types

- typedef IkParameterizationType Type RAVE_DEPRECATED

Public Member Functions

- IkParameterization (const Transform &t)

  sets a 6D transform parameterization

- IkParameterization (const RAY &r)

  sets a ray parameterization

- IkParameterization (const Transform &t, IkParameterizationType type)

  set a custom parameterization using a transform as the source of the data. Not all types are supported with this method.

- int GetDOF () const

  Returns the minimum degree of freedoms required for the IK type.

- int GetNumberOfValues () const

  Returns the number of values used to represent the parameterization (>= dof). The number of values serialized is this number plus 1 for the iktype.

- void SetLookat3D (const RAY &ray)

  the ray direction is not used for IK, however it is needed in order to compute the error

- dReal ComputeDistanceSqr (const IkParameterization &ikparam) const

  Computes the distance squared between two IK parmaeterizations.

- void GetValues (std::vector< dReal >::iterator itvalues) const

  fills the iterator with the serialized values of the ikparameterization.

  - void SetTransform (const Transform &t) RAVE_DEPRECATED

Static Public Member Functions

- static int GetDOF (IkParameterizationType type)

  Returns the minimum degree of freedoms required for the IK type.

- static int GetNumberOfValues (IkParameterizationType type)

  Returns the number of values used to represent the parameterization (>= dof). The number of values serialized is this number plus 1 for the iktype.

11.21.1 Detailed Description

Parameterization of basic primitives for querying inverse-kinematics solutions. Holds the parameterization of a geometric primitive useful for autonomous manipulation scenarios like: 6D pose, 3D translation, 3D rotation, 3D look at direction, and ray look at direction.

Examples:

ikfastloader.cpp, and orikfilter.cpp.

11.21.2 Member Typedef Documentation

11.21.2.1 typedef IkParameterizationType Type RAVE_DEPRECATED

11.21.3 Member Function Documentation

11.21.3.1 void GetValues (std::vector< dReal >::iterator *itvalues*) const

fills the iterator with the serialized values of the ikparameterization.

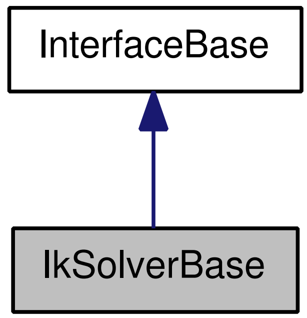
the container the iterator points to needs to have GetNumberOfValues() available.

11.21.3.2 void SetTransform (const Transform & *t*)

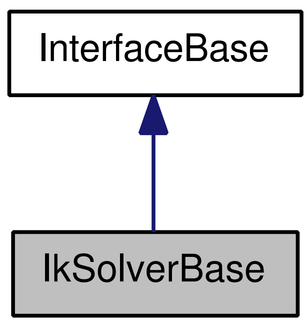
11.22 IkSolverBase Class Reference

[interface] Base class for all Inverse Kinematic solvers. **If not specified, method is not multi-thread safe.** See Inverse Kinematics Solver Concepts.

Inheritance diagram for IkSolverBase:



Collaboration diagram for IkSolverBase:



Public Types

- typedef boost::function< IkFilterReturn(std::vector< dReal > &, RobotBase::ManipulatorPtr, const IkParameterization &)> IkFilterCallbackFn

Public Member Functions

- virtual bool Init (RobotBase::ManipulatorPtr pmanip)=0
- virtual UserDataPtr RegisterCustomFilter (int priority, const IkFilterCallbackFn &filterfn)

    Sets an ik solution filter that is called for every ik solution.

- virtual void SetCustomFilter (const IkFilterCallbackFn &filterfn) RAVE_DEPRECATED
- virtual int GetNumFreeParameters () const =0

Number of free parameters defining the null solution space.

- virtual bool GetFreeParameters (std::vector< dReal > &vFreeParameters) const =0

  gets the free parameters from the current robot configuration

- virtual bool Solve (const IkParameterization &param, const std::vector< dReal > &q0, int filteroptions, boost::shared_ptr< std::vector< dReal > > solution)=0

  Return a joint configuration for the given end effector transform.

- virtual bool Solve (const IkParameterization &param, int filteroptions, std::vector< std::vector< dReal > > &solutions)=0

  Return all joint configurations for the given end effector transform.

- virtual bool Solve (const IkParameterization &param, const std::vector< dReal > &q0, const std::vector< dReal > &vFreeParameters, int filteroptions, boost::shared_ptr< std::vector< dReal > > solution)=0
- virtual bool Solve (const IkParameterization &param, const std::vector< dReal > &vFreeParameters, int filteroptions, std::vector< std::vector< dReal > > &solutions)=0

  Return all joint configurations for the given end effector transform.

- virtual bool Supports (IkParameterizationType iktype) const OPENRAVE_DUMMY_IMPLEMENTATION

  returns true if the solver supports a particular ik parameterization as input.

Static Public Member Functions

- static InterfaceType GetInterfaceTypeStatic ()

  return the static interface type this class points to (used for safe casting)

Protected Member Functions

- virtual IkFilterReturn _CallFilters (std::vector< dReal > &solution, RobotBase::ManipulatorPtr manipulator, const IkParameterization &param)

  calls the registered filters in their priority order and returns the value of the last called filter.

11.22.1 Detailed Description

[interface] Base class for all Inverse Kinematic solvers. **If not specified, method is not multi-thread safe.** See Inverse Kinematics Solver Concepts.

11.22.2 Member Typedef Documentation

11.22.2.1 typedef boost::function<IkFilterReturn(std::vector<dReal>&, RobotBase::ManipulatorPtr, const IkParameterization&)> IkFilterCallbackFn

Inverse kinematics filter callback function.

The filter is of the form `return = filterfn(solution, manipulator, param)`. The solution is guaranteed to be set on the robot's joint values before this function is called. If modifying the robot state, should restore it before this function returns.

Parameters

solution The current solution of the manipulator. Can be modified by this function, but note that it will not go through previous checks again.

manipulator The current manipulator that the ik is being solved for.

param The paramterization that IK was called with. This is in the manipulator base link's coordinate system (which is not necessarily the world coordinate system).

Returns

IkFilterReturn controlling the behavior of the ik search process.

11.22.3 Member Function Documentation

11.22.3.1 virtual bool GetFreeParameters (std::vector< dReal > & *vFreeParameters*) const `[pure virtual]`

gets the free parameters from the current robot configuration

Parameters

→ ***vFreeParameters*** is filled with GetNumFreeParameters() parameters in [0,1] range

Returns

true if succeeded

11.22.3.2 virtual int GetNumFreeParameters () const `[pure virtual]`

Number of free parameters defining the null solution space.

Each parameter is always in the range of [0,1].

11.22.3.3 virtual bool Init (RobotBase::ManipulatorPtr *pmanip*) `[pure virtual]`

brief Sets the IkSolverBase attached to a specific robot and sets IkSolverBase specific options.

For example, some ik solvers might have different ways of computing optimal solutions.

Parameters

pmanip The manipulator the IK solver is attached to

11.22.3.4 virtual UserDataPtr RegisterCustomFilter (int *priority*, const IkFilterCallbackFn & *filterfn*) `[virtual]`

Sets an ik solution filter that is called for every ik solution.

Multiple filters can be set at once, each filter will be called according to its priority; higher values get called first. The default implementation of IkSolverBase manages the filters internally. Users implementing their own IkSolverBase should call _CallFilters to run the internally managed filters.

Parameters

filterfn - an optional filter function to be called, see IkFilterCallbackFn.

priority - The priority of the filter that controls the order in which filters get called. Higher priority filters get called first. If not certain what to set, use 0.

Returns

a managed handle to the filter. If this handle is released, then the fitler will be removed. Release operation is **[multi-thread safe]**.

11.22.3.5 virtual void SetCustomFilter (const IkFilterCallbackFn & *filterfn*) `[virtual]`

11.22.3.6 virtual bool Solve (const IkParameterization & *param*, const std::vector< dReal > & *vFreeParameters*, int *filteroptions*, std::vector< std::vector< dReal > > & *solutions*) `[pure virtual]`

Return all joint configurations for the given end effector transform.

Can specify the free parameters in [0,1] range. If NULL, the regular equivalent Solve is called

Parameters

← ***param*** the pose the end effector has to achieve. Note that the end effector pose takes into account the grasp coordinate frame for the RobotBase::Manipulator

← ***vFreeParameters*** The free parameters of the null space of the IK solutions. Always in range of [0,1]

← ***filteroptions*** A bitmask of IkFilterOptions values controlling what is checked for each ik solution.

→ ***solutions*** All solutions within a reasonable discretization level of the free parameters.

Returns

true at least one solution is found

11.22.3.7 virtual bool Solve (const IkParameterization & *param*, const std::vector< dReal > & *q0*, const std::vector< dReal > & *vFreeParameters*, int *filteroptions*, boost::shared_ptr< std::vector< dReal > > *solution*) `[pure virtual]`

Return a joint configuration for the given end effector transform.

Can specify the free parameters in [0,1] range. If NULL, the regular equivalent Solve is called

Parameters

← ***param*** the pose the end effector has to achieve. Note that the end effector pose takes into account the grasp coordinate frame for the RobotBase::Manipulator

← ***q0*** Return a solution nearest to the given configuration q0 in terms of the joint distance. If q0 is empty, returns the first solution found

← ***vFreeParameters*** The free parameters of the null space of the IK solutions. Always in range of [0,1]

← ***filteroptions*** A bitmask of IkFilterOptions values controlling what is checked for each ik solution.

→ ***solution*** Holds the IK solution, must be of size RobotBase::Manipulator::_-vecarmjoints

Returns

true if solution is found

11.22.3.8 virtual bool Solve (const IkParameterization & *param*, int *filteroptions*, std::vector< std::vector< dReal > > & *solutions*) `[pure virtual]`

Return all joint configurations for the given end effector transform.

Parameters

← ***param*** the pose the end effector has to achieve. Note that the end effector pose takes into account the grasp coordinate frame for the RobotBase::Manipulator

← ***filteroptions*** A bitmask of IkFilterOptions values controlling what is checked for each ik solution.

→ ***solutions*** All solutions within a reasonable discretization level of the free parameters.

Returns

true if at least one solution is found

11.22.3.9 virtual bool Solve (const IkParameterization & *param*, const std::vector< dReal > & *q0*, int *filteroptions*, boost::shared_ptr< std::vector< dReal > > *solution*) `[pure virtual]`

Return a joint configuration for the given end effector transform.

Parameters

← ***param*** the pose the end effector has to achieve. Note that the end effector pose takes into account the grasp coordinate frame for the RobotBase::Manipulator

← ***q0*** Return a solution nearest to the given configuration q0 in terms of the joint distance. If q0 is NULL, returns the first solution found

← ***filteroptions*** A bitmask of IkFilterOptions values controlling what is checked for each ik solution.

→ ***solution*** [optional] Holds the IK solution

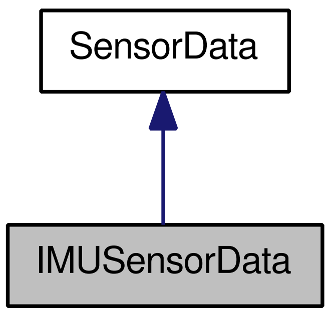
Returns

true if solution is found

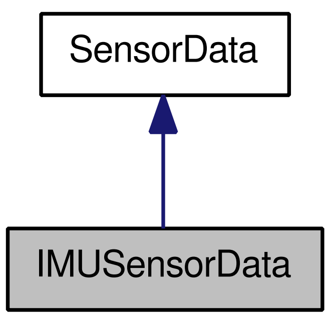
11.23 IMUSensorData Class Reference

Stores IMU data.

Inheritance diagram for IMUSensorData:



Collaboration diagram for IMUSensorData:



Public Attributes

- Vector rotation

quaternion

- boost::array< dReal, 9 > rotation_covariance

  Row major about x, y, z axes.

- boost::array< dReal, 9 > angular_velocity_covariance

  Row major about x, y, z axes.

- boost::array< dReal, 9 > linear_acceleration_covariance

  Row major x, y z axes.

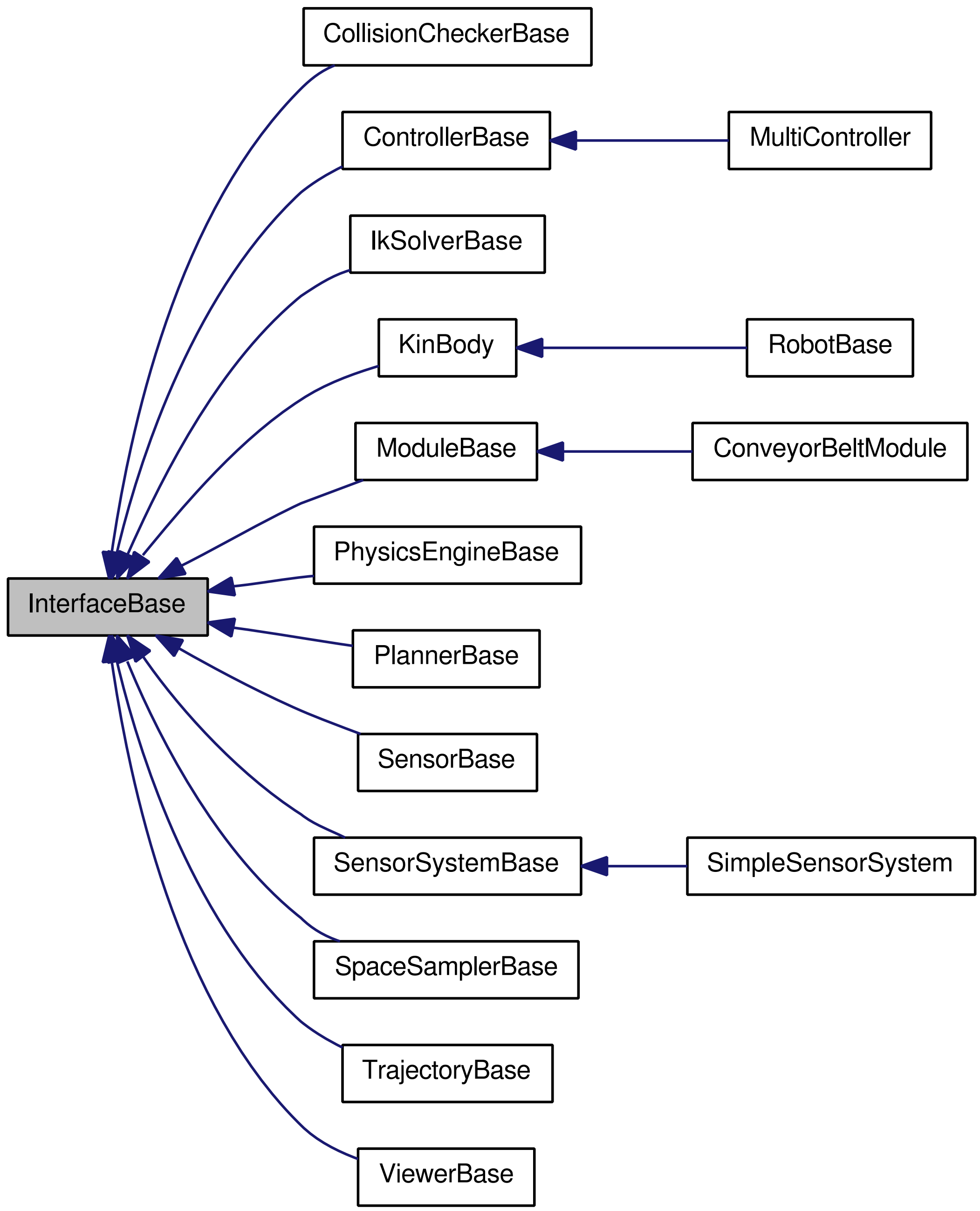
11.23.1 Detailed Description

Stores IMU data.

11.24 InterfaceBase Class Reference

[interface] Base class for all interfaces that OpenRAVE provides. See Base Interface Concepts.

Inheritance diagram for InterfaceBase:



Public Member Functions

- const std::string & GetXMLId () const
- const std::string & GetPluginName () const
- EnvironmentBasePtr GetEnv () const
- virtual const std::string & GetDescription () const

  Documentation of the interface in reStructuredText format. See Documenting Interfaces.

- virtual void SetUserData (UserDataPtr data)

  set user data

- virtual void SetUserData (boost::shared_ptr< void > data) RAVE_-DEPRECATED
- virtual UserDataPtr GetUserData () const

  return the user custom data

- virtual const std::string & GetURI () const

  the URI used to load the interface (sometimes this is not possible if the definition lies inside an environment file).

- virtual void Clone (InterfaceBaseConstPtr preference, int cloningoptions)

  Clone the contents of an interface to the current interface.

- virtual bool SendCommand (std::ostream &os, std::istream &is)

  Used to send special commands to the interface and receive output.

Protected Types

- typedef boost::function< bool(std::ostream &, std::istream &)> InterfaceCommandFn

  The function to be executed for every command.

Protected Member Functions

- virtual void RegisterCommand (const std::string &cmdname, InterfaceCommandFn fncmd, const std::string &strhelp)

  Registers a command and its help string.

- virtual void UnregisterCommand (const std::string &cmdname)

  Unregisters the command.

11.24.1 Detailed Description

[interface] Base class for all interfaces that OpenRAVE provides. See Base Interface Concepts.

11.24.2 Member Typedef Documentation

11.24.2.1 typedef boost::function<bool (std::ostream&, std::istream&)> InterfaceCommandFn `[protected]`

The function to be executed for every command.

Parameters

sinput - input of the command

sout - output of the command

Returns

If false, there was an error with the command, true if successful

11.24.3 Member Function Documentation

11.24.3.1 virtual void Clone (InterfaceBaseConstPtr *preference*, int *cloningoptions*) `[virtual]`

Clone the contents of an interface to the current interface.

Parameters

preference the interface whose information to clone

cloningoptions mask of CloningOptions

Exceptions

openrave_exception if command doesn't succeed

Reimplemented in RobotBase.

11.24.3.2 EnvironmentBasePtr GetEnv () const

Returns

the environment that this interface is attached to

Examples:

orconveyormovement.cpp, orpythonbinding.cpp, and plugincpp.cpp.

11.24.3.3 const std::string& GetPluginName () const

set internally by RaveDatabase

Returns

the pluginname this interface was loaded from

11.24.3.4 const std::string& GetXMLId () const

set internally by RaveDatabase

Returns

the unique identifier that describes this class type, case is ignored should be the same id used to create the object

11.24.3.5 virtual void RegisterCommand (const std::string & *cmdname*, InterfaceCommandFn *fncmd*, const std::string & *strhelp*) `[protected, virtual]`

Registers a command and its help string.

Parameters

cmdname - command name, converted to lower case

fncmd function to execute for the command

strhelp - help string in reStructuredText, see Documenting Interfaces.

Exceptions

openrave_exception Throw if there exists a registered command already.

Examples:

orconveyormovement.cpp, and plugincpp.cpp.

11.24.3.6 virtual bool SendCommand (std::ostream & *os*, std::istream & *is*) [virtual]

Used to send special commands to the interface and receive output.

The command must be registered by RegisterCommand. A special command '**help**' is always supported and provides a way for the user to query the current commands and the help string. The format of the returned help commands are in reStructuredText. The following commands are possible:

- '**help** [command name]' - get the help string of just that command.
- '**help** commands' - return the names of all the possible commands

Parameters

is the input stream containing the command

os the output stream containing the output

Exceptions

openrave_exception Throw if the command is not supported.

Returns

true if the command is successfully processed, otherwise false.

11.24.3.7 virtual void SetUserData (boost::shared_ptr< void > *data*) [virtual]

11.25 Joint Class Reference

Information about a joint that controls the relationship between two links.

Classes

- struct MIMIC

  Holds mimic information about position, velocity, and acceleration of one axis of the joint.

Public Types

- enum JointType

  The type of joint movement.

Public Member Functions

- const std::string & GetName () const

  The unique name of the joint.

- int GetDOFIndex () const

  Get the degree of freedom index in the body's DOF array.

- int GetJointIndex () const

  Get the joint index into KinBody::GetJoints.

- dReal GetResolution () const

  The discretization of the joint used when line-collision checking.

- virtual int GetDOF () const

  The degrees of freedom of the joint. Each joint supports a max of 3 degrees of freedom.

- virtual bool IsCircular (int iaxis) const

  Return true if joint axis has an identification at some of its lower and upper limits.

- virtual bool IsRevolute (int iaxis) const

  returns true if the axis describes a rotation around an axis.

- virtual bool IsPrismatic (int iaxis) const

  returns true if the axis describes a translation around an axis.

- virtual bool IsStatic () const

  Return true if joint can be treated as a static binding (ie all limits are 0).

- virtual void GetValues (std::vector< dReal > &values, bool bAppend=false) const

  Return all the joint values with the correct offsets applied.

- virtual dReal GetValue (int axis) const

  Return the value of the specified joint axis only.

- virtual void GetVelocities (std::vector< dReal > &values, bool bAppend=false) const

  Gets the joint velocities.

- virtual void AddTorque (const std::vector< dReal > &torques)

  Add effort (force or torque) to the joint.

- virtual Vector GetAnchor () const

  The anchor of the joint in global coordinates.

- virtual Vector GetAxis (int axis=0) const

  The axis of the joint in global coordinates.

- virtual void GetLimits (std::vector< dReal > &vLowerLimit, std::vector< dReal > &vUpperLimit, bool bAppend=false) const

  Returns the limits of the joint.

- virtual void SetLimits (const std::vector< dReal > &lower, const std::vector< dReal > &upper)
- virtual void SetJointLimits (const std::vector< dReal > &lower, const std::vector< dReal > &upper) RAVE_DEPRECATED
- virtual void GetVelocityLimits (std::vector< dReal > &vmax, bool bAppend=false) const

  Returns the max velocities of the joint.

- virtual void SetVelocityLimits (const std::vector< dReal > &vmax)
- virtual void GetAccelerationLimits (std::vector< dReal > &vmax, bool bAppend=false) const

  Returns the max accelerations of the joint.

- virtual void SetAccelerationLimits (const std::vector< dReal > &vmax)
- virtual dReal GetWeight (int axis=0) const

  The weight associated with a joint's axis for computing a distance in the robot configuration space.

- virtual void SetWeights (const std::vector< dReal > &weights)
- dReal GetWrapOffset (int iaxis=0) const

  Return internal offset parameter that determines the branch the angle centers on.

- virtual void SetWrapOffset (dReal offset, int iaxis=0)
- virtual void SetOffset (dReal offset, int iaxis=0) RAVE_DEPRECATED

Internal Hierarchy Methods

- virtual LinkPtr GetHierarchyParentLink () const

  Return the parent link which the joint measures its angle off from (either GetFirstAttached() or GetSecondAttached()).

- virtual LinkPtr GetHierarchyChildLink () const

  Return the child link whose transformation is computed by this joint's values (either GetFirstAttached() or GetSecondAttached()).

- virtual Vector GetInternalHierarchyAnchor () const RAVE_DEPRECATED
- virtual Vector GetInternalHierarchyAxis (int axis=0) const

  The axis of the joint in local coordinates.

- virtual Transform GetInternalHierarchyLeftTransform () const

  Left multiply transform given the base body.

- virtual Transform GetInternalHierarchyRightTransform () const

  Right multiply transform given the base body.

Mimic Joint Properties

A mimic joint's angles are automatically determined from other joints based on a general purpose formula. A user does not have control of the the mimic joint values, even if they appear in the DOF list.

- int GetMimicJointIndex () const RAVE_DEPRECATED
- const std::vector< dReal > GetMimicCoeffs () const RAVE_DEPRECATED
- bool IsMimic (int axis=-1) const

  Returns true if a particular axis of the joint is mimiced.

- std::string GetMimicEquation (int axis=0, int type=0, const std::string &format="") const

  If the joint is mimic, returns the equation to compute its value.

- void GetMimicDOFIndices (std::vector< int > &vmimicdofs, int axis=0) const

  Returns the set of DOF indices that the computation of a joint axis depends on. Order is arbitrary.

- void SetMimicEquations (int axis, const std::string &poseq, const std::string &veleq, const std::string &acceleq="")

  Sets the mimic properties of the joint.

Protected Member Functions

- virtual void _ComputePartialVelocities (std::vector< std::pair< int, dReal > > &vpartials, int iaxis, std::map< std::pair< MIMIC::DOFFormat, int >, dReal > &mapcachedpartials) const *computes the partial velocities with respect to all dependent DOFs specified by MIMIC::_vmimicdofs.*

- virtual void _ComputeInternalInformation (LinkPtr plink0, LinkPtr plink1, const Vector &vanchor, const std::vector< Vector > &vaxes, const std::vector< dReal > &vcurrentvalues)

  Compute internal transformations and specify the attached links of the joint.

Protected Attributes

- boost::array< Vector, 3 > _vaxes

  axes in body[0]'s or environment coordinate system used to define joint movement

- Vector vanchor

  anchor of the joint, this is only used to construct the internal left/right matrices

- dReal fResolution

  interpolation resolution

- boost::array< dReal, 3 > _vmaxvel

  the soft maximum velocity (rad/s) to move the joint when planning

- boost::array< dReal, 3 > fHardMaxVel

  the hard maximum velocity, robot cannot exceed this velocity. used for verification checking

- boost::array< dReal, 3 > _vmaxaccel

  the maximum acceleration (rad/s^2) of the joint

- boost::array< dReal, 3 > _vmaxtorque

  maximum torque (N.m, kg m^2/s^2) that can be applied to the joint

- boost::array< dReal, 3 > _vweights

  the weights of the joint for computing distance metrics.

- boost::array< dReal, 3 > _voffsets
- boost::array< dReal, 3 > _vupperlimit
- boost::array< boost::shared_ptr< MIMIC >, 3 > _vmimic

  the mimic properties of each of the joint axes. It is theoretically possible for a multi-dof joint to have one axes mimiced and the others free. When cloning, is it ok to copy this and assume it is constant?

- std::string _name
- boost::array< bool, 3 > _bIsCircular

11.25.1 Detailed Description

Information about a joint that controls the relationship between two links.

11.25.2 Member Enumeration Documentation

11.25.2.1 enum JointType

The type of joint movement.

Non-special joints that are combinations of revolution and prismatic joints. The first 4 bits specify the joint DOF, the next bits specify whether the joint is revolute (0) or prismatic (1). There can be also special joint types that are valid if the JointSpecialBit is set.

For multi-dof joints, the order is transform(parentlink) $*$ transform(axis0) $*$ transform(axis1) ...

11.25.3 Member Function Documentation

11.25.3.1 virtual void _ComputeInternalInformation (LinkPtr *plink0*, LinkPtr *plink1*, const Vector & *vanchor*, const std::vector< Vector > & *vaxes*, const std::vector< dReal > & *vcurrentvalues*) `[protected, virtual]`

Compute internal transformations and specify the attached links of the joint.

Called after the joint protected parameters {vAxes, vanchor, and _voffsets} have been initialized. vAxes and vanchor should be in the frame of plink0. Compute the left and right multiplications of the joint transformation and cleans up the attached bodies. After function completes, the following parameters are initialized: _tRight, _tLeft, _tinvRight, _tinvLeft, _attachedbodies. _attachedbodies does not necessarily contain the links in the same order as they were input.

Parameters

plink0 the first attaching link, all axes and anchors are defined in its coordinate system

plink1 the second attaching link

vanchor the anchor of the rotation axes

vaxes the axes in plink0's coordinate system of the joints

vinitialvalues the current values of the robot used to set the 0 offset of the robot

11.25.3.2 virtual void _ComputePartialVelocities (std::vector< std::pair< int, dReal > > & *vpartials*, int *iaxis*, std::map< std::pair< MIMIC::DOFFormat, int >, dReal > & *mapcachedpartials*) const `[protected, virtual]`

computes the partial velocities with respect to all dependent DOFs specified by MIMIC::_vmimicdofs.

If the joint is not mimic, then just returns its own index

Parameters

$\rightarrow$ ***vpartials*** A list of dofindex/velocity_partial pairs. The final velocity is computed by taking the dot product. The dofindices do not repeat.

$\leftarrow$ ***iaxis*** the axis

$\leftrightarrow$ ***vcachedpartials*** set of cached partials for each degree of freedom

11.25.3.3 virtual void GetAccelerationLimits (std::vector< dReal > & *vmax*, bool *bAppend* = `false`) const `[virtual]`

Returns the max accelerations of the joint.

Parameters

$\rightarrow$ ***the*** max acceleration

$\leftarrow$ ***bAppend*** if true will append to the end of the vector instead of erasing it

11.25.3.4 virtual Vector GetAxis (int *axis* = `0`) const `[virtual]`

The axis of the joint in global coordinates.

Parameters

$\leftarrow$ ***axis*** the axis to get

11.25.3.5 int GetDOFIndex () const

Get the degree of freedom index in the body's DOF array.

This does not index in KinBody::GetJoints() directly! In other words, KinBody::GetDOFValues()[GetDOFIndex()] == GetValues()[0]

11.25.3.6 virtual Vector GetInternalHierarchyAnchor () const [virtual]

11.25.3.7 virtual void GetLimits (std::vector< dReal > & *vLowerLimit*, std::vector< dReal > & *vUpperLimit*, bool *bAppend* = false) const [virtual]

Returns the limits of the joint.

Parameters

→ ***vLowerLimit*** the lower limits
→ ***vUpperLimit*** the upper limits
← ***bAppend*** if true will append to the end of the vector instead of erasing it

11.25.3.8 const std::vector<dReal> GetMimicCoeffs () const

11.25.3.9 void GetMimicDOFIndices (std::vector< int > & *vmimicdofs*, int *axis* = 0) const

Returns the set of DOF indices that the computation of a joint axis depends on. Order is arbitrary.

If the mimic joint uses the values of other mimic joints, then the dependent DOFs of that joint are also copied over. Therefore, the dof indices returned can be more than the actual variables used in the equation.

Exceptions

openrave_exception Throws an exception if the axis is not mimic.

11.25.3.10 std::string GetMimicEquation (int *axis* = 0, int *type* = 0, const std::string & *format* = "") const

If the joint is mimic, returns the equation to compute its value.

Parameters

← ***axis*** the axis index

← ***type*** 0 for position, 1 for velocity, 2 for acceleration.

← ***format*** the format the equations are returned in. If empty or "fparser", equation in fparser format. Also supports: "mathml".

MathML:

Set 'format' to "mathml". The joint variables are specified with <csymbol>. If a targetted joint has more than one degree of freedom, then axis is suffixed with _%d. If 'type' is 1 or 2, the partial derivatives are outputted as consecutive $$ tags in the same order as MIMIC::_vdofformat

11.25.3.11 int GetMimicJointIndex () const

11.25.3.12 dReal GetResolution () const

The discretization of the joint used when line-collision checking.

The resolutions are set as large as possible such that the joint will not go through obstacles of determined size.

11.25.3.13 virtual void GetValues (std::vector< dReal > & *values*, bool *bAppend* = false) const [virtual]

Return all the joint values with the correct offsets applied.

Parameters

bAppend if true will append to the end of the vector instead of erasing it

Returns

degrees of freedom of the joint (even if pValues is NULL)

11.25.3.14 virtual void GetVelocities (std::vector< dReal > & *values*, bool *bAppend* = `false`) const [virtual]

Gets the joint velocities.

Parameters

bAppend if true will append to the end of the vector instead of erasing it

Returns

the degrees of freedom of the joint (even if pValues is NULL)

11.25.3.15 virtual void GetVelocityLimits (std::vector< dReal > & *vmax*, bool *bAppend* = `false`) const [virtual]

Returns the max velocities of the joint.

Parameters

→ ***the*** max velocity

← ***bAppend*** if true will append to the end of the vector instead of erasing it

11.25.3.16 dReal GetWrapOffset (int *iaxis* = `0`) const

Return internal offset parameter that determines the branch the angle centers on.

Wrap offsets are needed for rotation joints since the range is limited to 2∗pi. This allows the wrap offset to be set so the joint can function in [-pi+offset,pi+offset]..

Parameters

iaxis the axis to get the offset from

11.25.3.17 virtual bool IsCircular (int *iaxis*) const [virtual]

Return true if joint axis has an identification at some of its lower and upper limits.

An identification of the lower and upper limits means that once the joint reaches its upper limits, it is also at its lower limit. The most common identification on revolute joints at -pi and pi. 'circularity' means the joint does not stop at limits. Although currently not developed, it could be possible to support identification for joints that are not revolute.

11.25.3.18 bool IsMimic (int *axis* = `-1`) const

Returns true if a particular axis of the joint is mimiced.

Parameters

axis the axis to query. When -1 returns true if any of the axes have mimic joints

11.25.3.19 virtual bool IsPrismatic (int *iaxis*) const `[virtual]`

returns true if the axis describes a translation around an axis.

Parameters

iaxis the axis of the joint to return the results for

11.25.3.20 virtual bool IsRevolute (int *iaxis*) const `[virtual]`

returns true if the axis describes a rotation around an axis.

Parameters

iaxis the axis of the joint to return the results for

11.25.3.21 virtual void SetAccelerationLimits (const std::vector< dReal > & *vmax*) `[virtual]`

See also

GetAccelerationLimits

11.25.3.22 virtual void SetJointLimits (const std::vector< dReal > & *lower*, const std::vector< dReal > & *upper*) [virtual]

11.25.3.23 virtual void SetLimits (const std::vector< dReal > & *lower*, const std::vector< dReal > & *upper*) [virtual]

See also

GetLimits

11.25.3.24 void SetMimicEquations (int *axis*, const std::string & *poseq*, const std::string & *veleq*, const std::string & *acceleq* = "")

Sets the mimic properties of the joint.

The equations can use the joint names directly in the equation, which represent the position of the joint. Any non-mimic joint part of KinBody::GetJoints() can be used in the computation of the values. If a joint has more than one degree of freedom, then suffix it '_' and the axis index. For example universaljoint_0 $*$ 10 + sin(universaljoint_1).

See `http://warp.povusers.org/FunctionParser/fparser.html` for a full description of the equation formats.

The velocity and acceleration equations are specified in terms of partial derivatives, which means one expression needs to be specified per degree of freedom of used. In order to separate the expressions use "|name ...". The name should immediately follow '|'. For example:

|universaljoint_0 10 |universaljoint_1 10$*$cos(universaljoint_1)

If there is only one variable used in the position equation, then the equation can be specified directly without using "{}".

Parameters

- ← ***axis*** the axis to set the properties for.
- ← ***poseq*** Equation for joint's position. If it is empty, the mimic properties are turned off for this joint.
- ← ***veleq*** First-order partial derivatives of poseq with respect to all used DOFs. Only the variables used in poseq are allowed to be used. If poseq is not empty, this is required.

← ***acceleq*** Second-order partial derivatives of poseq with respect to all used DOFs. Only the variables used in poseq are allowed to be used. Optional.

Exceptions

openrave_exception Throws an exception if the mimic equation is invalid in any way.

11.25.3.25 virtual void SetOffset (dReal *offset*, int *iaxis* = 0) [virtual]

11.25.3.26 virtual void SetVelocityLimits (const std::vector< dReal > & *vmax*) [virtual]

See also

GetVelocityLimits

11.25.3.27 virtual void SetWeights (const std::vector< dReal > & *weights*) [virtual]

See also

GetWeight

11.25.3.28 virtual void SetWrapOffset (dReal *offset*, int *iaxis* = 0) [virtual]

See also

GetWrapOffset

11.25.4 Member Data Documentation

11.25.4.1 boost::array<bool,3> _bIsCircular [protected]

See also

IsCircular

11.25.4.2 std::string _name [protected]

See also

GetName

11.25.4.3 boost::array<dReal,3> _voffsets [protected]

See also

GetOffset

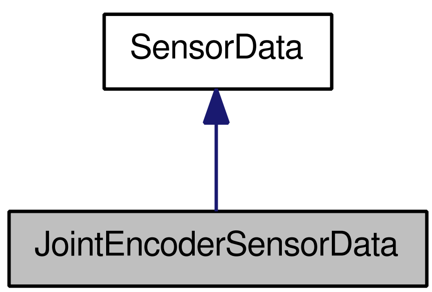
11.25.4.4 boost::array<dReal,3> _vupperlimit [protected]

joint limits

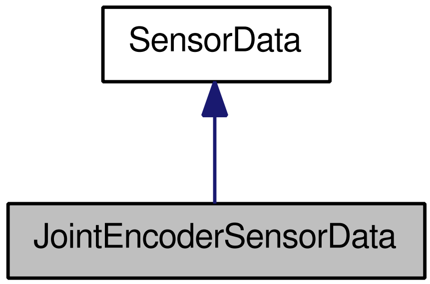
11.26 JointEncoderSensorData Class Reference

Stores joint angles and EE position.

Inheritance diagram for JointEncoderSensorData:



Collaboration diagram for JointEncoderSensorData:



Public Attributes

- std::vector< dReal > encoderValues

  measured joint angles in radians

- std::vector< dReal > encoderVelocity

  measured joint velocity in radians

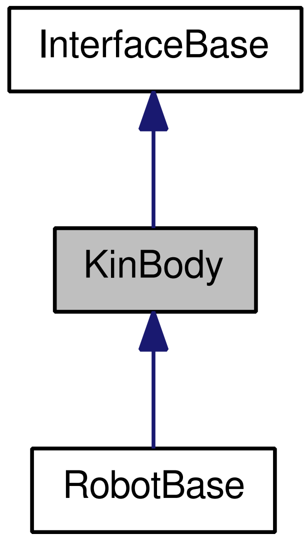
11.26.1 Detailed Description

Stores joint angles and EE position.

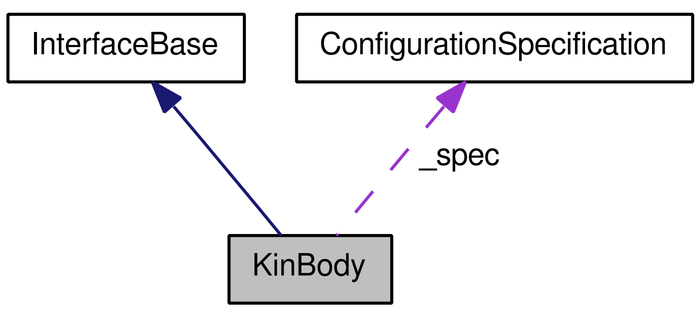
11.27 KinBody Class Reference

[interface] A kinematic body of links and joints. **If not specified, method is not multi-thread safe.** See Kinematics Body Concepts.

Inheritance diagram for KinBody:



Collaboration diagram for KinBody:



Classes

- class BodyState

  Stores the state of the current body that is published in a thread safe way from the environment without requiring locking the environment.

- class Joint

  Information about a joint that controls the relationship between two links.

- class KinBodyStateSaver

  Helper class to save and restore the entire kinbody state.

- class Link

  A rigid body holding all its collision and rendering data.

- class ManageData

  Access point of the sensor system that manages the body.

Public Types

- enum KinBodyProperty {

  Prop_Joints = 0x1, Prop_JointLimits = 0x2|Prop_Joints , Prop_JointProperties = 0x8|Prop_Joints, Prop_Links = 0x10,

  Prop_Name = 0x20, Prop_LinkDraw = 0x40, Prop_LinkGeometry = 0x80|Prop_Links, Prop_JointMimic = 0x100|Prop_Joints,

  Prop_JointAccelerationVelocityLimits = 0x200|Prop_Joints, Prop_LinkStatic = 0x400|Prop_Links, Prop_RobotManipulators = 0x00010000 , Prop_-RobotSensors = 0x00020000 ,

  Prop_RobotSensorPlacement = 0x00040000 , Prop_RobotActiveDOFs = 0x00080000, Prop_RobotManipulatorTool = 0x00100000 }

  A set of properties for the kinbody. These properties are used to describe a set of variables used in KinBody.

- enum SaveParameters {

  Save_LinkTransformation = 0x00000001, Save_LinkEnable = 0x00000002, Save_LinkVelocities = 0x00000004, Save_ActiveDOF = 0x00010000,

  Save_ActiveManipulator = 0x00020000, Save_GrabbedBodies = 0x00040000 }

  Parameters passed into the state savers to control what information gets saved.

- enum AdjacentOptions { AO_Enabled = 1, AO_ActiveDOFs = 2 }

  specifies the type of adjacent link information to receive

Public Member Functions

- virtual bool InitFromFile (const std::string &filename, const AttributesList &atts=AttributesList()) RAVE_DEPRECATED
- virtual bool InitFromData (const std::string &data, const AttributesList &atts=AttributesList()) RAVE_DEPRECATED

- virtual bool InitFromBoxes (const std::vector< AABB > &boxes, bool draw)

  Create a kinbody with one link composed of an array of aligned bounding boxes.

- virtual bool InitFromBoxes (const std::vector< OBB > &boxes, bool draw)

  Create a kinbody with one link composed of an array of oriented bounding boxes.

- virtual bool InitFromSpheres (const std::vector< Vector > &spheres, bool draw)

  Create a kinbody with one link composed of an array of spheres.

- virtual bool InitFromTrimesh (const Link::TRIMESH &trimesh, bool draw)

  Create a kinbody with one link composed of a triangle mesh surface.

- bool InitFromGeometries (std::list< KinBody::Link::GEOMPROPERTIES > &geometries, bool draw)

  Create a kinbody with one link composed of a list of geometries.

- virtual const std::string & GetName () const

  Unique name of the robot.

- virtual void SetName (const std::string &name)

  Set the name of the robot, notifies the environment and checks for uniqueness.

- virtual void SubtractDOFValues (std::vector< dReal > &values1, const std::vector< dReal > &values2) const

  Computes the configuration difference values1-values2 and stores it in values1.

- virtual void SubtractJointValues (std::vector< dReal > &q1, const std::vector< dReal > &q2) const RAVE_DEPRECATED
- virtual void SetDOFTorques (const std::vector< dReal > &torques, bool add)

  Adds a torque to every joint.

- virtual void SetJointTorques (const std::vector< dReal > &torques, bool add) RAVE_DEPRECATED
- virtual const std::vector< LinkPtr > & GetLinks () const

  Returns all the rigid links of the body.

- virtual LinkPtr GetLink (const std::string &name) const

  return a pointer to the link with the given name

- virtual void SimulationStep (dReal fElapsedTime)

  Updates the bounding box and any other parameters that could have changed by a simulation step.

- virtual void GetLinkTransformations (std::vector< Transform > &transforms) const

  get the transformations of all the links at once

- virtual void GetBodyTransformations (std::vector< Transform > &transforms) const RAVE_DEPRECATED
- virtual Transform GetTransform () const

  queries the transfromation of the first link of the body

- virtual bool SetVelocity (const Vector &linearvel, const Vector &angularvel)

  Set the velocity of the base link, rest of links are set to a consistent velocity so entire robot moves correctly.

- virtual void SetDOFVelocities (const std::vector< dReal > &vDOFVelocities, const Vector &linearvel, const Vector &angularvel, bool checklimits=false)

  Sets the velocity of the base link and each of the joints.

- virtual void SetDOFVelocities (const std::vector< dReal > &vDOFVelocities, bool checklimits=false)

  Sets the velocity of the joints.

- virtual void GetLinkVelocities (std::vector< std::pair< Vector, Vector > > &velocities) const

  Returns the linear and angular velocities for each link.

- virtual void SetTransform (const Transform &transform)

  set the transform of the first link (the rest of the links are computed based on the joint values).

- virtual AABB ComputeAABB () const

  Return an axis-aligned bounding box of the entire object in the world coordinate system.

- virtual Vector GetCenterOfMass () const

  Return the center of mass of entire robot in the world coordinate system.

- virtual void Enable (bool enable)

  Enables or disables the bodies.

- virtual void EnableLink (LinkPtr plink, bool bEnable) RAVE_DEPRECATED
- virtual bool IsEnabled () const
- virtual void SetDOFValues (const std::vector< dReal > &values, bool checklimits=false)

Sets the joint values of the robot.

- virtual void SetDOFValues (const std::vector< dReal > &values, const Transform &transform, bool checklimits=false)

  Sets the joint values and transformation of the body.

- virtual void SetLinkTransformations (const std::vector< Transform > &transforms)

  sets the transformations of all the links at once

- virtual void SetBodyTransformations (const std::vector< Transform > &transforms) RAVE_DEPRECATED
- virtual void SetLinkVelocities (const std::vector< std::pair< Vector, Vector > > &velocities)

  sets the link velocities

- virtual void CalculateJacobian (int linkindex, const Vector &offset, boost::multi_array< dReal, 2 > &vjacobian) const

  Computes the translation jacobian with respect to a world position.

- virtual void CalculateRotationJacobian (int linkindex, const Vector &quat, boost::multi_array< dReal, 2 > &vjacobian) const

  Computes the rotational jacobian as a quaternion with respect to an initial rotation.

- virtual void CalculateAngularVelocityJacobian (int linkindex, boost::multi_array< dReal, 2 > &vjacobian) const

  Computes the angular velocity jacobian of a specified link about the axes of world coordinates.

- virtual bool CheckSelfCollision (CollisionReportPtr report=CollisionReportPtr()) const

  Check if body is self colliding. Links that are joined together are ignored.

- virtual bool IsAttached (KinBodyConstPtr body) const
- virtual void GetAttached (std::set< KinBodyPtr > &setAttached) const

  Recursively get all attached bodies of this body, including this body.

- virtual bool IsRobot () const

  Return true if this body is derived from RobotBase.

- virtual int GetEnvironmentId () const

  return a unique id of the body used in the environment.

- virtual int8_t DoesAffect (int jointindex, int linkindex) const

  Returns a nonzero value if the joint effects the link transformation.

- virtual UserDataPtr GetViewerData () const
- virtual UserDataPtr GetGuiData () const RAVE_DEPRECATED
- virtual const std::set< int > & GetNonAdjacentLinks (int adjacentoptions=0) const

  return all possible link pairs that could get in collision.

- virtual const std::set< int > & GetAdjacentLinks () const

  return all possible link pairs whose collisions are ignored.

- virtual UserDataPtr GetPhysicsData () const
- virtual UserDataPtr GetCollisionData () const

  SetCollisionData.

- virtual int GetUpdateStamp () const

  Return a unique id for every transformation state change of any link. Used to check if robot state has changed.

- virtual UserDataPtr RegisterChangeCallback (int properties, const boost::function< void()> &callback)

  Register a callback with the interface.

- virtual const std::string & GetKinematicsGeometryHash () const

  A md5 hash unique to the particular kinematic and geometric structure of a KinBody.

- virtual void SetJointVelocities (const std::vector< dReal > &pJointVelocities) RAVE_DEPRECATED
- virtual void GetVelocity (Vector &linearvel, Vector &angularvel) const RAVE_-DEPRECATED
- virtual void SetZeroConfiguration ()

  Sets the joint offsets so that the current configuration becomes the new zero state of the robot.

Basic Information

Methods for accessing basic information about joints

- virtual int GetDOF () const

  Number controllable degrees of freedom of the body.

- virtual void GetDOFValues (std::vector< dReal > &v) const

Returns all the joint values as organized by the DOF indices.

- virtual void GetDOFVelocities (std::vector< dReal > &v) const

  Returns all the joint velocities as organized by the DOF indices.

- virtual void GetDOFLimits (std::vector< dReal > &lowerlimit, std::vector< dReal > &upperlimit) const

  Returns all the joint limits as organized by the DOF indices.

- virtual void GetDOFVelocityLimits (std::vector< dReal > &lowerlimit, std::vector< dReal > &upperlimit) const

  Returns all the joint velocity limits as organized by the DOF indices.

- virtual void GetDOFVelocityLimits (std::vector< dReal > &maxvelocities) const

  Returns the max velocity for each DOF.

- virtual void GetDOFAccelerationLimits (std::vector< dReal > &maxaccelerations) const

  Returns the max acceleration for each DOF.

- virtual void GetDOFMaxVel (std::vector< dReal > &v) const RAVE_DEPRECATED
- virtual void **GetDOFMaxAccel** (std::vector< dReal > &v) const RAVE_DEPRECATED
- virtual void **GetDOFMaxTorque** (std::vector< dReal > &v) const
- virtual void **GetDOFResolutions** (std::vector< dReal > &v) const
- virtual void **GetDOFWeights** (std::vector< dReal > &v) const
- const std::vector< JointPtr > & GetJoints () const

  Returns the joints making up the controllable degrees of freedom of the body.

- const std::vector< JointPtr > & GetPassiveJoints () const

  Returns the passive joints, order does not matter.

- virtual void GetRigidlyAttachedLinks (int linkindex, std::vector< LinkPtr > &vattachedlinks) const RAVE_DEPRECATED
- virtual const std::vector< JointPtr > & GetDependencyOrderedJoints () const

  Returns the joints in hierarchical order starting at the base link.

- virtual const std::vector< std::vector< std::pair< LinkPtr, JointPtr > > > & GetClosedLoops () const

  Return the set of unique closed loops of the kinematics hierarchy.

- virtual bool GetChain (int linkindex1, int linkindex2, std::vector< JointPtr > &vjoints) const

Computes the minimal chain of joints that are between two links in the order of linkindex1 to linkindex2.

- virtual bool GetChain (int linkindex1, int linkindex2, std::vector< LinkPtr > &vlinks) const

  similar to GetChain(int,int,std::vector<JointPtr>&) except returns the links along the path.

- virtual bool IsDOFInChain (int linkindex1, int linkindex2, int dofindex) const

  Returns true if the dof index affects the relative transformation between the two links.

- virtual int GetJointIndex (const std::string &name) const

  Return the index of the joint with the given name, else -1.

- virtual JointPtr GetJoint (const std::string &name) const

  Return a pointer to the joint with the given name. Search in the regular and passive joints.

- virtual JointPtr GetJointFromDOFIndex (int dofindex) const

  Returns the joint that covers the degree of freedom index.

Configuration Specification API

Functions dealing with configuration specifications

- virtual const ConfigurationSpecification & GetConfigurationSpecification () const

  return the configuration specification of the joint values and transform

- virtual ConfigurationSpecification GetConfigurationSpecificationIndices (const std::vector< int > &indices) const

  return the configuration specification of the specified joint indices.

- virtual void SetConfigurationValues (std::vector< dReal >::const_iterator itvalues, bool checklimits=false)

  sets joint values and transform of the body using configuration values as specified by GetConfigurationSpecification()

- virtual void GetConfigurationValues (std::vector< dReal > &v) const

  returns the configuration values as specified by GetConfigurationSpecification()

Static Public Member Functions

- static InterfaceType GetInterfaceTypeStatic ()

return the static interface type this class points to (used for safe casting)

Protected Member Functions

- KinBody (InterfaceType type, EnvironmentBasePtr penv)

  constructors declared protected so that user always goes through environment to create bodies

- virtual void SetPhysicsData (UserDataPtr pdata)

  custom data managed by the current active physics engine, should be set only by PhysicsEngineBase

- virtual void SetCollisionData (UserDataPtr pdata)

  custom data managed by the current active collision checker, should be set only by CollisionCheckerBase

- virtual void SetViewerData (UserDataPtr pdata)

  custom data managed by the current active viewer, should be set only by ViewerBase

- virtual void _ComputeInternalInformation ()

  Final post-processing stage before a kinematics body can be used.

- virtual void _ParametersChanged (int parameters)

  Called to notify the body that certain groups of parameters have been changed.

- virtual bool _IsAttached (KinBodyConstPtr body, std::set< KinBodyConstPtr > &setChecked) const

  Return true if two bodies should be considered as one during collision (ie one is grabbing the other).

- virtual void _AttachBody (KinBodyPtr body)

  adds an attached body

- virtual bool _RemoveAttachedBody (KinBodyPtr body)

  removes an attached body

- virtual void _ResetInternalCollisionCache ()

  resets cached information dependent on the collision checker (usually called when the collision checker is switched or some big mode is set.

- virtual boost::shared_ptr< FunctionParserBase< dReal > > _CreateFunctionParser ()

  creates the function parser connected to this body's joint values

Protected Attributes

- std::string _name

  name of body

- std::vector< JointPtr > _vecjoints
- std::vector< JointPtr > _vTopologicallySortedJoints
- std::vector< JointPtr > _vTopologicallySortedJointsAll

  Similar to _vDependencyOrderedJoints except includes _vecjoints and _vPassiveJoints.

- std::vector< int > _vTopologicallySortedJointIndicesAll

  the joint indices of the joints in _vTopologicallySortedJointsAll. Passive joint indices have _vecjoints.size() added to them.

- std::vector< JointPtr > _vDOFOrderedJoints

  all joints of the body ordered on how they are arranged within the degrees of freedom

- std::vector< LinkPtr > _veclinks
- std::vector< int > _vDOFIndices

  cached start joint indices, indexed by dof indices

- std::vector< std::pair< int16_t, int16_t > > _vAllPairsShortestPaths

  all-pairs shortest paths through the link hierarchy. The first value describes the parent link index, and the second value is an index into _vecjoints or _vPassiveJoints. If the second value is greater or equal to _vecjoints.size() then it indexes into _vPassiveJoints.

- std::vector< int8_t > _vJointsAffectingLinks

  joint x link: (jointindex∗_veclinks.size()+linkindex). entry is non-zero if the joint affects the link in the forward kinematics. If negative, the partial derivative of ds/dtheta should be negated.

- std::vector< std::vector< std::pair< LinkPtr, JointPtr > > > _vClosedLoops
- std::vector< std::vector< std::pair< int16_t, int16_t > > > _vClosedLoopIndices
- std::vector< JointPtr > _vPassiveJoints
- std::set< int > _setAdjacentLinks
- std::vector< std::pair< std::string, std::string > > _vForcedAdjacentLinks

  internally stores forced adjacent links

- std::list< KinBodyWeakPtr > _listAttachedBodies

  list of bodies that are directly attached to this body (can have duplicates)

- std::list< UserDataWeakPtr > _listRegisteredCallbacks

    callbacks to call when particular properties of the body change.

- boost::array< std::set< int >, 4 > _setNonAdjacentLinks

    contains cached versions of the non-adjacent links depending on values in AdjacentOptions. Declared as mutable since data is cached.

- int _nNonAdjacentLinkCache

    specifies what information is currently valid in the AdjacentOptions. Declared as mutable since data is cached. If 0x80000000 (ie < 0), then everything needs to be recomputed including _setNonAdjacentLinks[0].

- std::vector< Transform > _vInitialLinkTransformations

    the initial transformations of each link specifying at least one pose where the robot is collision free

- int _environmentid
- int _nUpdateStampId
- int _nParametersChanged

    set of parameters that changed and need callbacks

- UserDataPtr _pViewerData
- UserDataPtr _pPhysicsData
- UserDataPtr _pCollisionData
- uint32_t _nHierarchyComputed

    true if the joint heirarchy and other cached information is computed

11.27.1 Detailed Description

[interface] A kinematic body of links and joints. **If not specified, method is not multi-thread safe.** See Kinematics Body Concepts.

11.27.2 Member Enumeration Documentation

11.27.2.1 enum AdjacentOptions

specifies the type of adjacent link information to receive

Enumerator:

AO_Enabled return only enabled link pairs

AO_ActiveDOFs return only link pairs that have an active in its path

11.27.2.2 enum KinBodyProperty

A set of properties for the kinbody. These properties are used to describe a set of variables used in KinBody.

Enumerator:

- ***Prop_Joints*** all properties of all joints
- ***Prop_JointLimits*** regular limits
- ***Prop_JointProperties*** max velocity, max acceleration, resolution, max torque
- ***Prop_Links*** all properties of all links
- ***Prop_Name*** name changed
- ***Prop_LinkDraw*** toggle link geometries rendering
- ***Prop_LinkGeometry*** the geometry of the link changed
- ***Prop_JointMimic*** joint mimic equations
- ***Prop_JointAccelerationVelocityLimits*** velocity + acceleration
- ***Prop_LinkStatic*** static property of link changed
- ***Prop_RobotManipulators*** [robot only] all properties of all manipulators
- ***Prop_RobotSensors*** [robot only] all properties of all sensors
- ***Prop_RobotSensorPlacement*** [robot only] relative sensor placement of sensors
- ***Prop_RobotActiveDOFs*** [robot only] active dofs changed
- ***Prop_RobotManipulatorTool*** [robot only] the tool coordinate system changed

11.27.2.3 enum SaveParameters

Parameters passed into the state savers to control what information gets saved.

Enumerator:

- ***Save_LinkTransformation*** [default] save link transformations
- ***Save_LinkEnable*** [default] save link enable states
- ***Save_LinkVelocities*** save the link velocities
- ***Save_ActiveDOF*** [robot only], saves and restores the current active degrees of freedom
- ***Save_ActiveManipulator*** [robot only], saves the active manipulator
- ***Save_GrabbedBodies*** [robot only], saves the grabbed state of the bodies. This does not affect the configuraiton of those bodies.

11.27.3 Member Function Documentation

11.27.3.1 virtual void _ComputeInternalInformation () `[protected, virtual]`

Final post-processing stage before a kinematics body can be used.

This method is called after the body is finished being initialized with data and before being added to the environment. Also builds the hashes. Builds the internal hierarchy and kinematic body hash.

Avoids making specific calls on the collision checker (like CheckCollision) or physics engine (like simulating velocities/torques) since this information can change depending on the attached plugin.

Reimplemented in RobotBase.

11.27.3.2 virtual void _ParametersChanged (int *parameters*) `[protected, virtual]`

Called to notify the body that certain groups of parameters have been changed.

This function in calls every registers calledback that is tracking the changes. It also recomputes the hashes if geometry changed.

Reimplemented in RobotBase.

11.27.3.3 virtual bool _RemoveAttachedBody (KinBodyPtr *body*) `[protected, virtual]`

removes an attached body

Returns

true if body was successfully found and removed

11.27.3.4 virtual void CalculateAngularVelocityJacobian (int *linkindex*, boost::multi_array< dReal, 2 > & *vjacobian*) const `[virtual]`

Computes the angular velocity jacobian of a specified link about the axes of world coordinates.

Parameters

linkindex of the link that the rotation is attached to

vjacobian 3xDOF matrix

11.27.3.5 virtual void CalculateJacobian (int *linkindex*, const Vector & *offset*, boost::multi_array< dReal, 2 > & *vjacobian*) const `[virtual]`

Computes the translation jacobian with respect to a world position.

Gets the jacobian with respect to a link by computing the partial differentials for all joints that in the path from the root node to GetLinks()[index] (doesn't touch the rest of the values)

Parameters

linkindex of the link that the rotation is attached to

position position in world space where to compute derivatives from.

vjacobian 3xDOF matrix

11.27.3.6 virtual void CalculateRotationJacobian (int *linkindex*, const Vector & *quat*, boost::multi_array< dReal, 2 > & *vjacobian*) const `[virtual]`

Computes the rotational jacobian as a quaternion with respect to an initial rotation.

Parameters

linkindex of the link that the rotation is attached to

qInitialRot the rotation in world space whose derivative to take from.

vjacobian 4xDOF matrix

11.27.3.7 virtual int8_t DoesAffect (int *jointindex*, int *linkindex*) const `[virtual]`

Returns a nonzero value if the joint effects the link transformation.

In closed loops, all joints on all paths to the root link are counted as affecting the link. If a mimic joint affects the link, then all the joints used in the mimic joint's computation affect the link. If negative, the partial derivative of the Jacobian should be negated.

Parameters

jointindex index of the joint

linkindex index of the link

11.27.3.8 virtual void EnableLink (LinkPtr *plink*, bool *bEnable*) `[virtual]`

11.27.3.9 virtual void GetAttached (std::set< KinBodyPtr > & *setAttached*) const `[virtual]`

Recursively get all attached bodies of this body, including this body.

Parameters

setAttached fills with the attached bodies. If any bodies are already in setAttached, then ignores recursing on their attached bodies.

11.27.3.10 virtual void GetBodyTransformations (std::vector< Transform > & *transforms*) const `[virtual]`

11.27.3.11 virtual bool GetChain (int *linkindex1*, int *linkindex2*, std::vector< JointPtr > & *vjoints*) const `[virtual]`

Computes the minimal chain of joints that are between two links in the order of linkindex1 to linkindex2.

Passive joints are also used in the computation of the chain and can be returned. Note that a passive joint has a joint index and dof index of -1.

Parameters

← ***linkindex1*** the link index to start the search

← ***linkindex2*** the link index where the search ends

→ ***vjoints*** the joints to fill that describe the chain

Returns

true if the two links are connected (vjoints will be filled), false if the links are separate

11.27.3.12 virtual const std::vector< std::vector< std::pair<LinkPtr, JointPtr> > >& GetClosedLoops () const [virtual]

Return the set of unique closed loops of the kinematics hierarchy.

Each loop is a set of link indices and joint indices. For example, a loop of link indices: [l_0,l_1,l_2] will consist of three joints connecting l_0 to l_1, l_1 to l_2, and l_2 to l_0. The first element in the pair is the link l_X, the second element in the joint connecting l_X to l_(X+1).

11.27.3.13 virtual ConfigurationSpecification GetConfigurationSpecificationIndices (const std::vector< int > & *indices*) const [virtual]

return the configuration specification of the specified joint indices.

Note that the return type is by-value, so should not be used in iteration

11.27.3.14 virtual const std::vector<JointPtr>& GetDependencyOrderedJoints () const [virtual]

Returns the joints in hierarchical order starting at the base link.

In the case of closed loops, the joints are returned in the order closest to the root. All the joints affecting a particular joint's transformation will always come before the joint in the list.

11.27.3.15 virtual int GetDOF () const [virtual]

Number controllable degrees of freedom of the body.

Only uses _vecjoints and last joint for computation, so can work before _ComputeInternalInformation is called.

11.27.3.16 virtual void GetDOFMaxVel (std::vector< dReal > & *v*) const [virtual]

11.27.3.17 virtual int GetEnvironmentId () const [virtual]

return a unique id of the body used in the environment.

If object is not added to the environment, this will return 0. So checking if GetEnvironmentId() is 0 is a good way to check if object is present in the environment. This id will not be copied when cloning in order to respect another environment's ids.

11.27.3.18 virtual UserDataPtr GetGuiData () const [virtual]

11.27.3.19 virtual JointPtr GetJointFromDOFIndex (int *dofindex*) const [virtual]

Returns the joint that covers the degree of freedom index.

Note that the mapping of joint structures is not the same as the values in GetJointValues since each joint can have more than one degree of freedom.

11.27.3.20 virtual const std::string& GetKinematicsGeometryHash () const [virtual]

A md5 hash unique to the particular kinematic and geometric structure of a KinBody.

This 32 byte string can be used to check if two bodies have the same kinematic structure and can be used to index into tables when looking for body-specific models. OpenRAVE stores all such models in the OPENRAVE_HOME directory (usually ~/.openrave), indexed by the particular robot/body hashes.

Returns

md5 hash string of kinematics/geometry

11.27.3.21 virtual const std::set<int>& GetNonAdjacentLinks (int *adjacentoptions* = 0) const [virtual]

return all possible link pairs that could get in collision.

Parameters

adjacentoptions a bitmask of AdjacentOptions values

11.27.3.22 const std::vector<JointPtr>& GetPassiveJoints () const

Returns the passive joints, order does not matter.

A passive joint is not directly controlled by the body's degrees of freedom so it has no joint index and no dof index. Passive joints allows mimic joints to be hidden from the users. However, there are cases when passive joints are not mimic; for example, suspension mechanism on vehicles.

11.27.3.23 virtual UserDataPtr GetPhysicsData () const [virtual]

See also

SetPhysicsData

11.27.3.24 virtual void GetRigidlyAttachedLinks (int *linkindex*, std::vector< LinkPtr > & *vattachedlinks*) const [virtual]

See also

Link::GetRigidlyAttachedLinks (10/12/12)

11.27.3.25 virtual int GetUpdateStamp () const [virtual]

Return a unique id for every transformation state change of any link. Used to check if robot state has changed.

The stamp is used by the collision checkers, physics engines, or any other item that needs to keep track of any changes of the KinBody as it moves. Currently stamps monotonically increment for every transformation/joint angle change.

11.27.3.26 virtual void GetVelocity (Vector & *linearvel*, Vector & *angularvel*) const `[virtual]`

11.27.3.27 virtual UserDataPtr GetViewerData () const `[virtual]`

See also

SetViewerData

11.27.3.28 virtual bool InitFromBoxes (const std::vector< OBB > & *boxes*, bool *draw*) `[virtual]`

Create a kinbody with one link composed of an array of oriented bounding boxes.

Parameters

boxes the array of oriented bounding boxes that will comprise of the body

draw if true, the boxes will be rendered in the scene

11.27.3.29 virtual bool InitFromBoxes (const std::vector< AABB > & *boxes*, bool *draw*) `[virtual]`

Create a kinbody with one link composed of an array of aligned bounding boxes.

Parameters

boxes the array of aligned bounding boxes that will comprise of the body

draw if true, the boxes will be rendered in the scene

11.27.3.30 virtual bool InitFromData (const std::string & *data*, const AttributesList & *atts* = `AttributesList()`) `[virtual]`

See also

EnvironmentBase::ReadKinBodyXMLData

Reimplemented in RobotBase.

11.27.3.31 virtual bool InitFromFile (const std::string & *filename*, const AttributesList & *atts* = `AttributesList()`) `[virtual]`

See also

EnvironmentBase::ReadKinBodyXMLFile

Reimplemented in RobotBase.

11.27.3.32 bool InitFromGeometries (std::list< KinBody::Link::GEOMPROPERTIES > & *geometries*, bool *draw*)

Create a kinbody with one link composed of a list of geometries.

Parameters

geometries In order to save memory, the geometries in this list are transferred to the link. After function completes, the size should be 0.

draw if true, will be rendered in the scene

11.27.3.33 virtual bool InitFromSpheres (const std::vector< Vector > & *spheres*, bool *draw*) `[virtual]`

Create a kinbody with one link composed of an array of spheres.

Parameters

spheres the XYZ position of the spheres with the W coordinate representing the individual radius

11.27.3.34 virtual bool InitFromTrimesh (const Link::TRIMESH & *trimesh*, bool *draw*) [virtual]

Create a kinbody with one link composed of a triangle mesh surface.

Parameters

trimesh the triangle mesh

draw if true, will be rendered in the scene

11.27.3.35 virtual bool IsAttached (KinBodyConstPtr *body*) const [virtual]

Returns

true if two bodies should be considered as one during collision (ie one is grabbing the other)

11.27.3.36 virtual bool IsDOFInChain (int *linkindex1*, int *linkindex2*, int *dofindex*) const [virtual]

Returns true if the dof index affects the relative transformation between the two links.

The internal implementation uses KinBody::DoesAffect, therefore mimic indices are correctly handled.

Parameters

← ***linkindex1*** the link index to start the search

← ***linkindex2*** the link index where the search ends

11.27.3.37 virtual bool IsEnabled () const [virtual]

Returns

true if any link of the KinBody is enabled

11.27.3.38 virtual UserDataPtr RegisterChangeCallback (int *properties*, const boost::function< void()> & *callback*) [virtual]

Register a callback with the interface.

Everytime a static property of the interface changes, all registered callbacks are called to update the users of the changes. Note that the callbacks will block the thread that made the parameter change.

Parameters

callback

properties a mask of the KinBodyProperty values that the callback should be called for when they change

11.27.3.39 virtual void SetBodyTransformations (const std::vector< Transform > & *transforms*) [virtual]

11.27.3.40 virtual void SetConfigurationValues (std::vector< dReal >::const_iterator *itvalues*, bool *checklimits* = false) [virtual]

sets joint values and transform of the body using configuration values as specified by GetConfigurationSpecification()

Parameters

itvalues the iterator to the vector containing the dof values. Must have GetConfigurationSpecification().GetDOF() values!

11.27.3.41 virtual void SetDOFTorques (const std::vector< dReal > & *torques*, bool *add*) [virtual]

Adds a torque to every joint.

Parameters

bAdd if true, adds to previous torques, otherwise resets the torques on all bodies and starts from 0

11.27.3.42 virtual void SetDOFValues (const std::vector< dReal > & *values*, const Transform & *transform*, bool *checklimits* = `false`) `[virtual]`

Sets the joint values and transformation of the body.

Parameters

values the values to set the joint angles (ordered by the dof indices)

transform represents the transformation of the first body. checklimits if true, will excplicitly check the joint limits before setting the values.

11.27.3.43 virtual void SetDOFValues (const std::vector< dReal > & *values*, bool *checklimits* = `false`) `[virtual]`

Sets the joint values of the robot.

Parameters

values the values to set the joint angles (ordered by the dof indices) checklimits if true, will excplicitly check the joint limits before setting the values.

11.27.3.44 virtual void SetDOFVelocities (const std::vector< dReal > & *vDOFVelocities*, bool *checklimits* = `false`) `[virtual]`

Sets the velocity of the joints.

Copies the current velocity of the base link and calls SetDOFVelocities(linearvel,angularvel,vDOFVelocities)

Parameters

$\leftarrow$ ***vDOFVelocity*** - velocities of each of the degrees of freeom checklimits if true, will excplicitly check the joint velocity limits before setting the values.

11.27.3.45 virtual void SetDOFVelocities (const std::vector< dReal > & *vDOFVelocities*, const Vector & *linearvel*, const Vector & *angularvel*, bool *checklimits* = `false`) `[virtual]`

Sets the velocity of the base link and each of the joints.

Computes internally what the correponding velocities of each of the links should be in order to achieve consistent results with the joint velocities. Sends the velocities to the physics engine. Velocities correspond to the link's coordinate system origin.

Parameters

← ***linearvel*** linear velocity of base link

← ***angularvel*** angular velocity rotation_axis∗theta_dot

← ***vDOFVelocities*** - velocities of each of the degrees of freeom

checklimits if true, will excplicitly check the joint velocity limits before setting the values.

11.27.3.46 virtual void SetJointTorques (const std::vector< dReal > & *torques*, bool *add*) `[virtual]`

11.27.3.47 virtual void SetJointVelocities (const std::vector< dReal > & *pJointVelocities*) `[virtual]`

11.27.3.48 virtual void SetTransform (const Transform & *transform*) `[virtual]`

set the transform of the first link (the rest of the links are computed based on the joint values).

Parameters

transform affine transformation

Reimplemented in RobotBase.

11.27.3.49 virtual bool SetVelocity (const Vector & *linearvel*, const Vector & *angularvel*) `[virtual]`

Set the velocity of the base link, rest of links are set to a consistent velocity so entire robot moves correctly.

Parameters

linearvel linear velocity

angularvel is the rotation axis $*$ angular speed

11.27.3.50 virtual void SetZeroConfiguration () `[virtual]`

Sets the joint offsets so that the current configuration becomes the new zero state of the robot.

When this function returns, the returned DOF values should be all zero for controllable joints. Mimic equations will use the new offsetted values when computing their joints. This is primarily used for calibrating a robot's zero position

11.27.3.51 virtual void SubtractDOFValues (std::vector< dReal > & *values1*, const std::vector< dReal > & *values2*) const `[virtual]`

Computes the configuration difference values1-values2 and stores it in values1.

Takes into account joint limits and wrapping of circular joints.

11.27.3.52 virtual void SubtractJointValues (std::vector< dReal > & *q1*, const std::vector< dReal > & *q2*) const `[virtual]`

11.27.4 Member Data Documentation

11.27.4.1 int _environmentid `[protected]`

See also

GetEnvironmentId

11.27.4.2 int _nUpdateStampId `[mutable, protected]`

See also

GetUpdateStamp

11.27.4.3 UserDataPtr _pCollisionData [protected]

See also

SetCollisionData

11.27.4.4 UserDataPtr _pPhysicsData [protected]

See also

SetPhysicsData

11.27.4.5 UserDataPtr _pViewerData [protected]

See also

SetViewerData

11.27.4.6 std::set<int> _setAdjacentLinks [protected]

a set of which links are connected to which if link i and j are connected then i|(j<<16) will be in the set where i<j.

11.27.4.7 std::vector< std::vector< std::pair<int16_t,int16_t> > > _vClosedLoopIndices [protected]

See also

GetClosedLoops

11.27.4.8 std::vector< std::vector< std::pair<LinkPtr,JointPtr> > > _vClosedLoops [protected]

See also

GetClosedLoops

11.27.4.9 std::vector<JointPtr> _vecjoints [protected]

See also

GetJoints

11.27.4.10 std::vector<LinkPtr> _veclinks [protected]

See also

GetLinks

11.27.4.11 std::vector<JointPtr> _vPassiveJoints [protected]

See also

GetPassiveJoints()

11.27.4.12 std::vector<JointPtr> _vTopologicallySortedJoints [protected]

See also

GetDependencyOrderedJoints

11.28 KinBodyStateSaver Class Reference

Helper class to save and restore the entire kinbody state.

Protected Attributes

- int _options
  saved options

11.28.1 Detailed Description

Helper class to save and restore the entire kinbody state. Options can be passed to the constructor in order to choose which parameters to save (see SaveParameters)

11.29 LineCollisionConstraint Class Reference

Line collision.

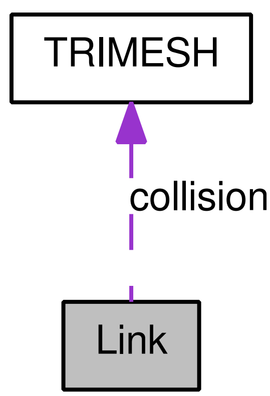
11.29.1 Detailed Description

Line collision.

11.30 Link Class Reference

A rigid body holding all its collision and rendering data.

Collaboration diagram for Link:



Classes

- class GEOMPROPERTIES

- class TRIMESH

    User data for trimesh geometries. Vertices are defined in counter-clockwise order for outward pointing faces.

Public Member Functions

- Link (KinBodyPtr parent)

    pass in a ODE world

- bool IsStatic () const

    Indicates a static body that does not move with respect to the root link.

- virtual bool IsEnabled () const

    returns true if the link is enabled.

- virtual void Enable (bool enable)

    Enables a Link. An enabled link takes part in collision detection and physics simulations.

- KinBodyPtr GetParent () const

    parent body that link belong to.

- int GetIndex () const

    unique index into parent KinBody::GetLinks vector

- virtual AABB ComputeAABB () const

    Compute the aabb of all the geometries of the link in the world coordinate system.

- Transform GetTransform () const

    Return the current transformation of the link in the world coordinate system.

- virtual void GetParentLinks (std::vector< boost::shared_ptr< Link > > &vParentLinks) const

    Return all the direct parent links in the kinematics hierarchy of this link.

- virtual bool IsParentLink (boost::shared_ptr< Link const > plink) const

    Tests if a link is a direct parent.

- Vector GetCOMOffset () const
- virtual void SetStatic (bool bStatic)

    sets a link to be static.

- virtual void SetTransform (const Transform &transform)

    Sets the transform of the link regardless of kinematics.

- virtual void SetForce (const Vector &force, const Vector &pos, bool add)
- virtual void SetTorque (const Vector &torque, bool add)
- virtual void SetVelocity (const Vector &linearvel, const Vector &angularvel)
- virtual void GetVelocity (Vector &linearvel, Vector &angularvel) const
- const std::list< GEOMPROPERTIES > & GetGeometries () const

    returns a list of all the geometry objects.

- virtual void SwapGeometries (std::list< GEOMPROPERTIES > &listNewGeometries)

    swaps the current geometries with the new geometries.

- virtual bool ValidateContactNormal (const Vector &position, Vector &normal) const
- virtual bool IsRigidlyAttached (boost::shared_ptr< Link const > plink) const

    returns true if plink is rigidily attahced to this link.

- virtual void GetRigidlyAttachedLinks (std::vector< boost::shared_ptr< Link > > &vattachedlinks) const

    Gets all the rigidly attached links to linkindex, also adds the link to the list.

Protected Member Functions

- virtual void _Update ()

    Updates the cached information due to changes in the collision data.

Protected Attributes

- Transform _t
- TransformMatrix _transMass

    the 3x3 inertia and center of mass of the link in the link's coordinate system

- TRIMESH collision
- std::string _name

    optional link name

- std::list< GEOMPROPERTIES > _listGeomProperties
- bool _bStatic
- bool _bIsEnabled

11.30.1 Detailed Description

A rigid body holding all its collision and rendering data.

11.30.2 Member Function Documentation

11.30.2.1 Vector GetCOMOffset () const

Returns

center of mass offset in the link's local coordinate frame

11.30.2.2 virtual void GetParentLinks (std::vector< boost::shared_ptr< Link > > & *vParentLinks*) const `[virtual]`

Return all the direct parent links in the kinematics hierarchy of this link.

A parent link is is immediately connected to this link by a joint and has a path to the root joint so that it is possible to compute this link's transformation from its parent.

Parameters

→ ***filled*** with the parent links

11.30.2.3 virtual void GetRigidlyAttachedLinks (std::vector< boost::shared_ptr< Link > > & *vattachedlinks*) const `[virtual]`

Gets all the rigidly attached links to linkindex, also adds the link to the list.

Parameters

vattachedlinks the array to insert all links attached to linkindex with the link itself.

11.30.2.4 virtual void GetVelocity (Vector & *linearvel*, Vector & *angularvel*) const `[virtual]`

get the velocity of the link

Parameters

→ ***linearvel*** the translational velocity

→ ***angularvel*** is the rotation axis $*$ angular speed

11.30.2.5 virtual bool IsEnabled () const [virtual]

returns true if the link is enabled.

See also

Enable

11.30.2.6 virtual bool IsParentLink (boost::shared_ptr< Link const > *plink*) const [virtual]

Tests if a link is a direct parent.

See also

GetParentLinks

Parameters

link The link to test if it is one of the parents of this link.

11.30.2.7 bool IsStatic () const

Indicates a static body that does not move with respect to the root link.

Static should be used when an object has infinite mass and shouldn't be affected by physics (including gravity). Collision still works.

11.30.2.8 virtual void SetForce (const Vector & *force*, const Vector & *pos*, bool *add*) [virtual]

adds an external force at pos (absolute coords)

Parameters

← ***force*** the direction and magnitude of the force
← ***pos*** in the world where the force is getting applied
← ***add*** if true, force is added to previous forces, otherwise it is set

11.30.2.9 virtual void SetStatic (bool *bStatic*) `[virtual]`

sets a link to be static.

Because this can affect the kinematics, it requires the body's internal structures to be recomputed

11.30.2.10 virtual void SetTorque (const Vector & *torque*, bool *add*) `[virtual]`

adds torque to a body (absolute coords)

Parameters

add if true, torque is added to previous torques, otherwise it is set

11.30.2.11 virtual void SetTransform (const Transform & *transform*) `[virtual]`

Sets the transform of the link regardless of kinematics.

Parameters

← ***t*** the new transformation

11.30.2.12 virtual void SetVelocity (const Vector & *linearvel*, const Vector & *angularvel*) `[virtual]`

forces the velocity of the link

Parameters

← ***linearvel*** the translational velocity
← ***angularvel*** is the rotation axis $*$ angular speed

11.30.2.13 virtual void SwapGeometries (std::list< GEOMPROPERTIES > & *listNewGeometries*) `[virtual]`

swaps the current geometries with the new geometries.

This gives a user control for dynamically changing the object geometry. Note that the kinbody/robot hash could change.

11.30.2.14 virtual bool ValidateContactNormal (const Vector & *position*, Vector & *normal*) const `[virtual]`

validates the contact normal on link and makes sure the normal faces "outside" of the geometry shape it lies on. An exception can be thrown if position is not on a geometry surface

Parameters

position the position of the contact point specified in the link's coordinate system, assumes it is on a particular geometry

normal the unit normal of the contact point specified in the link's coordinate system

Returns

true if the normal is changed to face outside of the shape

11.30.3 Member Data Documentation

11.30.3.1 bool _bIsEnabled `[protected]`

See also

IsEnabled

11.30.3.2 bool _bStatic `[protected]`

See also

IsStatic

11.30.3.3 std::list<GEOMPROPERTIES> _listGeomProperties [protected]

See also

GetGeometries

11.30.3.4 Transform _t [protected]

See also

GetTransform

11.30.3.5 TRIMESH collision [protected]

triangles for collision checking, triangles are always the triangulation of the body when it is at the identity transformation

11.31 ManageData Class Reference

Access point of the sensor system that manages the body.

Inherited by BodyData.

Public Member Functions

- virtual XMLReadableConstPtr GetData () const =0
- virtual KinBody::LinkPtr GetOffsetLink () const =0
- virtual bool IsPresent () const =0

    true if the object is being updated by the system due to its presence in the real environment

- virtual bool IsEnabled () const =0

    true if should update openrave body

- virtual bool IsLocked () const =0

    if true, the vision system should not destroy this object once it stops being present

- virtual bool Lock (bool bDoLock)=0

    set a lock on a particular body

11.31.1 Detailed Description

Access point of the sensor system that manages the body.

11.31.2 Member Function Documentation

11.31.2.1 virtual XMLReadableConstPtr GetData () const [pure virtual]

returns a pointer to the data used to initialize the BODY with AddKinBody. if psize is not NULL, will be filled with the size of the data in bytes This function will be used to restore bodies that were removed

11.31.2.2 virtual KinBody::LinkPtr GetOffsetLink () const [pure virtual]

particular link that sensor system is tracking. All transformations describe this link.

11.32 Manipulator Class Reference

Defines a chain of joints for an arm and set of joints for a gripper. Simplifies operating with them.

Public Member Functions

- virtual Transform GetTransform () const

  Return the transformation of the end effector (manipulator frame).

- virtual bool SetIkSolver (IkSolverBasePtr iksolver)

  Sets the ik solver and initializes it with the current manipulator.

- virtual IkSolverBasePtr GetIkSolver () const

  Returns the currently set ik solver.

- virtual int GetNumFreeParameters () const RAVE_DEPRECATED
- virtual bool GetFreeParameters (std::vector< dReal > &vFreeParameters) const RAVE_DEPRECATED
- virtual LinkPtr GetBase () const

  the base used for the iksolver

- virtual LinkPtr GetEndEffector () const *the end effector link (used to define workspace distance)*

- virtual Transform GetLocalToolTransform () const

  Return transform with respect to end effector defining the grasp coordinate system.

- virtual void SetLocalToolTransform (const Transform &t)

  Sets the local tool transform with respect to the end effector.

- virtual Transform GetGraspTransform () const RAVE_DEPRECATED
- virtual const std::vector< int > & GetGripperIndices () const

  Gripper indices of the joints that the manipulator controls.

- virtual const std::vector< int > & GetArmIndices () const

  Return the indices of the DOFs of the arm (used for IK, etc).

- virtual const std::vector< dReal > & GetClosingDirection () const

  return the normal direction to move joints to 'close' the hand

- virtual Vector GetLocalToolDirection () const

  direction of palm/head/manipulator used for approaching. defined inside the manipulator/grasp coordinate system

- virtual Vector GetDirection () const
- virtual bool FindIKSolution (const IkParameterization &param, std::vector< dReal > &solution, int filteroptions) const

  Find a close solution to the current robot's joint values.

- virtual bool FindIKSolutions (const IkParameterization &param, std::vector< std::vector< dReal > > &solutions, int filteroptions) const

  Find all the IK solutions for the given end effector transform.

- virtual IkParameterization GetIkParameterization (IkParameterizationType iktype) const

  returns the parameterization of a given IK type for the current manipulator position.

- virtual IkParameterization GetIkParameterization (const IkParameterization &ikparam) const

  returns a full parameterization of a given IK type for the current manipulator position using an existing IkParameterization as the seed.

- virtual void GetChildJoints (std::vector< JointPtr > &vjoints) const

  Get all child joints of the manipulator starting at the pEndEffector link.

- virtual void GetChildDOFIndices (std::vector< int > &vdofndices) const

  Get all child DOF indices of the manipulator starting at the pEndEffector link.

- virtual bool IsChildLink (LinkConstPtr plink) const

  returns true if a link is part of the child links of the manipulator.

- virtual void GetChildLinks (std::vector< LinkPtr > &vlinks) const

  Get all child links of the manipulator starting at pEndEffector link.

- virtual void GetIndependentLinks (std::vector< LinkPtr > &vlinks) const

  Get all links that are independent of the arm and gripper joints.

- virtual bool CheckEndEffectorCollision (const Transform &tEE, CollisionReportPtr report=CollisionReportPtr()) const

  Checks collision with only the gripper given its end-effector transform. Ignores disabled links.

- virtual bool CheckIndependentCollision (CollisionReportPtr report=CollisionReportPtr()) const

  Checks collision with the environment with all the independent links of the robot. Ignores disabled links.

- virtual bool IsGrabbing (KinBodyConstPtr body) const

  return true if the body is being grabbed by any link on this manipulator

- virtual void CalculateJacobian (boost::multi_array< dReal, 2 > &mjacobian) const

  computes the jacobian of the manipulator arm indices from the current manipulator frame origin.

- virtual void CalculateRotationJacobian (boost::multi_array< dReal, 2 > &mjacobian) const

  computes the quaternion jacobian of the manipulator arm indices from the current manipulator frame rotation.

- virtual void CalculateAngularVelocityJacobian (boost::multi_array< dReal, 2 > &mjacobian) const

  computes the angule axis jacobian of the manipulator arm indices.

- virtual const std::string & GetStructureHash () const

  Return hash of just the manipulator definition.

- virtual const std::string & GetKinematicsStructureHash () const

Return hash of all kinematics information that involves solving the inverse kinematics equations.

Protected Attributes

- std::vector< std::string > _vgripperjointnames

  names of the gripper joints

11.32.1 Detailed Description

Defines a chain of joints for an arm and set of joints for a gripper. Simplifies operating with them.

11.32.2 Member Function Documentation

11.32.2.1 virtual bool CheckEndEffectorCollision (const Transform & *tEE*, CollisionReportPtr *report* = `CollisionReportPtr()`) const `[virtual]`

Checks collision with only the gripper given its end-effector transform. Ignores disabled links.

Parameters

tEE the end effector transform

→ ***report*** [optional] collision report

Returns

true if a collision occurred

11.32.2.2 virtual bool CheckIndependentCollision (CollisionReportPtr *report* = `CollisionReportPtr()`) const `[virtual]`

Checks collision with the environment with all the independent links of the robot. Ignores disabled links.

Parameters

$\rightarrow$ ***report*** [optional] collision report

Returns

true if a collision occurred

11.32.2.3 virtual bool FindIKSolution (const IkParameterization & *param*, std::vector< dReal > & *solution*, int *filteroptions*) const [virtual]

Find a close solution to the current robot's joint values.

The function is a wrapper around the IkSolver interface. Note that the solution returned is not guaranteed to be the closest solution. In order to compute that, will have to compute all the ik solutions using FindIKSolutions.

Parameters

param The transformation of the end-effector in the global coord system

solution Will be of size GetArmIndices().size() and contain the best solution

$\leftarrow$ ***filteroptions*** A bitmask of IkFilterOptions values controlling what is checked for each ik solution.

11.32.2.4 virtual bool FindIKSolutions (const IkParameterization & *param*, std::vector< std::vector< dReal > > & *solutions*, int *filteroptions*) const [virtual]

Find all the IK solutions for the given end effector transform.

Parameters

param The transformation of the end-effector in the global coord system

solutions An array of all solutions, each element in solutions is of size GetArmIndices().size()

$\leftarrow$ ***filteroptions*** A bitmask of IkFilterOptions values controlling what is checked for each ik solution.

11.32.2.5 virtual const std::vector<int>& GetArmIndices () const [virtual]

Return the indices of the DOFs of the arm (used for IK, etc).

Usually the DOF indices from pBase to pEndEffector

11.32.2.6 virtual void GetChildLinks (std::vector< LinkPtr > & *vlinks*) const [virtual]

Get all child links of the manipulator starting at pEndEffector link.

The child links do not include the arm links.

11.32.2.7 virtual Vector GetDirection () const [virtual]

11.32.2.8 virtual bool GetFreeParameters (std::vector< dReal > & *vFreeParameters*) const [virtual]

11.32.2.9 virtual Transform GetGraspTransform () const [virtual]

11.32.2.10 virtual IkParameterization GetIkParameterization (const IkParameterization & *ikparam*) const [virtual]

returns a full parameterization of a given IK type for the current manipulator position using an existing IkParameterization as the seed.

Ideally pluging the returned ik parameterization into FindIkSolution should return the a manipulator configuration such that a new call to GetIkParameterization returns the same values.

Parameters

ikparam Some IK types like Lookat3D and TranslationLocalGlobal6D set constraints in the global coordinate system of the manipulator. Because these values are not stored in manipulator itself, they have to be passed in through an existing IkParameterization.

11.32.2.11 virtual IkParameterization GetIkParameterization (IkParameterizationType *iktype*) const [virtual]

returns the parameterization of a given IK type for the current manipulator position.

Ideally pluging the returned ik parameterization into FindIkSolution should return the a manipulator configuration such that a new call to GetIkParameterization returns the same values. In other words:

```
ikparam = manip->GetIkParameterization(iktype);
... move robot
std::vector<dReal> sol;
if( FindIKSolution(ikparam,sol, filteroptions) ) {
    manip->GetRobot()->SetActiveDOFs(manip->GetArmIndices());
    manip->GetRobot()->SetActiveDOFValues(sol);
    BOOST_ASSERT( dist(manip->GetIkParameterization(iktype), ikparam)
<= epsilon );
}
```

Parameters

iktype the type of parameterization to request

11.32.2.12 virtual void GetIndependentLinks (std::vector< LinkPtr > & *vlinks*) const [virtual]

Get all links that are independent of the arm and gripper joints.

In other words, returns all links not on the path from the base to the end effector and not children of the end effector. The base and all links rigidly attached to it are also returned.

11.32.2.13 virtual const std::string& GetKinematicsStructureHash () const [virtual]

Return hash of all kinematics information that involves solving the inverse kinematics equations.

This includes joint axes, joint positions, and final grasp transform. Hash is used to cache the solvers.

11.32.2.14 virtual int GetNumFreeParameters () const [virtual]

11.32.2.15 virtual Transform GetTransform () const [virtual]

Return the transformation of the end effector (manipulator frame).

All inverse kinematics and grasping queries are specifying this frame.

11.32.2.16 virtual bool IsChildLink (LinkConstPtr *plink*) const [virtual]

returns true if a link is part of the child links of the manipulator.

The child links do not include the arm links.

11.32.2.17 virtual bool SetIkSolver (IkSolverBasePtr *iksolver*) [virtual]

Sets the ik solver and initializes it with the current manipulator.

Due to complications with translation,rotation,direction,and ray ik, the ik solver should take into account the grasp transform (_tLocalTool) internally. The actual ik primitives are transformed into the base frame only.

11.32.2.18 virtual void SetLocalToolTransform (const Transform & *t*) [virtual]

Sets the local tool transform with respect to the end effector.

Because this call will change manipulator hash, it resets the loaded IK and sets the Prop_RobotManipulatorTool message.

11.33 ManipulatorIKGoalSampler Class Reference

Samples numsamples of solutions and each solution to vsolutions.

11.33.1 Detailed Description

Samples numsamples of solutions and each solution to vsolutions.

Parameters

nummaxsamples the max samples to query from a particular workspace goal. This does not necessarily mean every goal will have this many samples.

nummaxtries number of attemps to return a goal per Sample call.

fsampleprob The probability to attempt to sample a goal

11.34 MIMIC Struct Reference

Holds mimic information about position, velocity, and acceleration of one axis of the joint.

Public Attributes

- std::vector< DOFFormat > _vdofformat

  the format of the values the equation takes order is important.

- std::vector< DOFHierarchy > _vmimicdofs

  all dof indices that the equations depends on. DOFHierarchy::dofindex can repeat

- std::vector< boost::shared_ptr< FunctionParserBase< dReal > > > _accelfns

  the velocity and acceleration partial derivatives with respect to each of the values in _vdofformat

- boost::array< std::string, 3 > _equations

  the original equations

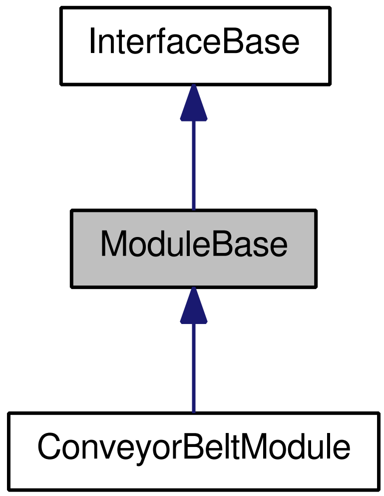
11.34.1 Detailed Description

Holds mimic information about position, velocity, and acceleration of one axis of the joint. In every array, [0] is position, [1] is velocity, [2] is acceleration.

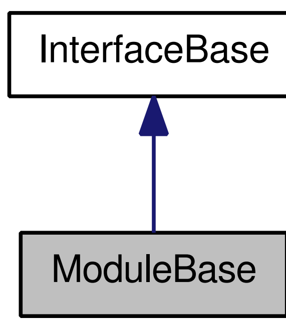
11.35 ModuleBase Class Reference

[interface] A loadable module of user code meant to solve a specific domain. **If not specified, method is not multi-thread safe.** See Module Concepts.

Inheritance diagram for ModuleBase:



Collaboration diagram for ModuleBase:



Public Member Functions

- virtual int main (const std::string &cmd)
- virtual void Destroy ()

  called when problem gets unloaded from environment

- virtual void Reset ()

  called when environment is reset

Static Public Member Functions

- static InterfaceType GetInterfaceTypeStatic ()

  return the static interface type this class points to (used for safe casting)

11.35.1 Detailed Description

[interface] A loadable module of user code meant to solve a specific domain. **If not specified, method is not multi-thread safe.** See Module Concepts.

Examples:

orconveyormovement.cpp, orpythonbinding.cpp, and plugincpp.cpp.

11.35.2 Member Function Documentation

11.35.2.1 virtual int main (const std::string & *cmd*) `[virtual]`

gets called every time a problem instance is loaded to initialize the problem. Robots might not necessarily be set before this function call returns 0 on success

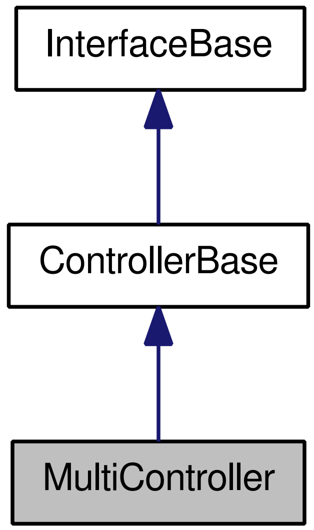
Examples:

plugincpp.cpp.

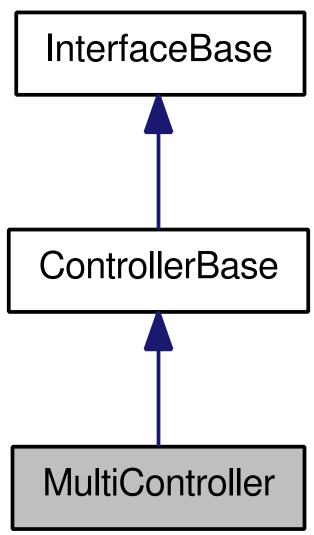
11.36 MultiController Class Reference

controller that manage multiple controllers, allows users to easily set multiple controllers for one robot.

Inheritance diagram for MultiController:



Collaboration diagram for MultiController:



Public Member Functions

- virtual bool Init (RobotBasePtr robot, const std::vector< int > &dofindices, int nControlTransformation)

    removes all controllers. ***[multi-thread safe]***

- virtual bool AttachController (ControllerBasePtr controller, const std::vector< int > &dofindices, int nControlTransformation)

    initializes and adds a controller, must be called after being initialized. ***[multi-thread safe]***

- virtual void RemoveController (ControllerBasePtr controller)

    removes a controller from being managed. ***[multi-thread safe]***

- virtual ControllerBasePtr GetController (int dof) const

    gets the controller responsible for dof (in the robot). If dof < 0, returns the transform controller. ***[multi-thread safe]***

- virtual bool IsDone ()

    returns true only if all controllers return true

- virtual dReal GetTime () const

    return the maximum time

- virtual void GetTorque (std::vector< dReal > &torque) const

11.36.1 Detailed Description

controller that manage multiple controllers, allows users to easily set multiple controllers for one robot. The class also make sure individual controllers do not have colliding DOF. It ignores the

Examples:

ormulticontrol.cpp.

11.36.2 Member Function Documentation

11.36.2.1 virtual bool AttachController (ControllerBasePtr *controller*, const std::vector< int > & *dofindices*, int *nControlTransformation*) `[virtual]`

initializes and adds a controller, must be called after being initialized. **[multi-thread safe]**

Parameters

controller the controller to init

dofindices robot dof indices to control

Exceptions

openrave_exception if the controller dofs interfere with current set dofs, will throw an exception

11.36.2.2 virtual void GetTorque (std::vector< dReal > & *torque*) const [virtual]

get torque/current/strain values

Parameters

torque [out] - returns the current torque/current/strain exerted by each of the dofs from outside forces. The feedforward and friction terms should be subtracted out already

Reimplemented from ControllerBase.

11.37 obb< T > Class Template Reference

An oriented bounding box.

11.37.1 Detailed Description

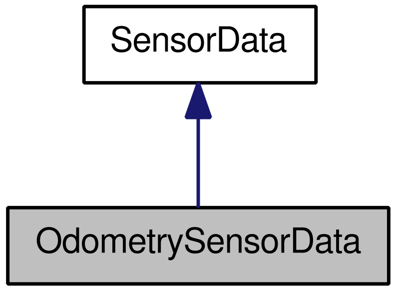
template<typename T> class OpenRAVE::geometry::obb< T >

An oriented bounding box.

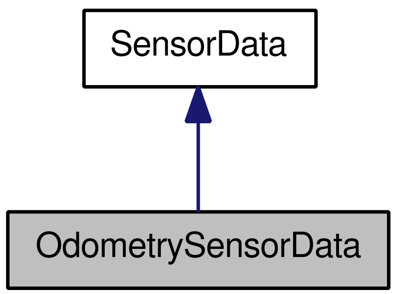
11.38 OdometrySensorData Class Reference

odometry data storing full 6D pose and velocity

Inheritance diagram for OdometrySensorData:



Collaboration diagram for OdometrySensorData:



Public Attributes

- Transform pose

    measured pose

- Vector angular_velocity

    measured velocity

- boost::array< dReal, 36 > pose_covariance

    Row major of 6x6 matrix about linear x, y, z axes.

- boost::array< dReal, 36 > velocity_covariance

    Row major of 6x6 matrix about rotational x, y, z axes.

11.38.1 Detailed Description

odometry data storing full 6D pose and velocity

11.39 openrave_exception Class Reference

Exception that all OpenRAVE internal methods throw; the error codes are held in OpenRAVEErrorCode.

Inherits std::exception.

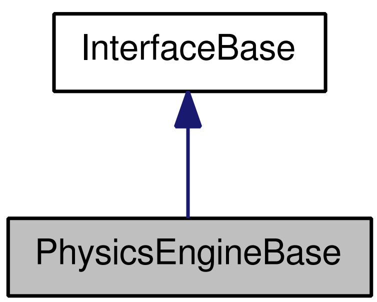
11.39.1 Detailed Description

Exception that all OpenRAVE internal methods throw; the error codes are held in OpenRAVEErrorCode.

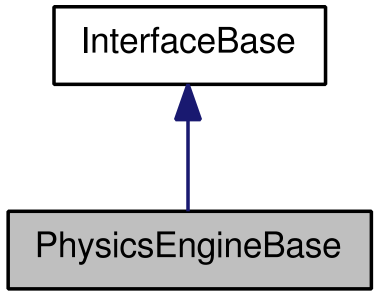
11.40 PhysicsEngineBase Class Reference

[interface] The physics engine interfaces supporting simulations and dynamics. See Physics Engine Concepts.

Inheritance diagram for PhysicsEngineBase:



Collaboration diagram for PhysicsEngineBase:



Public Member Functions

- virtual bool SetPhysicsOptions (int physicsoptions)=0

  Set basic physics engine using the PhysicsEngineOptions enum.

- virtual bool SetPhysicsOptions (std::ostream &sout, std::istream &sinput) RAVE_DEPRECATED=0
- virtual bool InitEnvironment ()=0

  called when environment sets this physics engine, engine assumes responsibility for KinBody::_pPhysicsData

- virtual void DestroyEnvironment ()=0
- virtual bool InitKinBody (KinBodyPtr body)=0

  notified when a new body has been initialized in the environment

- virtual bool SetLinkVelocity (KinBody::LinkPtr link, const Vector &linearvel, const Vector &angularvel)=0

  Force the body velocity of a link, velocities correspond to the link's coordinate system origin.

- virtual bool SetLinkVelocities (KinBodyPtr body, const std::vector< std::pair< Vector, Vector > > &velocities)=0

  Sets the velocities for each link, velocities correspond to the link's coordinate system origin.

- virtual bool GetLinkVelocity (KinBody::LinkConstPtr link, Vector &linearvel, Vector &angularvel)=0

  Gets the velocity of a link, velocities correspond to the link's coordinate system origin.

- virtual bool GetLinkVelocities (KinBodyConstPtr body, std::vector< std::pair< Vector, Vector > > &velocities)=0

  Sets the velocities for each link, velocities correspond to the link's coordinate system origin.

- virtual bool SetBodyForce (KinBody::LinkPtr link, const Vector &force, const Vector &position, bool bAdd) OPENRAVE_DUMMY_IMPLEMENTATION
- virtual bool SetBodyTorque (KinBody::LinkPtr link, const Vector &torque, bool bAdd) OPENRAVE_DUMMY_IMPLEMENTATION
- virtual bool AddJointTorque (KinBody::JointPtr pjoint, const std::vector< dReal > &pTorques) OPENRAVE_DUMMY_IMPLEMENTATION
- virtual bool GetLinkForceTorque (KinBody::LinkConstPtr link, Vector &force, Vector &torque) OPENRAVE_DUMMY_IMPLEMENTATION
- virtual void SetGravity (const Vector &gravity) OPENRAVE_DUMMY_-IMPLEMENTATION

  set the gravity direction

- virtual void SimulateStep (dReal fTimeElapsed)=0
- virtual bool GetBodyVelocity (KinBodyConstPtr body, std::vector< Vector > &vLinearVelocities, std::vector< Vector > &vAngularVelocities) RAVE_-DEPRECATED

Static Public Member Functions

- static InterfaceType GetInterfaceTypeStatic ()

  return the static interface type this class points to (used for safe casting)

11.40.1 Detailed Description

[interface] The physics engine interfaces supporting simulations and dynamics. See Physics Engine Concepts.

11.40.2 Member Function Documentation

11.40.2.1 virtual bool AddJointTorque (KinBody::JointPtr *pjoint*, const std::vector< dReal > & *pTorques*) [virtual]

adds torque to a joint

Parameters

pjoint - the joint the torque is added to

pTorques - the torques added to the joint. Pointer because the joint dof can be greater than 1.

11.40.2.2 virtual void DestroyEnvironment () `[pure virtual]`

called when environment switches to a different physics engine has to clear/deallocate any memory associated with KinBody::_pPhysicsData

11.40.2.3 virtual bool GetBodyVelocity (KinBodyConstPtr *body*, std::vector< Vector > & *vLinearVelocities*, std::vector< Vector > & *vAngularVelocities*) `[virtual]`

11.40.2.4 virtual bool GetLinkForceTorque (KinBody::LinkConstPtr *link*, Vector & *force*, Vector & *torque*) `[virtual]`

Parameters

← ***link*** a constant pointer to a link

→ ***force*** current force on the COM of the link

→ ***torque*** current torque on the COM of the link

11.40.2.5 virtual bool GetLinkVelocities (KinBodyConstPtr *body*, std::vector< std::pair< Vector, Vector > > & *velocities*) `[pure virtual]`

Sets the velocities for each link, velocities correspond to the link's coordinate system origin.

Parameters

→ ***velocities*** the linear and angular (axis $*$ angular_speed) velocities for each link.

11.40.2.6 virtual bool GetLinkVelocity (KinBody::LinkConstPtr *link*, Vector & *linearvel*, Vector & *angularvel*) `[pure virtual]`

Gets the velocity of a link, velocities correspond to the link's coordinate system origin.

Parameters

→ ***linearvel*** - linear velocity of base link

→ ***angularvel*** - angular velocity rotation_axis∗theta_dot

11.40.2.7 virtual bool SetBodyForce (KinBody::LinkPtr *link*, const Vector & *force*, const Vector & *position*, bool *bAdd*) `[virtual]`

add a force at a particular position in a link

Parameters

force the direction and magnitude of the force

position in the world where the force is getting applied

bAdd if true, force is added to previous forces, otherwise it is set

11.40.2.8 virtual bool SetBodyTorque (KinBody::LinkPtr *link*, const Vector & *torque*, bool *bAdd*) `[virtual]`

adds torque to a body (absolute coords)

Parameters

link the link to add a torque to

torque torque vector

bAdd if true, torque is added to previous torques, otherwise it is set

11.40.2.9 virtual void SetGravity (const Vector & *gravity*) `[virtual]`

set the gravity direction

Parameters

← ***joint***

→ ***force*** current accumulated force on the COM of the link

→ ***torque*** current accumulated torque on the COM of the link

11.40.2.10 virtual bool SetLinkVelocities (KinBodyPtr *body*, const std::vector< std::pair< Vector, Vector > > & *velocities*) `[pure virtual]`

Sets the velocities for each link, velocities correspond to the link's coordinate system origin.

Parameters

← ***body*** the body to query velocities from.

← ***velocities*** sets the linear and angular (axis ∗ angular_speed) velocities for each link

11.40.2.11 virtual bool SetLinkVelocity (KinBody::LinkPtr *link*, const Vector & *linearvel*, const Vector & *angularvel*) `[pure virtual]`

Force the body velocity of a link, velocities correspond to the link's coordinate system origin.

Parameters

← ***link*** link to set velocities.

← ***linearvel*** linear velocity of base link

← ***angularvel*** angular velocity rotation_axis∗theta_dot

11.40.2.12 virtual bool SetPhysicsOptions (std::ostream & *sout*, std::istream & *sinput*) `[pure virtual]`

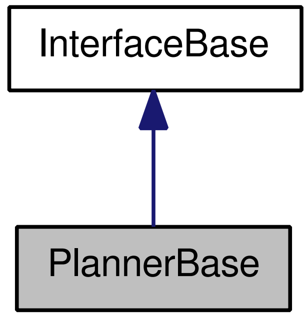
11.40.2.13 virtual void SimulateStep (dReal *fTimeElapsed*) `[pure virtual]`

dynamically simulate system for fTimeElapsed seconds add torques to the joints of the body. Torques disappear after one timestep of simulation

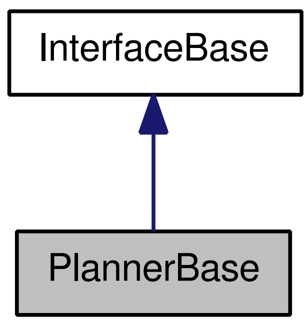
11.41 PlannerBase Class Reference

[interface] Planner interface that generates trajectories for target objects to follow through the environment. **If not specified, method is not multi-thread safe.** See Planner Concepts.

Inheritance diagram for PlannerBase:



Collaboration diagram for PlannerBase:



Classes

- class PlannerParameters

  Describes a common and serializable interface for planning parameters.

- class PlannerProgress

  Planner progress information passed to each callback function.

Public Types

- typedef boost::function< PlannerAction(const PlannerProgress &)> PlanCallbackFn

  Callback function during planner execute.

Public Member Functions

- virtual bool InitPlan (RobotBasePtr robot, PlannerParametersConstPtr params)=0

    Setup scene, robot, and properties of the plan, and reset all internal structures.

- virtual bool InitPlan (RobotBasePtr robot, std::istream &isParameters)

    Setup scene, robot, and properties of the plan, and reset all structures with pparams.

- virtual PlannerStatus PlanPath (TrajectoryBasePtr ptraj)=0

    Executes the main planner trying to solve for the goal condition.

- virtual PlannerStatus PlanPath (TrajectoryBasePtr ptraj, boost::shared_ptr< std::ostream > pOutStream) RAVE_DEPRECATED
- virtual PlannerParametersConstPtr GetParameters () const =0

    return the internal parameters of the planner

- virtual UserDataPtr RegisterPlanCallback (const PlanCallbackFn &callbackfn)

    register a function that is called periodically during the plan loop.

Static Public Member Functions

- static InterfaceType GetInterfaceTypeStatic ()

Protected Member Functions

- virtual PlannerStatus _ProcessPostPlanners (RobotBasePtr probot, TrajectoryBasePtr ptraj)

    Calls a planner to optimizes the trajectory path.

- virtual PlannerAction _CallCallbacks (const PlannerProgress &progress)

    Calls the registered callbacks in order and returns immediately when an action other than PA_None is returned.

11.41.1 Detailed Description

[interface] Planner interface that generates trajectories for target objects to follow through the environment. **If not specified, method is not multi-thread safe.** See Planner Concepts.

11.41.2 Member Typedef Documentation

11.41.2.1 typedef boost::function<PlannerAction(const PlannerProgress&)> PlanCallbackFn

Callback function during planner execute.

Parameters

progress planner progress information

11.41.3 Member Function Documentation

11.41.3.1 virtual PlannerAction _CallCallbacks (const PlannerProgress & *progress*) [protected, virtual]

Calls the registered callbacks in order and returns immediately when an action other than PA_None is returned.

Parameters

progress planner progress information

11.41.3.2 virtual PlannerStatus _ProcessPostPlanners (RobotBasePtr *probot*, TrajectoryBasePtr *ptraj*) [protected, virtual]

Calls a planner to optimizes the trajectory path.

The PlannerParameters structure passed into the optimization planner is constructed with the same freespace constraints as this planner. This function should always be called in PlanPath to post-process the trajectory.

Parameters

probot the robot this trajectory is meant for, also uses the robot for checking collisions.

ptraj Initial trajectory to be smoothed is inputted. If optimization path succeeds, final trajectory output is set in this variable. The trajectory is for the configuration degrees of freedom defined by the planner parameters.

11.41.3.3 static InterfaceType GetInterfaceTypeStatic () `[static]`

Returns

the static interface type this class points to (used for safe casting)

11.41.3.4 virtual bool InitPlan (RobotBasePtr *robot*, std::istream & *isParameters*) `[virtual]`

Setup scene, robot, and properties of the plan, and reset all structures with pparams.

Parameters

robot main robot to be used for planning

isParameters The serialized form of the parameters. By default, this exists to allow third parties to pass information to planners without excplicitly knowning the format/internal structures used

Returns

true if plan is initialized successfully and initial conditions are satisfied.

11.41.3.5 virtual bool InitPlan (RobotBasePtr *robot*, PlannerParametersConstPtr *params*) `[pure virtual]`

Setup scene, robot, and properties of the plan, and reset all internal structures.

Parameters

robot main robot to be used for planning

params The parameters of the planner, any class derived from PlannerParameters can be passed. The planner should copy these parameters for future instead of storing the pointer.

11.41.3.6 virtual PlannerStatus PlanPath (TrajectoryBasePtr *ptraj*, boost::shared_ptr< std::ostream > *pOutStream*) `[virtual]`

11.41.3.7 virtual PlannerStatus PlanPath (TrajectoryBasePtr *ptraj*) `[pure virtual]`

Executes the main planner trying to solve for the goal condition.

Fill ptraj with the trajectory of the planned path that the robot needs to execute

Parameters

ptraj The output trajectory the robot has to follow in order to successfully complete the plan. If this planner is a path optimizer, the trajectory can be used as an input for generating a smoother path. The trajectory is for the configuration degrees of freedom defined by the planner parameters.

Returns

the status that the planner returned in.

11.41.3.8 virtual UserDataPtr RegisterPlanCallback (const PlanCallbackFn & *callbackfn*) `[virtual]`

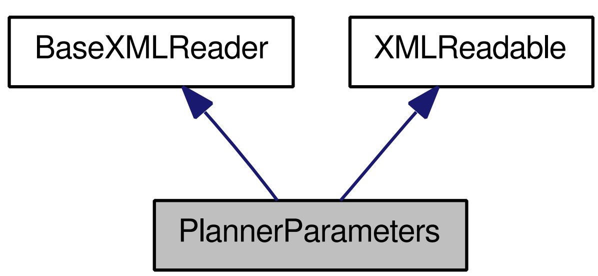
register a function that is called periodically during the plan loop.

Allows the calling process to control the behavior of the planner from a high-level perspective

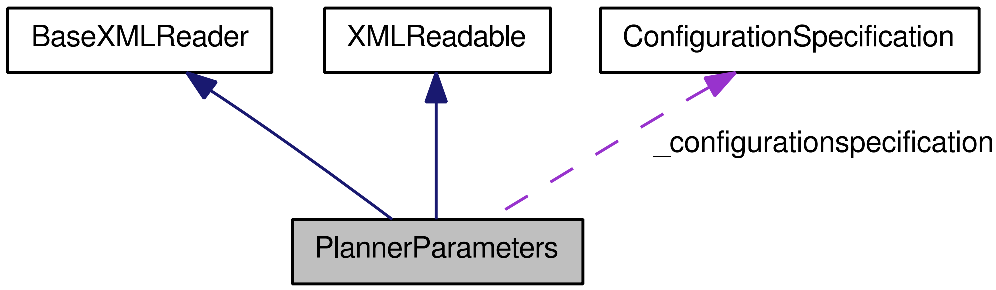
11.42 PlannerParameters Class Reference

Describes a common and serializable interface for planning parameters.

Inheritance diagram for PlannerParameters:



Collaboration diagram for PlannerParameters:



Public Types

- typedef boost::function< dReal(const std::vector< dReal > &)> CostFn

  Cost function on the state pace (optional).

- typedef boost::function< dReal(const std::vector< dReal > &)> GoalFn

  Goal heuristic function.(optional).

- typedef boost::function< dReal(const std::vector< dReal > &, const std::vector< dReal > &)> DistMetricFn

  Distance metric between configuration spaces (optional).

- typedef boost::function< bool(const std::vector< dReal > &, const std::vector< dReal > &, IntervalType, PlannerBase::ConfigurationListPtr)> CheckPathConstraintFn

  Checks that all the constraints are satisfied between two configurations.

- typedef boost::function< bool(std::vector< dReal > &)> SampleFn

  Samples a random configuration (mandatory).

- typedef boost::function< bool(std::vector< dReal > &)> SampleGoalFn

  Samples a valid goal configuration (optional).

- typedef boost::function< bool(std::vector< dReal > &)> SampleInitialFn

  Samples a valid initial configuration (optional).

- typedef boost::function< bool(std::vector< dReal > &, const std::vector< dReal > &, dReal)> SampleNeighFn

  Returns a random configuration around a neighborhood (optional).

- typedef boost::function< void(const std::vector< dReal > &)> SetStateFn

  Sets the state of the robot. Default is active robot joints (mandatory).

- typedef boost::function< void(std::vector< dReal > &)> GetStateFn

  Gets the state of the robot. Default is active robot joints (mandatory).

- typedef boost::function< void(std::vector< dReal > &, const std::vector< dReal > &)> DiffStateFn

  Computes the difference of two states.

- typedef boost::function< bool(std::vector< dReal > &, const std::vector< dReal > &, int)> NeighStateFn

  Adds a delta state to a curent state, acting like a next-nearest-neighbor function along a given direction.

Public Member Functions

- virtual PlannerParameters & operator= (const PlannerParameters &r)

  Attemps to copy data from one set of parameters to another in the safest manner.

- virtual void SetRobotActiveJoints (RobotBasePtr robot)

  sets up the planner parameters to use the active joints of the robot

- virtual int GetDOF () const

  Return the degrees of freedom of the planning configuration space.

Public Attributes

- ConfigurationSpecification _configurationspecification

  the configuration specification in which the planner works in. This specification is passed to the trajecotry creation modules.

- std::vector< dReal > vinitialconfig
- std::vector< dReal > _vConfigLowerLimit

  the absolute limits of the configuration space.

- std::vector< dReal > _vConfigVelocityLimit

  the absolute velocity limits of each DOF of the configuration space.

- std::vector< dReal > _vConfigAccelerationLimit *the absolute acceleration limits of each DOF of the configuration space.*

- std::vector< dReal > _vConfigResolution

  the discretization resolution of each dimension of the configuration space

- dReal _fStepLength

  a discretization between the path that connects two configurations

- int _nMaxIterations

  maximum number of iterations before the planner gives up. If 0 or less, planner chooses best iterations.

- std::string _sPostProcessingPlanner

  Specifies the planner that will perform the post-processing path smoothing before returning.

- std::string _sPostProcessingParameters

  The serialized planner parameters to pass to the path optimizer.

- std::string _sExtraParameters

  Extra parameters data that does not fit within this planner parameters structure, but is still important not to lose all the information.

Protected Member Functions

- virtual bool serialize (std::ostream &O) const

  output the planner parameters in a string (in XML format) don't use PlannerParameters as a tag!

Friends

- OPENRAVE_API std::ostream & operator<< (std::ostream &O, const PlannerParameters &v)

  outputs the data and surrounds it with

- OPENRAVE_API std::istream & operator>> (std::istream &I, PlannerParameters &v)

  expects

- std::stringstream _ss

  holds the data read by characters

- std::vector< std::string > _vXMLParameters

  all the top-level XML parameter tags (lower case) that are handled by this parameter structure, should be registered in the constructor

11.42.1 Detailed Description

Describes a common and serializable interface for planning parameters. The class is serializable to XML, so can be loaded from file or passed around the network. If extra parameters need to be specified, derive from this class and

- add the extra tags to PlannerParameters::_vXMLParameters
- override PlannerParameters::startElement and PlannerParameters::endElement for processing
- possibly override the PlannerParameters::characters

Also allows the parameters and descriptions to be serialized to reStructuredText for documentation purposes.

Examples:

orplanning_planner.cpp.

11.42.2 Member Typedef Documentation

11.42.2.1 typedef boost::function<bool (const std::vector<dReal>&, const std::vector<dReal>&, IntervalType, PlannerBase::ConfigurationListPtr)> CheckPathConstraintFn

Checks that all the constraints are satisfied between two configurations.

The simplest and most fundamental constraint is line-collision checking. The robot goes from q0 to q1.

success = _checkpathconstraints(q0,q1,interval,configurations)

When called, q0 is guaranteed to be set on the robot. The function returns true if the path to q1 satisfies all the constraints of the planner. If q0==q1, and interval==IT_OpenStart or IT_OpenEnd, then only one configuration should be checked. It is recommended to use IT_OpenStart. Because this function can internally use neighstatefn, need to make sure that Q0->Q1 is going from initial to goal direction.

Parameters

- ***q0*** is the configuration the robot is coming from (currently set).
- ***q1*** is the configuration the robot should move to.
- ***interval*** Specifies whether to check the end points of the interval for constraints
- ***configurations*** Optional argument that will hold the intermediate configuraitons checked between q0 and q1 configurations. The appended configurations will be all valid and in free space. They are appended after the items already stored on the list.

11.42.2.2 typedef boost::function<dReal(const std::vector<dReal>&)> CostFn

Cost function on the state pace (optional).

cost = _costfn(config)

Parameters

- ***cost*** the cost of being in the current state

11.42.2.3 typedef boost::function<void (std::vector<dReal>&,const std::vector<dReal>&)> DiffStateFn

Computes the difference of two states.

_diffstatefn(q1,q2) -> q1 -= q2

An explicit difference function is necessary for correct interpolation when there are circular joints. Default is regular subtraction.

11.42.2.4 typedef boost::function<dReal(const std::vector<dReal>&, const std::vector<dReal>&)> DistMetricFn

Distance metric between configuration spaces (optional).

distmetric(config1,config2)

Two configurations are considered the same when function returns 0.

11.42.2.5 typedef boost::function<dReal(const std::vector<dReal>&)> GoalFn

Goal heuristic function.(optional).

distance = _goalfn(config)

Goal is complete when returns 0

Parameters

distance - distance to closest goal

11.42.2.6 typedef boost::function<bool (std::vector<dReal>&,const std::vector<dReal>&, int)> NeighStateFn

Adds a delta state to a curent state, acting like a next-nearest-neighbor function along a given direction.

success = _neighstatefn(q,qdelta,fromgoal) -> q = Filter(q+qdelta)

Parameters

q the current state

qdelta the delta to add

fromgoal 1 if q is coming from a goal state, 0 if it is coming from an initial state

In RRTs this is used for the extension operation. The new state is stored in the first parameter q. Note that the function can also add a filter to the final destination (like projecting onto a constraint manifold).

11.42.2.7 typedef boost::function<bool (std::vector<dReal>&)> SampleFn

Samples a random configuration (mandatory).

The dimension of the returned sample is the dimension of the configuration space. success = samplefn(newsample)

11.42.2.8 typedef boost::function<bool (std::vector<dReal>&)> SampleGoalFn

Samples a valid goal configuration (optional).

If valid, the function should be called at every iteration. Any type of sampling probabilities and conditions can be encoded inside the function. The dimension of the returned sample is the dimension of the configuration space. success = samplegoalfn(newsample)

11.42.2.9 typedef boost::function<bool (std::vector<dReal>&)> SampleInitialFn

Samples a valid initial configuration (optional).

If valid, the function should be called at every iteration. Any type of sampling probabilities and conditions can be encoded inside the function. The dimension of the returned sample is the dimension of the configuration space. success = sampleinitialfn(newsample)

11.42.2.10 typedef boost::function<bool (std::vector<dReal>&, const std::vector<dReal>&, dReal)> SampleNeighFn

Returns a random configuration around a neighborhood (optional).

_sampleneighfn(newsample,pCurSample,fRadius)

Parameters

pCurSample - the neighborhood to sample around

fRadius - specifies the max distance of sampling. The higher the value, the farther the samples will go The distance metric can be arbitrary, but is usually PlannerParameters::pdistmetric.

Returns

if sample was successfully generated return true, otherwise false

11.42.3 Member Function Documentation

11.42.3.1 virtual PlannerParameters& operator= (const PlannerParameters & *r*) `[virtual]`

Attemps to copy data from one set of parameters to another in the safest manner.

First serializes the data of the right hand into a string, then initializes the current parameters via >> pointers to functions are copied directly

11.42.4 Friends And Related Function Documentation

11.42.4.1 OPENRAVE_API std::ostream& operator<< (std::ostream & *O*, const PlannerParameters & *v*) [friend]

outputs the data and surrounds it with

```
<PlannerParameters>
```

tags

11.42.4.2 OPENRAVE_API std::istream& operator>> (std::istream & *I*, PlannerParameters & *v*) [friend]

expects

```
<PlannerParameters>
```

to be the first token. Parses stream until

```
</PlannerParameters>
```

reached

11.42.5 Member Data Documentation

11.42.5.1 dReal _fStepLength

a discretization between the path that connects two configurations

This length represents how dense the samples get distributed across the configuration space. It represents the maximum distance between neighbors when adding new configuraitons. If 0 or less, planner chooses best step length.

11.42.5.2 std::string _sPostProcessingParameters

The serialized planner parameters to pass to the path optimizer.

For example: std::stringstream(_sPostProcessingParameters) >> _parameters;

11.42.5.3 std::string _sPostProcessingPlanner

Specifies the planner that will perform the post-processing path smoothing before returning.

If empty, will not path smooth the returned trajectories (used to measure algorithm time)

11.42.5.4 std::vector<dReal> vinitialconfig

to specify multiple initial or goal configurations, put them into the vector in series (note: not all planners support multiple goals)

11.43 PlannerProgress Class Reference

Planner progress information passed to each callback function.

11.43.1 Detailed Description

Planner progress information passed to each callback function.

11.44 PLUGININFO Class Reference

Holds all the OpenRAVE-specific information provided by a plugin.

Public Attributes

- std::map< InterfaceType, std::vector< std::string > > interfacenames

    offered interfaces

- int version

    OPENRAVE_VERSION.

11.44.1 Detailed Description

Holds all the OpenRAVE-specific information provided by a plugin. PLUGININFO has a hash computed for it to validate its size and type before having a plugin fill it.

Examples:

customreader.cpp, and plugincpp.cpp.

11.45 Point Class Reference

11.45.1 Detailed Description

11.46 RaveCameraIntrinsics< T > Class Template Reference

intrinsic parameters for a camera.

Public Attributes

- std::string distortion_model

  distortion model of the camera. if left empty, no distortion model is used.

- std::vector< T > distortion_coeffs

  coefficients of the distortion model

- T focal_length

  physical focal length distance since focal length cannot be recovered from the intrinsic matrix, but is necessary for determining the lens plane.

11.46.1 Detailed Description

template<typename T> class OpenRAVE::geometry::RaveCameraIntrinsics< T >

intrinsic parameters for a camera.

11.46.2 Member Data Documentation

11.46.2.1 std::string distortion_model

distortion model of the camera. if left empty, no distortion model is used.

Possible values are:

- "plumb_bob" - Brown. "Decentering Distortion of Lenses", Photometric Engineering, pages 444-462, Vol. 32, No. 3, 1966

11.47 RaveTransform< T > Class Template Reference

Affine transformation parameterized with quaterions.

Public Member Functions

- RaveVector< T > operator* (const RaveVector< T > &r) const
  transform a 3 dim vector
- RaveVector< T > rotate (const RaveVector< T > &r) const
  transform a vector by the rotation component only
- RaveTransform< T > rotate (const RaveTransform< T > &r) const
  transform a transform by the rotation component only
- RaveTransform< T > operator* (const RaveTransform< T > &r) const
  *t = this * r*

Public Attributes

- RaveVector< T > trans
  *rot is a quaternion=(cos(ang/2),axisx*sin(ang/2),axisy*sin(ang/2),axisz*sin(ang/2))*

11.47.1 Detailed Description

template<typename T> class OpenRAVE::geometry::RaveTransform< T >

Affine transformation parameterized with quaterions.

11.48 RaveTransformMatrix< T > Class Template Reference

Affine transformation parameterized with rotation matrices. Scales and shears are not supported.

Public Member Functions

- RaveTransformMatrix< T > operator∗ (const RaveTransformMatrix< T > &r) const

  t = this ∗ r

- RaveTransformMatrix< T > inverse () const

  being on the safe side, do the full inverse incase someone uses scaling.

Public Attributes

- T m [12]
- RaveVector< T > trans

  translation component

Friends

- template<typename U >
  std::ostream & operator<< (std::ostream &O, const RaveTransformMatrix< U > &v)

  serialize in column order! This is the format transformations are passed across the network

- template<typename U >
  std::istream & operator>> (std::istream &I, RaveTransformMatrix< U > &v)

  de-serialize in column order! This is the format transformations are passed across the network

11.48.1 Detailed Description

template<typename T> class OpenRAVE::geometry::RaveTransformMatrix< T >

Affine transformation parameterized with rotation matrices. Scales and shears are not supported.

11.48.2 Member Data Documentation

11.48.2.1 T m[12]

3x3 rotation matrix. Note that each row is 4 elements long! So row 1 starts at m[4], row 2 at m[8] The reason is to maintain 16 byte alignment when sizeof(T) is 4 bytes

11.49 RaveVector< T > Class Template Reference

Vector class containing 4 dimensions.

Public Member Functions

- template<typename U >
  RaveVector (const U ∗pf)

  note, it only copes 3 values!

- RaveVector< T > cross (const RaveVector< T > &v) const

  3 dim cross product, w is not touched

- template<typename U >
  RaveVector< T > operator^ (const RaveVector< U > &v) const

  cross product operator

11.49.1 Detailed Description

template<typename T> class OpenRAVE::geometry::RaveVector< T >

Vector class containing 4 dimensions. It is better to use this for a 3 dim vector because it is 16byte aligned and SIMD instructions can be used

11.50 ray< T > Class Template Reference

A ray defined by an origin and a direction.

11.50.1 Detailed Description

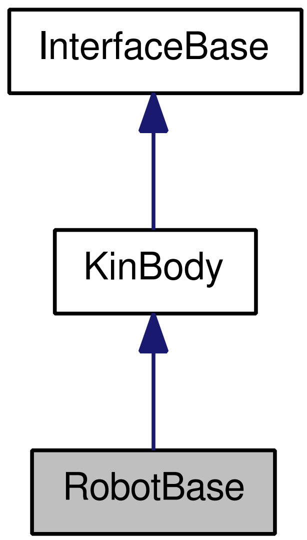
template<typename T> class OpenRAVE::geometry::ray< T >

A ray defined by an origin and a direction.

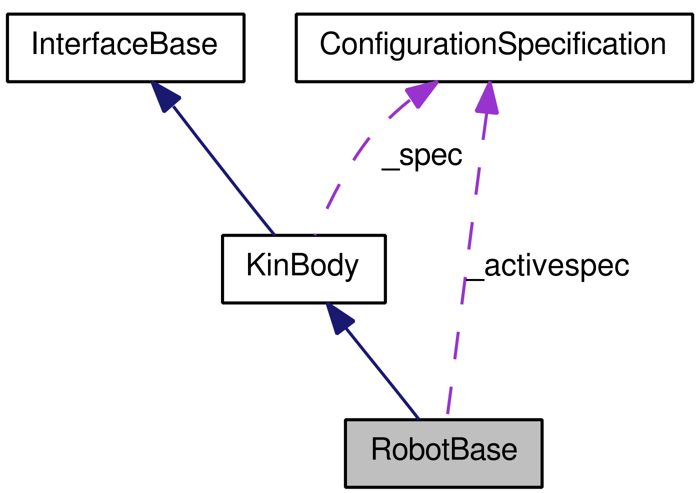
11.51 RobotBase Class Reference

[interface] A robot is a kinematic body that has attached manipulators, sensors, and controllers. **If not specified, method is not multi-thread safe.** See Robot Concepts.

Inheritance diagram for RobotBase:



Collaboration diagram for RobotBase:



Classes

- class AttachedSensor

  Attaches a sensor to a link on the robot.

- class Grabbed

  The information of a currently grabbed body.

- class Manipulator

  Defines a chain of joints for an arm and set of joints for a gripper. Simplifies operating with them.

- class RobotStateSaver

  Helper class derived from KinBodyStateSaver to additionaly save robot information.

Public Member Functions

- virtual bool InitFromFile (const std::string &filename, const AttributesList &atts=AttributesList()) RAVE_DEPRECATED
- virtual bool InitFromData (const std::string &data, const AttributesList &atts=AttributesList()) RAVE_DEPRECATED
- virtual std::vector< ManipulatorPtr > & GetManipulators ()

  Returns the manipulators of the robot.

- virtual void SetTransform (const Transform &trans)

  Transforms the robot and updates the attached sensors and grabbed bodies.

- virtual void SimulationStep (dReal fElapsedTime)

  Simulate the robot and update the grabbed bodies and attached sensors.

- virtual bool CheckSelfCollision (CollisionReportPtr report=CollisionReportPtr()) const

  Check if body is self colliding. Links that are joined together are ignored.

- virtual bool CheckLinkCollision (int ilinkindex, const Transform &tlinktrans, CollisionReportPtr report=CollisionReportPtr())

  checks collision of a robot link with the surrounding environment. Attached/Grabbed bodies to this link are also checked for collision.

- virtual void Clone (InterfaceBaseConstPtr preference, int cloningoptions)

  does not clone the grabbed bodies since it requires pointers from other bodies (that might not be initialized yet)

- virtual bool IsRobot () const
- virtual const std::string & GetRobotStructureHash () const
- virtual ControllerBasePtr GetController () const

  gets the robot controller

- virtual bool SetController (ControllerBasePtr controller, const std::vector< int > &dofindices, int nControlTransformation)

  set a controller for a robot

- virtual bool SetController (ControllerBasePtr controller, const std::string &args) RAVE_DEPRECATED
- void GetFullTrajectoryFromActive (TrajectoryBasePtr pfulltraj, TrajectoryBaseConstPtr pActiveTraj, bool bOverwriteTransforms=true) RAVE_DEPRECATED

Grabbing Bodies

A grabbed body becomes part of the robot and its relative pose with respect to a robot's link will be fixed. KinBody::_AttachBody is called for every grabbed body in order to make the grabbed body a part of the robot. Once grabbed, the inter-collisions between the robot and the body are regarded as self-collisions; any outside collisions of the body and the environment are regarded as environment collisions with the robot.

- virtual bool Grab (KinBodyPtr body, LinkPtr pRobotLinkToGrabWith, const std::set< int > &setRobotLinksToIgnore)

  Grab the body with the specified link.

- virtual bool Grab (KinBodyPtr body, LinkPtr pRobotLinkToGrabWith)

  Grab a body with the specified link.

- virtual bool Grab (KinBodyPtr body, const std::set< int > &setRobotLinksToIgnore)

  Grabs the body with the active manipulator's end effector.

- virtual bool Grab (KinBodyPtr body)

  Grabs the body with the active manipulator's end effector.

- virtual void Release (KinBodyPtr body)

  Release the body if grabbed.

- virtual void ReleaseAllGrabbed ()

  Release all grabbed bodies.

- virtual void RegrabAll ()

- virtual LinkPtr IsGrabbing (KinBodyConstPtr body) const
  return the robot link that is currently grabbing the body. If the body is not grabbed, will return an empty pointer.
- virtual void GetGrabbed (std::vector< KinBodyPtr > &vbodies) const
  gets all grabbed bodies of the robot

Static Public Member Functions

- static InterfaceType GetInterfaceTypeStatic ()
  Return the static interface type this class points to (used for safe casting).

Protected Member Functions

- virtual void _ComputeInternalInformation ()
  Proprocess the manipulators and sensors and build the specific robot hashes.
- virtual void _ParametersChanged (int parameters)
  Called to notify the body that certain groups of parameters have been changed.

Protected Attributes

- std::vector< Grabbed > _vGrabbedBodies
  vector of grabbed bodies
- std::vector< ManipulatorPtr > _vecManipulators
- int _nActiveManip
- std::vector< AttachedSensorPtr > _vecSensors
- int _nActiveDOF
  Active degrees of freedom; if -1, use robot dofs.
- int _nAffineDOFs
  dofs describe what affine transformations are allowed
- Vector _vRotationAxisLowerLimits
  the xyz components are used if the rotation axis is solely about X,Y,or Z; otherwise the W component is used.

Affine DOFs

Methods using the active degrees of freedoms of the robot. Active DOFs are a way for the user to specify degrees of freedom of interest for a current execution block. All planners by default use the robot's active DOF and active manipultor. For every Get∗ method, there is a corresponding GetActive∗ method rather than the methods when setting joints. The active DOFs also include affine transfomrations of the robot's base. Affine transformation DOFs can be found after the joint DOFs in this order: X, Y, Z, Rotation where rotation can be around a specified axis a full 3D rotation. Usually the affine transforamtion is with respect to the first link in the body

- static const DOFAffine DOF_NoTransform RAVE_DEPRECATED = OpenRAVE::DOF_NoTransform
- static const DOFAffine DOF_X **RAVE_DEPRECATED** = OpenRAVE::DOF_-X
- static const DOFAffine DOF_Y **RAVE_DEPRECATED** = OpenRAVE::DOF_-Y
- static const DOFAffine DOF_Z **RAVE_DEPRECATED** = OpenRAVE::DOF_Z
- static const DOFAffine DOF_RotationAxis **RAVE_DEPRECATED** = OpenRAVE::DOF_RotationAxis
- static const DOFAffine DOF_Rotation3D **RAVE_DEPRECATED** = OpenRAVE::DOF_Rotation3D
- static const DOFAffine DOF_RotationQuat **RAVE_DEPRECATED** = OpenRAVE::DOF_RotationQuat
- virtual void SetActiveDOFs (const std::vector< int > &dofindices, int affine=OpenRAVE::DOF_NoTransform)

  Set the joint indices and affine transformation dofs that the planner should use. If DOF_RotationAxis is specified, the previously set axis is used.

- virtual void SetActiveDOFs (const std::vector< int > &dofindices, int affine, const Vector &rotationaxis)

  Set the joint indices and affine transformation dofs that the planner should use. If DOF_RotationAxis is specified, then rotationaxis is set as the new axis.

- virtual int **GetActiveDOF** () const
- virtual int **GetAffineDOF** () const
- virtual int GetAffineDOFIndex (DOFAffine dof) const
- virtual const ConfigurationSpecification & GetActiveConfigurationSpecification () const

  return the configuration specification of the active dofs

- virtual const std::vector< int > & GetActiveDOFIndices () const

  Return the set of active dof indices of the joints.

- virtual Vector **GetAffineRotationAxis** () const
- virtual void **SetAffineTranslationLimits** (const Vector &lower, const Vector &upper)
- virtual void **SetAffineRotationAxisLimits** (const Vector &lower, const Vector &upper)
- virtual void **SetAffineRotation3DLimits** (const Vector &lower, const Vector &upper)
- virtual void SetAffineRotationQuatLimits (const Vector &quatangle)

  sets the quaternion limits using a starting rotation and the max angle deviation from it.

- virtual void **SetAffineTranslationMaxVels** (const Vector &vels)
- virtual void **SetAffineRotationAxisMaxVels** (const Vector &vels)
- virtual void **SetAffineRotation3DMaxVels** (const Vector &vels)
- virtual void **SetAffineRotationQuatMaxVels** (dReal vels)
- virtual void **SetAffineTranslationResolution** (const Vector &resolution)
- virtual void **SetAffineRotationAxisResolution** (const Vector &resolution)
- virtual void **SetAffineRotation3DResolution** (const Vector &resolution)
- virtual void **SetAffineRotationQuatResolution** (dReal resolution)
- virtual void **SetAffineTranslationWeights** (const Vector &weights)
- virtual void **SetAffineRotationAxisWeights** (const Vector &weights)
- virtual void **SetAffineRotation3DWeights** (const Vector &weights)
- virtual void **SetAffineRotationQuatWeights** (dReal weights)
- virtual void **GetAffineTranslationLimits** (Vector &lower, Vector &upper) const
- virtual void **GetAffineRotationAxisLimits** (Vector &lower, Vector &upper) const
- virtual void **GetAffineRotation3DLimits** (Vector &lower, Vector &upper) const
- virtual Vector GetAffineRotationQuatLimits () const

  gets the quaternion limits

- virtual Vector **GetAffineTranslationMaxVels** () const
- virtual Vector **GetAffineRotationAxisMaxVels** () const
- virtual Vector **GetAffineRotation3DMaxVels** () const
- virtual dReal **GetAffineRotationQuatMaxVels** () const
- virtual Vector **GetAffineTranslationResolution** () const
- virtual Vector **GetAffineRotationAxisResolution** () const
- virtual Vector **GetAffineRotation3DResolution** () const
- virtual dReal **GetAffineRotationQuatResolution** () const
- virtual Vector **GetAffineTranslationWeights** () const
- virtual Vector **GetAffineRotationAxisWeights** () const

- virtual Vector **GetAffineRotation3DWeights** () const
- virtual dReal **GetAffineRotationQuatWeights** () const
- virtual void **SetActiveDOFValues** (const std::vector< dReal > &values, bool bCheckLimits=false)
- virtual void **GetActiveDOFValues** (std::vector< dReal > &v) const
- virtual void **SetActiveDOFVelocities** (const std::vector< dReal > &velocities, bool bCheckLimits=false)
- virtual void **GetActiveDOFVelocities** (std::vector< dReal > &velocities) const

- virtual void **GetActiveDOFLimits** (std::vector< dReal > &lower, std::vector< dReal > &upper) const
- virtual void **GetActiveDOFResolutions** (std::vector< dReal > &v) const
- virtual void **GetActiveDOFWeights** (std::vector< dReal > &v) const
- virtual void **GetActiveDOFVelocityLimits** (std::vector< dReal > &v) const
- virtual void **GetActiveDOFAccelerationLimits** (std::vector< dReal > &v) const
- virtual void **GetActiveDOFMaxVel** (std::vector< dReal > &v) const
- virtual void **GetActiveDOFMaxAccel** (std::vector< dReal > &v) const
- virtual void SubtractActiveDOFValues (std::vector< dReal > &q1, const std::vector< dReal > &q2) const

  computes the configuration difference q1-q2 and stores it in q1. Takes into account joint limits and circular joints

- virtual void SetActiveManipulator (int index)
- virtual void SetActiveManipulator (const std::string &manipname)
- virtual ManipulatorPtr **GetActiveManipulator** ()
- virtual ManipulatorConstPtr **GetActiveManipulator** () const
- virtual int GetActiveManipulatorIndex () const
- virtual bool SetMotion (TrajectoryBaseConstPtr ptraj) RAVE_DEPRECATED
- virtual bool SetActiveMotion (TrajectoryBaseConstPtr ptraj) RAVE_-DEPRECATED
- virtual bool SetActiveMotion (TrajectoryBaseConstPtr ptraj, dReal fSpeed) RAVE_DEPRECATED
- virtual void CalculateActiveJacobian (int index, const Vector &offset, boost::multi_array< dReal, 2 > &mjacobian) const

  Calculates the translation jacobian with respect to a link.

- virtual void **CalculateActiveJacobian** (int index, const Vector &offset, std::vector< dReal > &pfJacobian) const
- virtual void **CalculateActiveRotationJacobian** (int index, const Vector &qInitialRot, boost::multi_array< dReal, 2 > &vjacobian) const
- virtual void **CalculateActiveRotationJacobian** (int index, const Vector &qInitialRot, std::vector< dReal > &pfJacobian) const

- virtual void CalculateActiveAngularVelocityJacobian (int index, boost::multi_array< dReal, 2 > &mjacobian) const
- virtual void **CalculateActiveAngularVelocityJacobian** (int index, std::vector< dReal > &pfJacobian) const
- virtual const std::set< int > & **GetNonAdjacentLinks** (int adjacentoptions=0) const

11.51.1 Detailed Description

[interface] A robot is a kinematic body that has attached manipulators, sensors, and controllers. **If not specified, method is not multi-thread safe.** See Robot Concepts.

11.51.2 Member Function Documentation

11.51.2.1 virtual void _ParametersChanged (int *parameters*) `[protected, virtual]`

Called to notify the body that certain groups of parameters have been changed.

This function in calls every registers calledback that is tracking the changes.

Reimplemented from KinBody.

11.51.2.2 virtual void CalculateActiveAngularVelocityJacobian (int *index*, boost::multi_array< dReal, 2 > & *mjacobian*) const `[virtual]`

Calculates the angular velocity jacobian of a specified link about the axes of world coordinates.

Parameters

index of the link that the rotation is attached to

mjacobian 3x(num ACTIVE DOF) matrix

11.51.2.3 virtual void CalculateActiveJacobian (int *index*, const Vector & *offset*, boost::multi_array< dReal, 2 > & *mjacobian*) const `[virtual]`

Calculates the translation jacobian with respect to a link.

Calculates the partial differentials for the active degrees of freedom that in the path from the root node to _veclinks[index] (doesn't touch the rest of the values).

Parameters

mjacobian a 3 x ActiveDOF matrix

11.51.2.4 virtual bool CheckLinkCollision (int *ilinkindex*, const Transform & *tlinktrans*, CollisionReportPtr *report* = `CollisionReportPtr()`) `[virtual]`

checks collision of a robot link with the surrounding environment. Attached/Grabbed bodies to this link are also checked for collision.

Parameters

← ***ilinkindex*** the index of the link to check

← ***tlinktrans*** The transform of the link to check

→ ***report*** [optional] collision report

11.51.2.5 virtual bool CheckSelfCollision (CollisionReportPtr *report* = `CollisionReportPtr()`) const `[virtual]`

Check if body is self colliding. Links that are joined together are ignored.

Parameters

report [optional] collision report

Reimplemented from KinBody.

11.51.2.6 virtual int GetActiveManipulatorIndex () const `[virtual]`

Returns

index of the current active manipulator

11.51.2.7 virtual int GetAffineDOFIndex (DOFAffine *dof*) const `[virtual]`

11.51.2.8 virtual Vector GetAffineRotationQuatLimits () const `[virtual]`

gets the quaternion limits

Parameters

quatangle quaternion_start ∗ max_angle. acos(q dot quaternion_start) <= max_angle

11.51.2.9 void GetFullTrajectoryFromActive (TrajectoryBasePtr *pfulltraj*, TrajectoryBaseConstPtr *pActiveTraj*, bool *bOverwriteTransforms* = `true`)

11.51.2.10 virtual void GetGrabbed (std::vector< KinBodyPtr > & *vbodies*) const `[virtual]`

gets all grabbed bodies of the robot

Parameters

→ ***vbodies*** filled with the grabbed bodies

11.51.2.11 virtual const std::string& GetRobotStructureHash () const `[virtual]`

A md5 hash unique to the particular robot structure that involves manipulation and sensing components The serialization for the attached sensors will not involve any sensor specific properties (since they can change through calibration)

11.51.2.12 virtual bool Grab (KinBodyPtr *body*) `[virtual]`

Grabs the body with the active manipulator's end effector.

Parameters

← ***body*** the body to be grabbed

Returns

true if successful and body is grabbed

11.51.2.13 virtual bool Grab (KinBodyPtr *body*, const std::set< int > & *setRobotLinksToIgnore*) [virtual]

Grabs the body with the active manipulator's end effector.

Parameters

← ***body*** the body to be grabbed

← ***setRobotLinksToIgnore*** Additional robot link indices that collision checker ignore when checking collisions between the grabbed body and the robot.

Returns

true if successful and body is grabbed

11.51.2.14 virtual bool Grab (KinBodyPtr *body*, LinkPtr *pRobotLinkToGrabWith*) [virtual]

Grab a body with the specified link.

Parameters

← ***body*** the body to be grabbed

← ***pRobotLinkToGrabWith*** the link of this robot that will perform the grab

Returns

true if successful and body is grabbed/

11.51.2.15 virtual bool Grab (KinBodyPtr *body*, LinkPtr *pRobotLinkToGrabWith*, const std::set< int > & *setRobotLinksToIgnore*) [virtual]

Grab the body with the specified link.

Parameters

← ***body*** the body to be grabbed

← ***pRobotLinkToGrabWith*** the link of this robot that will perform the grab

← ***setRobotLinksToIgnore*** Additional robot link indices that collision checker ignore when checking collisions between the grabbed body and the robot.

Returns

true if successful and body is grabbed.

11.51.2.16 virtual bool InitFromData (const std::string & *data*, const AttributesList & *atts* = `AttributesList()`) `[virtual]`

See also

EnvironmentBase::ReadRobotXMLData

Reimplemented from KinBody.

11.51.2.17 virtual bool InitFromFile (const std::string & *filename*, const AttributesList & *atts* = `AttributesList()`) `[virtual]`

See also

EnvironmentBase::ReadRobotXMLFile

Reimplemented from KinBody.

11.51.2.18 virtual LinkPtr IsGrabbing (KinBodyConstPtr *body*) const `[virtual]`

return the robot link that is currently grabbing the body. If the body is not grabbed, will return an empty pointer.

Parameters

← ***body*** the body to check

11.51.2.19 virtual bool IsRobot () const [virtual]

Returns

true if this body is derived from RobotBase

Reimplemented from KinBody.

11.51.2.20 virtual void RegrabAll () [virtual]

Releases and grabs all bodies, has the effect of recalculating all the initial collision with the bodies.

This has the effect of resetting the current collisions any grabbed body makes with the robot into an ignore list.

11.51.2.21 virtual void Release (KinBodyPtr *body*) [virtual]

Release the body if grabbed.

Parameters

body body to release

11.51.2.22 virtual void ReleaseAllGrabbed () [virtual]

Release all grabbed bodies.

release all bodies

11.51.2.23 virtual void SetActiveDOFs (const std::vector< int > & *dofindices*, int *affine*, const Vector & *rotationaxis*) [virtual]

Set the joint indices and affine transformation dofs that the planner should use. If DOF_RotationAxis is specified, then rotationaxis is set as the new axis.

Parameters

dofindices the indices of the original degrees of freedom to use.

affine A bitmask of DOFAffine values

rotationaxis if DOF_RotationAxis is specified, pRotationAxis is used as the new axis

11.51.2.24 virtual void SetActiveDOFs (const std::vector< int > & *dofindices*, int *affine* = `OpenRAVE::DOF_NoTransform`) `[virtual]`

Set the joint indices and affine transformation dofs that the planner should use. If DOF_RotationAxis is specified, the previously set axis is used.

Parameters

dofindices the indices of the original degrees of freedom to use.

affine A bitmask of DOFAffine values

11.51.2.25 virtual void SetActiveManipulator (const std::string & *manipname*) `[virtual]`

sets the active manipulator of the robot

Parameters

manipname manipulator name

11.51.2.26 virtual void SetActiveManipulator (int *index*) `[virtual]`

sets the active manipulator of the robot

Parameters

index manipulator index

11.51.2.27 virtual bool SetActiveMotion (TrajectoryBaseConstPtr *ptraj*, dReal *fSpeed*) `[virtual]`

11.51.2.28 virtual bool SetActiveMotion (TrajectoryBaseConstPtr *ptraj*) `[virtual]`

11.51.2.29 virtual void SetAffineRotationQuatLimits (const Vector & *quatangle*) [virtual]

sets the quaternion limits using a starting rotation and the max angle deviation from it.

Parameters

quatangle quaternion_start ∗ max_angle. acos(q dot quaternion_start) <= max_-angle. If max_angle is 0, then will take the current transform of the robot

11.51.2.30 virtual bool SetController (ControllerBasePtr *controller*, const std::string & *args*) [virtual]

11.51.2.31 virtual bool SetController (ControllerBasePtr *controller*, const std::vector< int > & *dofindices*, int *nControlTransformation*) [virtual]

set a controller for a robot

Parameters

pController - if NULL, sets the controller of this robot to NULL. otherwise attemps to set the controller to this robot.

args - the argument list to pass when initializing the controller

11.51.2.32 virtual bool SetMotion (TrajectoryBaseConstPtr *ptraj*) [virtual]

11.51.2.33 virtual void SimulationStep (dReal *fElapsedTime*) [virtual]

Simulate the robot and update the grabbed bodies and attached sensors.

Do not call SimulationStep for the attached sensors in this function.

Reimplemented from KinBody.

11.51.3 Member Data Documentation

11.51.3.1 int _nActiveManip [protected]

See also

GetActiveManipulatorIndex

11.51.3.2 std::vector<ManipulatorPtr> _vecManipulators [protected]

See also

GetManipulators

11.51.3.3 std::vector<AttachedSensorPtr> _vecSensors [protected]

See also

GetAttachedSensors

11.51.3.4 const DOFAffine DOF_NoTransform RAVE_DEPRECATED = OpenRAVE::DOF_NoTransform [static]

11.52 RobotStateSaver Class Reference

Helper class derived from KinBodyStateSaver to additionaly save robot information.

11.52.1 Detailed Description

Helper class derived from KinBodyStateSaver to additionaly save robot information.

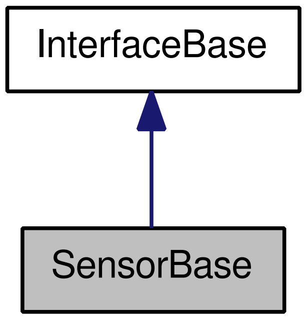
Examples:

ormulticontrol.cpp, orplanning_module.cpp, orplanning_planner.cpp, and ortrajectory.cpp.

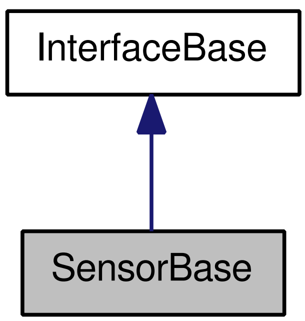
11.53 SensorBase Class Reference

[interface] A sensor measures physical properties from the environment. **If not specified, method is not multi-thread safe.** See Sensor Concepts.

Inheritance diagram for SensorBase:



Collaboration diagram for SensorBase:



Classes

- class ActuatorSensorData

  An actuator for modeling motors and other mechanisms that produce torque/force. The actuator has only one degree of freedom.

- class Force6DSensorData

  Stores force data.

- class IMUSensorData

  Stores IMU data.

- class JointEncoderSensorData

  Stores joint angles and EE position.

- class OdometrySensorData

  odometry data storing full 6D pose and velocity

- class SensorData

  used to pass sensor data around

- class SensorGeometry

  permanent properties of the sensors

- class TactileSensorData

  tactle data

Public Types

- enum ConfigureCommand {

  CC_PowerOn = 0x10, CC_PowerOff = 0x11, CC_PowerCheck = 0x12, CC_-RenderDataOn = 0x20,

  CC_RenderDataOff = 0x21, CC_RenderDataCheck = 0x23, CC_-RenderGeometryOn = 0x30, CC_RenderGeometryOff = 0x31,

  CC_RenderGeometryCheck = 0x32 }

  A set of commands used for run-time sensor configuration.

Public Member Functions

- virtual int Configure (ConfigureCommand command, bool blocking=false)=0

  Configures properties of the sensor like power.

- virtual bool SimulationStep (dReal fTimeElapsed) OPENRAVE_DUMMY_-IMPLEMENTATION

  Simulate one step forward for sensors.

- virtual SensorGeometryPtr GetSensorGeometry (SensorType type=ST_-Invalid)=0

  Returns the sensor geometry. This method is thread safe.

- virtual SensorDataPtr CreateSensorData (SensorType type=ST_Invalid)=0

  Creates the sensor data to be specifically used by this class.
- virtual bool GetSensorData (SensorDataPtr psensordata)=0

  Copy the most recent published data of the sensor given the type.
- virtual bool Supports (SensorType type)=0

  returns true if sensor supports a particular sensor type
- virtual void SetTransform (const Transform &trans)=0

  Set the transform of a sensor (global coordinate system).
- virtual UserDataPtr RegisterDataCallback (SensorType type, const boost::function< void(SensorDataConstPtr)> &callback) OPENRAVE_-DUMMY_IMPLEMENTATION

  Register a callback whenever new sensor data comes in.
- virtual const std::string & GetName () const
- virtual bool Init (const std::string &) RAVE_DEPRECATED

Static Public Member Functions

- static InterfaceType GetInterfaceTypeStatic ()

  return the static interface type this class points to (used for safe casting)

Protected Attributes

- std::string _name

  name of the sensor

11.53.1 Detailed Description

[interface] A sensor measures physical properties from the environment. **If not specified, method is not multi-thread safe.** See Sensor Concepts.

11.53.2 Member Enumeration Documentation

11.53.2.1 enum ConfigureCommand

A set of commands used for run-time sensor configuration.

Enumerator:

CC_PowerOn turns the sensor on, starts gathering data and using processor cycles. If the power is already on, servers as a reset. (off by default)

CC_PowerOff turns the sensor off, stops gathering data (off by default).

CC_PowerCheck returns whether power is on

CC_RenderDataOn turns on any rendering of the sensor data (off by default)

CC_RenderDataOff turns off any rendering of the sensor data (off by default)

CC_RenderDataCheck returns whether data rendering is on

CC_RenderGeometryOn turns on any rendering of the sensor geometry (on by default)

CC_RenderGeometryOff turns off any rendering of the sensor geometry (on by default)

CC_RenderGeometryCheck returns whether geometry rendering is on

11.53.3 Member Function Documentation

11.53.3.1 virtual int Configure (ConfigureCommand *command*, bool *blocking* = `false`) `[pure virtual]`

Configures properties of the sensor like power.

Parameters

type ConfigureCommand

blocking If set to true, makes sure the configuration ends before this function returns.(might cause problems if environment is locked).

Exceptions

openrave_exception if command doesn't succeed

11.53.3.2 virtual SensorDataPtr CreateSensorData (SensorType *type* = `ST_Invalid`) `[pure virtual]`

Creates the sensor data to be specifically used by this class.

Parameters

type the requested sensor type to create. A sensor can support many types. If type is ST_Invalid, then returns a data structure of the type most representative of this sensor.

Returns

new SensorData class

11.53.3.3 virtual const std::string& GetName () const `[virtual]`

Returns

the name of the sensor

11.53.3.4 virtual bool GetSensorData (SensorDataPtr *psensordata*) `[pure virtual]`

Copy the most recent published data of the sensor given the type.

Once GetSensorData returns, the caller has full unrestricted access to the data. This method is thread safe.

Parameters

psensordata A pointer to SensorData returned from CreateSensorData, the plugin will use psensordata->GetType() in order to return the correctly supported type.

11.53.3.5 virtual SensorGeometryPtr GetSensorGeometry (SensorType *type* = `ST_Invalid`) `[pure virtual]`

Returns the sensor geometry. This method is thread safe.

Parameters

type the requested sensor type to create. A sensor can support many types. If type is ST_Invalid, then returns a data structure

Returns

sensor geometry pointer, use delete to destroy it

11.53.3.6 virtual bool Init (const std::string &) `[virtual]`

11.53.3.7 virtual UserDataPtr RegisterDataCallback (SensorType *type*, const boost::function< void(SensorDataConstPtr)> & *callback*) `[virtual]`

Register a callback whenever new sensor data comes in.

Parameters

type the sensor type to register for

callback the user function to call, note that this might block the thread generating/receiving sensor data

11.53.3.8 virtual void SetTransform (const Transform & *trans*) `[pure virtual]`

Set the transform of a sensor (global coordinate system).

Sensors attached to the robot have their transforms automatically set every time the robot is moved

Parameters

trans - The transform defining the frame of the sensor.

11.53.3.9 virtual bool SimulationStep (dReal *fTimeElapsed*) `[virtual]`

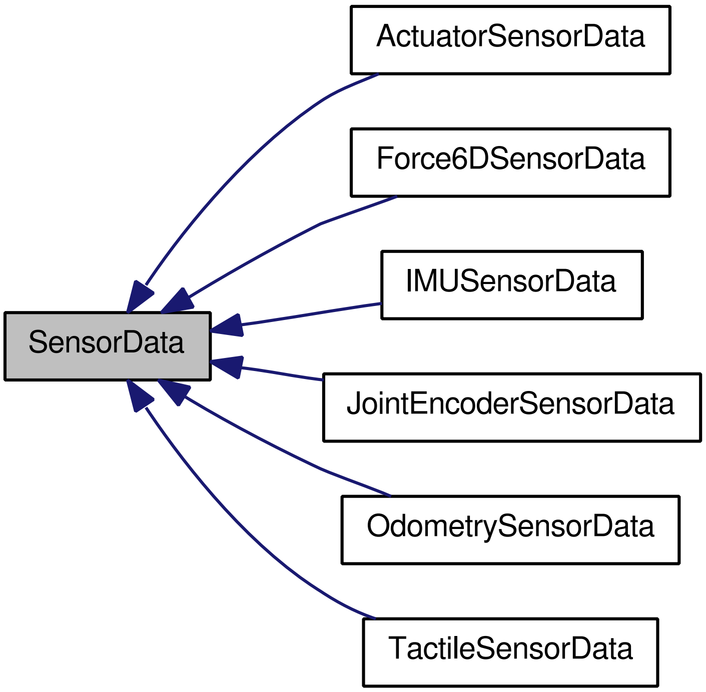
Simulate one step forward for sensors.

Only valid if this sensor is simulation based. A sensor hooked up to a real device can ignore this call

11.54 SensorData Class Reference

used to pass sensor data around

Inheritance diagram for SensorData:



Public Member Functions

- virtual bool serialize (std::ostream &O) const

    Serialize the sensor data to stream in XML format.

Public Attributes

- uint64_t __stamp

    time stamp of the sensor data in microseconds. If 0, then the data is uninitialized! (floating-point precision is bad here). This can be either simulation or real time depending on the sensor.

- Transform __trans

    the coordinate system the sensor was when the measurement was taken, this is taken directly from SensorBase::GetTransform

11.54.1 Detailed Description

used to pass sensor data around

11.55 SensorGeometry Class Reference

permanent properties of the sensors

Inherited by ActuatorGeomData, CameraGeomData, Force6DGeomData, IMUGeomData, JointEncoderGeomData, LaserGeomData, OdometryGeomData, TactileGeomData, and TouchGeomData.

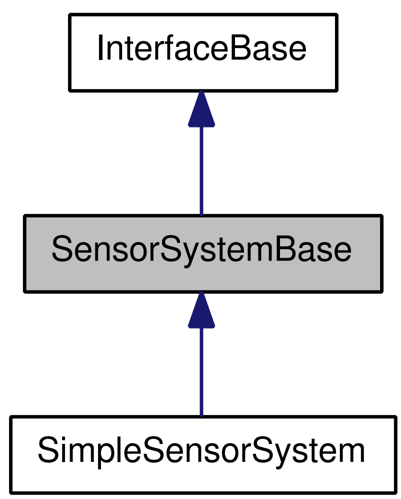
11.55.1 Detailed Description

permanent properties of the sensors

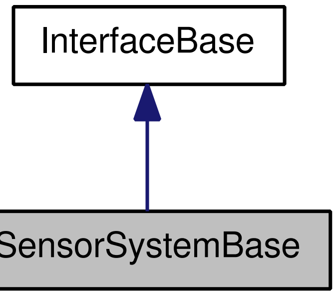
11.56 SensorSystemBase Class Reference

[interface] Used to manage the creation and destruction of bodies. See Sensor System Concepts.

Inheritance diagram for SensorSystemBase:



Collaboration diagram for SensorSystemBase:



Public Member Functions

- virtual void Reset ()=0

  resets the system and stops managing all objects. Any objects that are not locked, are deleted

- virtual void AddRegisteredBodies (const std::vector< KinBodyPtr > &vbodies)=0

  automatically register bodies that have some type of SensorSystem data (usually done through xml)

- virtual KinBody::ManageDataPtr AddKinBody (KinBodyPtr pbody, XMLReadableConstPtr pdata)=0
- virtual bool RemoveKinBody (KinBodyPtr pbody)=0

  remove body from sensory system. If bDestroy is true, will also deallocate the memory

- virtual bool IsBodyPresent (KinBodyPtr pbody)=0

  returns true if body is present

- virtual bool EnableBody (KinBodyPtr pbody, bool bEnable)=0

  enable/disable a body from being updated by the sensor system

- virtual bool SwitchBody (KinBodyPtr pbody1, KinBodyPtr pbody2)=0

Static Public Member Functions

- static InterfaceType GetInterfaceTypeStatic ()

  return the static interface type this class points to (used for safe casting)

11.56.1 Detailed Description

[interface] Used to manage the creation and destruction of bodies. See Sensor System Concepts.

11.56.2 Member Function Documentation

11.56.2.1 virtual KinBody::ManageDataPtr AddKinBody (KinBodyPtr *pbody*, XMLReadableConstPtr *pdata*) `[pure virtual]`

add body for registering with sensor system pdata is a pointer to a data structor holding tracking/registration information for the system

11.56.2.2 virtual bool SwitchBody (KinBodyPtr *pbody1*, KinBodyPtr *pbody2*) `[pure virtual]`

switches the registrations of two bodies. Can be used to quickly change the models of the current bodies

Parameters

pbody1 First body to switch

pbody2 Second body to switch

11.57 SimpleDistanceMetric Class Reference

simple distance metric based on joint weights

11.57.1 Detailed Description

simple distance metric based on joint weights

11.58 SimpleNeighborhoodSampler Class Reference

samples the neighborhood of a configuration using the configuration space distance metric and sampler.

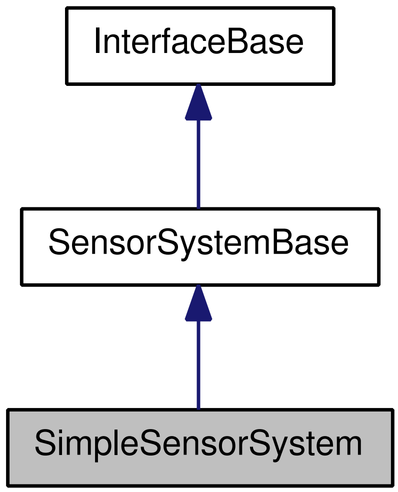
11.58.1 Detailed Description

samples the neighborhood of a configuration using the configuration space distance metric and sampler.

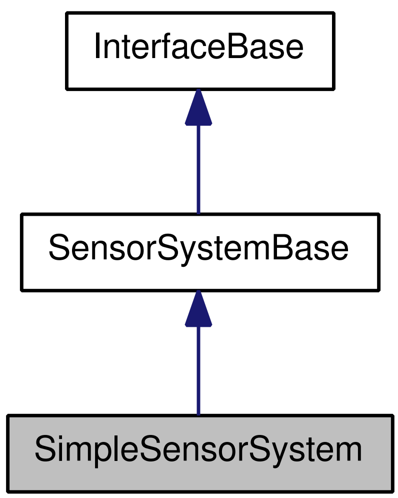
11.59 SimpleSensorSystem Class Reference

A very simple sensor system example that manages raw detection data.

Inheritance diagram for SimpleSensorSystem:



Collaboration diagram for SimpleSensorSystem:



Static Public Member Functions

- static UserDataPtr RegisterXMLReaderId (EnvironmentBasePtr penv, const std::string &xmlid)

  registers the XML reader, do not call in the constructor of this class!

Static Protected Member Functions

- static BaseXMLReaderPtr CreateXMLReaderId (const std::string &xmlid, InterfaceBasePtr ptr, const AttributesList &atts)

  creates a reader to parse the data

Protected Attributes

- uint64_t _expirationtime

  expiration time in us

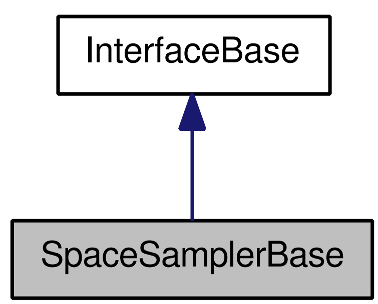
11.59.1 Detailed Description

A very simple sensor system example that manages raw detection data.

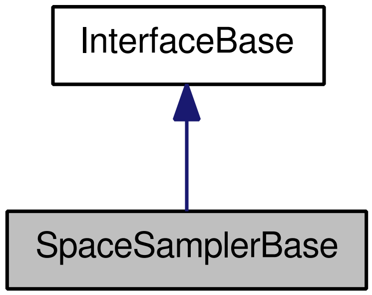
11.60 SpaceSamplerBase Class Reference

[interface] Contains space samplers commonly used in planners. **If not specified, method is not multi-thread safe.** See SpaceSampler Concepts.

Inheritance diagram for SpaceSamplerBase:



Collaboration diagram for SpaceSamplerBase:



Public Member Functions

- virtual void SetSeed (uint32_t seed) OPENRAVE_DUMMY_IMPLEMENTATION

  sets a new seed. For sequence samplers, the seed describes the n^th sample to begin at.

- virtual void SetSpaceDOF (int dof) OPENRAVE_DUMMY_IMPLEMENTATION

  Sets the degrees of freedom of the space (note this is different from the parameterization dimension).

- virtual int GetDOF () const =0

  returns the degrees of freedom of the sampling space

- virtual int GetNumberOfValues () const =0

  Dimension of the return samples.

- virtual void GetLimits (std::vector< dReal > &vLowerLimit, std::vector< dReal > &vUpperLimit) const OPENRAVE_DUMMY_IMPLEMENTATION

  returns the minimum and maximum values returned for each dimension (size is GetNumberOfValues())

- virtual void GetLimits (std::vector< uint32_t > &vLowerLimit, std::vector< uint32_t > &vUpperLimit) const OPENRAVE_DUMMY_IMPLEMENTATION

  returns the minimum and maximum values returned for each dimension (size is GetNumberOfValues())

- virtual void SampleSequence (std::vector< dReal > &samples, size_t num=1, IntervalType interval=IT_Closed) OPENRAVE_DUMMY_IMPLEMENTATION

  sequentially sampling returning the next 'num' samples

- virtual void SampleSequence (std::vector< uint32_t > &sample, size_t num=1) OPENRAVE_DUMMY_IMPLEMENTATION

  sequentially sampling returning the next 'num' samples

- virtual void SampleComplete (std::vector< dReal > &samples, size_t num, IntervalType interval=IT_Closed) OPENRAVE_DUMMY_IMPLEMENTATION

  returns N samples that best approximate the entire sampling space.

- virtual void SampleComplete (std::vector< uint32_t > &samples, size_t num) OPENRAVE_DUMMY_IMPLEMENTATION

  returns N samples that best approximate the entire sampling space.

Static Public Member Functions

- static InterfaceType GetInterfaceTypeStatic ()

  return the static interface type this class points to (used for safe casting)

11.60.1 Detailed Description

[interface] Contains space samplers commonly used in planners. **If not specified, method is not multi-thread safe.** See SpaceSampler Concepts.

11.60.2 Member Function Documentation

11.60.2.1 virtual void GetLimits (std::vector< uint32_t > & *vLowerLimit*, std::vector< uint32_t > & *vUpperLimit*) const `[virtual]`

returns the minimum and maximum values returned for each dimension (size is GetNumberOfValues())

By default the limits should be [0,2^32-1]

11.60.2.2 virtual void GetLimits (std::vector< dReal > & *vLowerLimit*, std::vector< dReal > & *vUpperLimit*) const [virtual]

returns the minimum and maximum values returned for each dimension (size is GetNumberOfValues())

By default the limits should be in [0,1]^N.

11.60.2.3 virtual int GetNumberOfValues () const [pure virtual]

Dimension of the return samples.

Number of values used to represent the parameterization of the space (>= dof). For example, let a quaternion describe a 3D rotation. The DOF of the space is 3, while the dimension of the returned samples is 4.

11.60.2.4 virtual void SampleComplete (std::vector< uint32_t > & *samples*, size_t *num*) [virtual]

returns N samples that best approximate the entire sampling space.

The sampler can fail by returning an array of size 0.

11.60.2.5 virtual void SampleComplete (std::vector< dReal > & *samples*, size_t *num*, IntervalType *interval* = IT_Closed) [virtual]

returns N samples that best approximate the entire sampling space.

The sampler can fail by returning an array of size 0.

11.60.2.6 virtual void SampleSequence (std::vector< uint32_t > & *sample*, size_t *num* = 1) [virtual]

sequentially sampling returning the next 'num' samples

The sampler can fail by returning an array of size 0.

Parameters

sample the values of the samples. This is a num*GetNumberOfValues() array.

num number of samples to return

11.60.2.7 virtual void SampleSequence (std::vector< dReal > & *samples*, size_t *num* = 1, IntervalType *interval* = IT_Closed) [virtual]

sequentially sampling returning the next 'num' samples

The sampler can fail by returning an array of size 0.

Parameters

sample the values of the samples. This is a num*GetNumberOfValues() array.

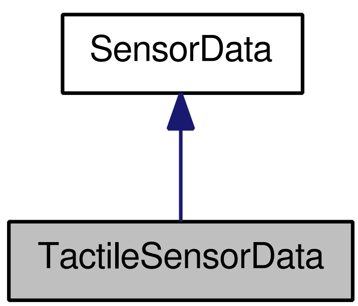
num number of samples to return

interval the sampling intervel for each of the dimensions.

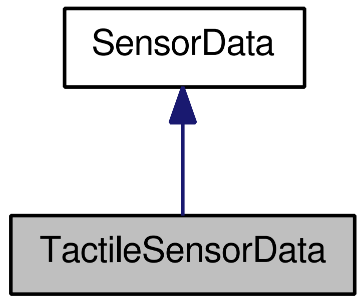
11.61 TactileSensorData Class Reference

tactle data

Inheritance diagram for TactileSensorData:



Collaboration diagram for TactileSensorData:



Public Attributes

- boost::array< dReal, 9 > force_covariance
  xyz force of each individual element

11.61.1 Detailed Description

tactle data

11.61.2 Member Data Documentation

11.61.2.1 boost::array<dReal,9> force_covariance

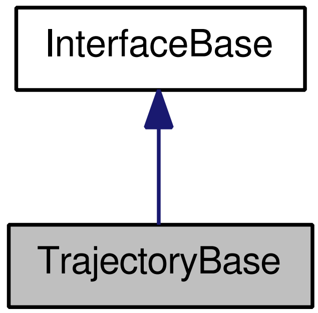
xyz force of each individual element

row major 3x3 matrix of the uncertainty on the xyz force measurements

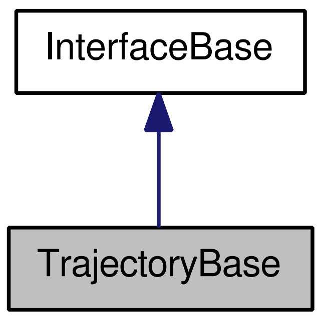
11.62 TrajectoryBase Class Reference

[interface] Encapsulate a time-parameterized trajectories of robot configurations. **If not specified, method is not multi-thread safe.** `Trajectory Concepts`

Inheritance diagram for TrajectoryBase:



Collaboration diagram for TrajectoryBase:



Classes

- class Point

Public Types

- typedef Point TPOINT RAVE_DEPRECATED

Public Member Functions

- virtual void Insert (size_t index, const std::vector< dReal > &data, bool bOverwrite=false)=0

  Sets/inserts new waypoints in the same configuration specification as the trajectory.

- virtual void Insert (size_t index, const std::vector< dReal > &data, const ConfigurationSpecification &spec, bool bOverwrite=false)=0

    *Sets/inserts new waypoints in a **user-given** configuration specification.*

- virtual void Remove (size_t startindex, size_t endindex)=0

    removes a number of waypoints starting at the specified index

- virtual void Sample (std::vector< dReal > &data, dReal time) const =0

    samples a data point on the trajectory at a particular time

- virtual void Sample (std::vector< dReal > &data, dReal time, const ConfigurationSpecification &spec) const

    samples a data point on the trajectory at a particular time and returns data for the group specified.

- virtual size_t GetNumWaypoints () const =0

    return the number of waypoints

- virtual void GetWaypoints (size_t startindex, size_t endindex, std::vector< dReal > &data) const =0

    return a set of waypoints in the range [startindex,endindex)

- virtual void GetWaypoints (size_t startindex, size_t endindex, std::vector< dReal > &data, const ConfigurationSpecification &spec) const

    return a set of waypoints in the range [startindex,endindex) in a different configuration specification.

- void GetWaypoint (int index, std::vector< dReal > &data) const

    returns one waypoint

- void GetWaypoint (int index, std::vector< dReal > &data, const ConfigurationSpecification &spec) const

    returns one waypoint

- virtual dReal GetDuration () const =0

    return the duration of the trajectory in seconds

- virtual void serialize (std::ostream &O, int options=0) const

    output the trajectory in XML format

- virtual void deserialize (std::istream &I)

    initialize the trajectory

- virtual bool SampleTrajectory (dReal time, Point &tp) const RAVE_DEPRECATED
- virtual const std::vector< Point > & GetPoints () const RAVE_DEPRECATED
- int GetDOF () const RAVE_DEPRECATED
- virtual dReal GetTotalDuration () const RAVE_DEPRECATED
- virtual bool Write (std::ostream &O, int options) const RAVE_DEPRECATED
- virtual bool Read (std::istream &I, RobotBaseConstPtr) RAVE_DEPRECATED
- virtual int GetInterpMethod () const RAVE_DEPRECATED
- virtual bool CalcTrajTiming (RobotBasePtr probot, int interp, bool autocalc, bool activedof, dReal fmaxvelmult=1) RAVE_DEPRECATED
- virtual void Clear () RAVE_DEPRECATED
- virtual void AddPoint (const Point &p) RAVE_DEPRECATED
- virtual void Reset (int dof) RAVE_DEPRECATED

Static Public Member Functions

- static InterfaceType GetInterfaceTypeStatic ()

    return the static interface type this class points to (used for safe casting)

Static Public Attributes

- static const int TO_OneLine RAVE_DEPRECATED = 1

11.62.1 Detailed Description

[interface] Encapsulate a time-parameterized trajectories of robot configurations. **If not specified, method is not multi-thread safe.** `Trajectory Concepts`

11.62.2 Member Typedef Documentation

11.62.2.1 typedef Point TPOINT RAVE_DEPRECATED

11.62.3 Member Function Documentation

11.62.3.1 virtual void AddPoint (const Point & *p*) `[virtual]`

11.62.3.2 virtual bool CalcTrajTiming (RobotBasePtr *probot*, int *interp*, bool *autocalc*, bool *activedof*, dReal *fmaxvelmult* = 1) [virtual]

11.62.3.3 virtual void Clear () [virtual]

11.62.3.4 int GetDOF () const

11.62.3.5 virtual int GetInterpMethod () const [virtual]

11.62.3.6 virtual const std::vector<Point>& GetPoints () const [virtual]

11.62.3.7 virtual dReal GetTotalDuration () const [virtual]

11.62.3.8 void GetWaypoint (int *index*, std::vector< dReal > & *data*, const ConfigurationSpecification & *spec*) const

returns one waypoint

Parameters

index[in] index of the waypoint. If < 0, then counting starts from the last waypoint. For example GetWaypoints(-1,data) returns the last waypoint.

data[out] the data of the waypoint

11.62.3.9 void GetWaypoint (int *index*, std::vector< dReal > & *data*) const

returns one waypoint

Parameters

- ***index[in]*** index of the waypoint. If < 0, then counting starts from the last waypoint. For example GetWaypoints(-1,data) returns the last waypoint.
- ***data[out]*** the data of the waypoint

11.62.3.10 virtual void GetWaypoints (size_t *startindex*, size_t *endindex*, std::vector< dReal > & *data*, const ConfigurationSpecification & *spec*) const [virtual]

return a set of waypoints in the range [startindex,endindex) in a different configuration specification.

The default implementation is very slow, so trajectory developers should really override it.

Parameters

- ***startindex[in]*** the start index of the waypoint (included)
- ***endindex[in]*** the end index of the waypoint (not included)
- ***spec[in]*** the specification to return the data in
- ***data[out]*** the data of the waypoint

11.62.3.11 virtual void GetWaypoints (size_t *startindex*, size_t *endindex*, std::vector< dReal > & *data*) const [pure virtual]

return a set of waypoints in the range [startindex,endindex)

Parameters

- ***startindex[in]*** the start index of the waypoint (included)
- ***endindex[in]*** the end index of the waypoint (not included)
- ***data[out]*** the data of the waypoint

11.62.3.12 virtual void Insert (size_t *index*, const std::vector< dReal > & *data*, const ConfigurationSpecification & *spec*, bool *bOverwrite* = `false`) `[pure virtual]`

Sets/inserts new waypoints in a **user-given** configuration specification.

Parameters

index The index where to start modifying the trajectory.

data The data to insert, can represent multiple consecutive waypoints. data.size()/GetConfigurationSpecification().GetDOF() waypoints are added.

spec the specification in which the input data come in. Depending on what data is offered, some values of this trajectory's specification might not be initialized.

bOverwrite If true, will overwrite the waypoints starting at index, and will insert new waypoints only if end of trajectory is reached. If false, will insert the points before index: a 0 index inserts the new data in the beginning, a GetNumWaypoints() index inserts the new data at the end.

11.62.3.13 virtual void Insert (size_t *index*, const std::vector< dReal > & *data*, bool *bOverwrite* = `false`) `[pure virtual]`

Sets/inserts new waypoints in the same configuration specification as the trajectory.

Parameters

index The index where to start modifying the trajectory.

data The data to insert, can represent multiple consecutive waypoints. data.size()/GetConfigurationSpecification().GetDOF() waypoints are added.

bOverwrite If true, will overwrite the waypoints starting at index, and will insert new waypoints only if end of trajectory is reached. If false, will insert the points before index: a 0 index inserts the new data in the beginning, a GetNumWaypoints() index inserts the new data at the end.

11.62.3.14 virtual bool Read (std::istream & *I*, RobotBaseConstPtr) `[virtual]`

11.62.3.15 virtual void Reset (int *dof*) `[virtual]`

11.62.3.16 virtual void Sample (std::vector< dReal > & *data*, dReal *time*, const ConfigurationSpecification & *spec*) const `[virtual]`

samples a data point on the trajectory at a particular time and returns data for the group specified.

The default implementation is slow, so interface developers should override it.

Parameters

data[out] the sampled point

time[in] the time to sample

spec[in] the specification format to return the data in

11.62.3.17 virtual void Sample (std::vector< dReal > & *data*, dReal *time*) const `[pure virtual]`

samples a data point on the trajectory at a particular time

Parameters

data[out] the sampled point

time[in] the time to sample

11.62.3.18 virtual bool SampleTrajectory (dReal *time*, Point & *tp*) const `[virtual]`

11.62.3.19 virtual bool Write (std::ostream & *O*, int *options*) const `[virtual]`

11.62.4 Member Data Documentation

11.62.4.1 const int TO_OneLine RAVE_DEPRECATED = 1 `[static]`

11.63 triangle< T > Class Template Reference

A triangle defined by 3 points.

Public Member Functions

- RaveVector< T > normal ()

  assumes CCW ordering of vertices

Public Attributes

- RaveVector< T > v3

  the vertices of the triangle

11.63.1 Detailed Description

template<typename T> class OpenRAVE::geometry::triangle< T >

A triangle defined by 3 points.

11.64 TRIMESH Class Reference

User data for trimesh geometries. Vertices are defined in counter-clockwise order for outward pointing faces.

Public Member Functions

- void Append (const TRIMESH &mesh)

  append another TRIMESH to this tri mesh

11.64.1 Detailed Description

User data for trimesh geometries. Vertices are defined in counter-clockwise order for outward pointing faces.

11.65 UserData Class Reference

base class for all user data

Inherited by FunctionUserData.

11.65.1 Detailed Description

base class for all user data

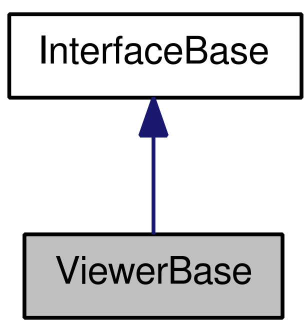
Examples:

orpythonbinding.cpp.

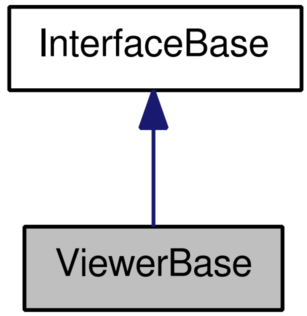
11.66 ViewerBase Class Reference

[interface] Base class for the graphics and gui engine that renders the environment and provides visual sensor information. **If not specified, method is not multi-thread safe.** See Viewer Concepts.

Inheritance diagram for ViewerBase:



Collaboration diagram for ViewerBase:



Public Types

- typedef boost::function< bool(KinBody::LinkPtr plink, RaveVector< float >, RaveVector< float >)> ItemSelectionCallbackFn

  callback function for item selection

- typedef boost::function< void(const uint8_t ∗, int, int, int) ViewerImageCallbackFn)

  callback function for item selection callback(imagememory,width,height,pixeldepth)

- typedef ItemSelectionCallbackFn ViewerCallbackFn RAVE_DEPRECATED

Public Member Functions

- virtual int main (bool bShow=true)=0

  goes into the main loop

- virtual void quitmainloop ()=0

  destroys the main loop

- virtual boost::shared_ptr< void > RegisterItemSelectionCallback (const ItemSelectionCallbackFn &fncallback) OPENRAVE_DUMMY_IMPLEMENTATION

  registers a function with the viewer that gets called everytime mouse button is clicked

- virtual boost::shared_ptr< void > RegisterViewerImageCallback (const ViewerImageCallbackFn &fncallback) OPENRAVE_DUMMY_IMPLEMENTATION

  registers a function with the viewer that gets called for every new image rendered.

- virtual void SetEnvironmentSync (bool bUpdate) OPENRAVE_DUMMY_-IMPLEMENTATION

  controls whether the viewer synchronizes with the newest environment automatically

- virtual void EnvironmentSync () OPENRAVE_DUMMY_-IMPLEMENTATION

  forces synchronization with the environment, returns when the environment is fully synchronized.

- virtual void ViewerSetSize (int w, int h) RAVE_DEPRECATED
- virtual void ViewerMove (int x, int y) RAVE_DEPRECATED
- virtual void ViewerSetTitle (const std::string &ptitle) RAVE_DEPRECATED
- virtual void UpdateCameraTransform () RAVE_DEPRECATED OPENRAVE_-DUMMY_IMPLEMENTATION
- virtual boost::shared_ptr< void > RegisterCallback (int properties, const Item-SelectionCallbackFn &fncallback) RAVE_DEPRECATED
- virtual bool LoadModel (const std::string &pfilename) RAVE_DEPRECATED OPENRAVE_DUMMY_IMPLEMENTATION

  - virtual void SetCamera (const RaveTransform< float > &trans, float focalD-istance=0) OPENRAVE_DUMMY_IMPLEMENTATION

    Set the camera transformation.

  - virtual RaveTransform< float > GetCameraTransform () const OPENRAVE_DUMMY_IMPLEMENTATION

    Return the current camera transform that the viewer is rendering the environment at.

  - virtual geometry::RaveCameraIntrinsics< float > GetCameraIntrinsics () const OPENRAVE_DUMMY_IMPLEMENTATION

    Return the closest camera intrinsics that the viewer is rendering the environment at.

  - virtual bool GetCameraImage (std::vector< uint8_t > &memory, int width, int height, const RaveTransform< float > &t, const SensorBase::CameraIntrinsics &intrinsics) OPENRAVE_DUMMY_-IMPLEMENTATION

    Renders a 24bit RGB image of dimensions width and height from the current scene.

Static Public Member Functions

- static InterfaceType GetInterfaceTypeStatic ()

  return the static interface type this class points to (used for safe casting)

11.66.1 Detailed Description

[interface] Base class for the graphics and gui engine that renders the environment and provides visual sensor information. **If not specified, method is not multi-thread safe.** See Viewer Concepts.

11.66.2 Member Typedef Documentation

11.66.2.1 typedef boost::function<bool (KinBody::LinkPtr plink,RaveVector<float>,RaveVector<float>)> ItemSelectionCallbackFn

callback function for item selection

If the function returns true, then the object will be selected. Otherwise, the object remains unselected. callback(target link,offset,direction)

11.66.2.2 typedef ItemSelectionCallbackFn ViewerCallbackFn RAVE_DEPRECATED

11.66.2.3 typedef boost::function<void (const uint8_t*,int,int,int) ViewerImageCallbackFn)

callback function for item selection callback(imagememory,width,height,pixeldepth)

Parameters

imagememory width x height x pixeldepth RGB image

11.66.3 Member Function Documentation

11.66.3.1 virtual void EnvironmentSync () `[virtual]`

forces synchronization with the environment, returns when the environment is fully synchronized.

Note that this method might not work if environment is locked in current thread

11.66.3.2 virtual bool GetCameraImage (std::vector< uint8_t > & *memory*, int *width*, int *height*, const RaveTransform< float > & *t*, const SensorBase::CameraIntrinsics & *intrinsics*) `[virtual]`

Renders a 24bit RGB image of dimensions width and height from the current scene.

The camera is meant to show the underlying OpenRAVE world as a robot would see it, so all graphs rendered with the plotX and drawX functions are hidden by default. Some viewers support the SetFiguresInCamera command to allow graphs to be also displayed.

Parameters

- ***memory*** the memory where the image will be stored at, has to store 3∗width∗height
- ***width*** width of the image, if 0 the width of the viewer is used
- ***height*** height of the image, if 0 the width of the viewer is used
- ***t*** the rotation and translation of the camera. Note that +z is treated as the camera direction axis! So all points in front of the camera have a positive dot product with its +z direction.
- ***intrinsics*** the intrinsic parameters of the camera defining FOV, distortion, principal point, and focal length. The focal length is used to define the near plane for culling.

11.66.3.3 virtual bool LoadModel (const std::string & *pfilename*) `[virtual]`

11.66.3.4 virtual int main (bool *bShow* = `true`) `[pure virtual]`

goes into the main loop

Parameters

- ***bShow*** if true will show the window

11.66.3.5 virtual boost::shared_ptr<void> RegisterCallback (int *properties*, const ItemSelectionCallbackFn & *fncallback*) `[virtual]`

11.66.3.6 virtual boost::shared_ptr<void> RegisterItemSelectionCallback (const ItemSelectionCallbackFn & *fncallback*) `[virtual]`

registers a function with the viewer that gets called everytime mouse button is clicked

Returns

a handle to the callback. If this handle is deleted, the callback will be unregistered.

11.66.3.7 virtual boost::shared_ptr<void> RegisterViewerImageCallback (const ViewerImageCallbackFn & *fncallback*) `[virtual]`

registers a function with the viewer that gets called for every new image rendered.

Returns

a handle to the callback. If this handle is deleted, the callback will be unregistered.

11.66.3.8 virtual void SetCamera (const RaveTransform< float > & *trans*, float *focalDistance* = 0) `[virtual]`

Set the camera transformation.

Parameters

trans new camera transformation in the world coordinate system

focalDistance The new focal distance of the camera (higher values is higher zoom). If 0, then the previous focal distance is preserved.

11.66.3.9 virtual void UpdateCameraTransform () `[virtual]`

11.66.3.10 virtual void ViewerMove (int *x*, int *y*) `[virtual]`

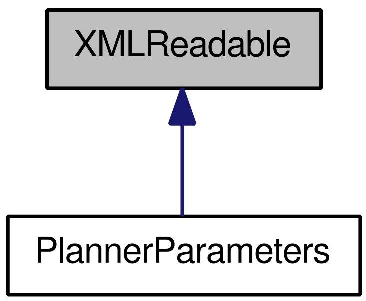
11.66.3.11 virtual void ViewerSetSize (int *w*, int *h*) `[virtual]`

11.66.3.12 virtual void ViewerSetTitle (const std::string & *ptitle*) `[virtual]`

11.67 XMLReadable Class Reference

base class for readable interfaces

Inheritance diagram for XMLReadable:



11.67.1 Detailed Description

base class for readable interfaces

Examples:

customreader.cpp.

12 Example Documentation

12.1 customreader.cpp

Author

Rosen Diankov

Creates a simple OpenRAVE::ControllerBase and shows how to add a custom XML reader to it.

Full Example Code:

```
#include <openrave/openrave.h>
#include <openrave/plugin.h>

using namespace std;
using namespace OpenRAVE;

class CustomController : public ControllerBase
{
public:
    class XMLData : public XMLReadable
    {
public:
        XMLData() : XMLReadable("piddata") {
        }
        vector<dReal> pgains,igains;
    };

    class PIDXMLReader : public BaseXMLReader
    {
public:
        PIDXMLReader(boost::shared_ptr<XMLData> piddata, const AttributesList& at
      ts) {
            _piddata = piddata;
            if( !_piddata )
                _piddata.reset(new XMLData());
            RAVELOG_INFO("the attributes piddata is created with are:\n");
            for(AttributesList::const_iterator itatt = atts.begin(); itatt != att
      s.end(); ++itatt)
                RAVELOG_INFO("%s=%s\n",itatt->first.c_str(),itatt->second.c_str()
      );
        }

        virtual XMLReadablePtr GetReadable() {
            return _piddata;
        }

        virtual ProcessElement startElement(const std::string& name, const Attrib
      utesList& atts) {
            _ss.str("");
            return (name == "pgains" || name=="igains") ? PE_Support : PE_Pass;
        }
```

```
        virtual bool endElement(const std::string& name)
        {
            if( name == "piddata" )
                return true;
            else if( name == "pgains" )
                // read all the float values into a vector
                _piddata->pgains = vector<dReal>((istream_iterator<dReal>(_ss)),
      istream_iterator<dReal>());
            else if( name == "igains" )
                // read all the float values into a vector
                _piddata->igains = vector<dReal>((istream_iterator<dReal>(_ss)),
      istream_iterator<dReal>());
            else
                RAVELOG_ERROR("unknown field %s\n", name.c_str());

            return false;
        }

        virtual void characters(const std::string& ch)
        {
            _ss.clear();
            _ss << ch;
        }

protected:
        boost::shared_ptr<XMLData> _piddata;
        stringstream _ss;
    };

    static BaseXMLReaderPtr CreateXMLReader(InterfaceBasePtr ptr, const Attribute
      sList& atts)
    {
        // ptr is the robot interface that this reader is being created for
        return BaseXMLReaderPtr(new PIDXMLReader(boost::shared_ptr<XMLData>(),att
      s));
    }

    CustomController(EnvironmentBasePtr penv) : ControllerBase(penv)
    {
    }
    virtual ~CustomController() {
    }

    virtual bool Init(RobotBasePtr robot, const std::vector<int>& dofindices, int
       nControlTransformation)
    {
        _probot = robot;
        _dofindices = dofindices;
        _nControlTransformation = nControlTransformation;

        // read the gains from the XML
        boost::shared_ptr<XMLData> piddata = boost::dynamic_pointer_cast<XMLData>
      (GetReadableInterface("piddata"));
        if( !!piddata ) {
            stringstream ss;
            ss << "piddata from custom XML reader is" << endl << "pgains: ";
```

```
            for(vector<dReal>::iterator it = piddata->pgains.begin(); it != pidda
      ta->pgains.end(); ++it)
                ss << *it << " ";
            ss << endl << "igains: ";
            for(vector<dReal>::iterator it = piddata->igains.begin(); it != pidda
      ta->igains.end(); ++it)
                ss << *it << " ";
            ss << endl;
            RAVELOG_INFOA(ss.str());
        }
        else
            RAVELOG_WARN("failed to find piddata\n");
        return true;
    }

    virtual const std::vector<int>& GetControlDOFIndices() const {
        return _dofindices;
    }
    virtual int IsControlTransformation() const {
        return _nControlTransformation;
    }

    virtual void Reset(int options) {
    }
    virtual bool SetDesired(const std::vector<dReal>& values, TransformConstPtr t
      rans) {
        return false;
    }
    virtual bool SetPath(TrajectoryBaseConstPtr ptraj) {
        return false;
    }
    virtual void SimulationStep(dReal fTimeElapsed) {
    }
    virtual bool IsDone() {
        return false;
    }
    virtual dReal GetTime() const {
        return 0;
    }
    virtual RobotBasePtr GetRobot() const {
        return _probot;
    }

protected:
    RobotBasePtr _probot;
    std::vector<int> _dofindices;
    int _nControlTransformation;
};

static boost::shared_ptr<void> s_RegisteredReader;

InterfaceBasePtr CreateInterfaceValidated(InterfaceType type, const std::string&
      interfacename, std::istream& sinput, EnvironmentBasePtr penv)
{
    if( !s_RegisteredReader ) {
        s_RegisteredReader = RaveRegisterXMLReader(PT_Controller,"piddata",Custom
      Controller::CreateXMLReader);
```

```
    }
    switch(type) {
    case PT_Controller:
        if( interfacename == "customcontroller")
            return InterfaceBasePtr(new CustomController(penv));
        break;
    default:
        break;
    }
    return InterfaceBasePtr();
}

void GetPluginAttributesValidated(PLUGININFO& info)
{
    info.interfacenames[PT_Controller].push_back("CustomController");
}

OPENRAVE_PLUGIN_API void DestroyPlugin()
{
    s_RegisteredReader.reset(); // unregister the reader
}
```

12.2 FindOpenRAVE.cmake

Allows OpenRAVE installation to be found when using the CMake build system.

```
# - Find Open Robotics Automation Virtual Enviornment (OpenRAVE) Installation
# http://www.openrave.org
#
# OpenRAVE provides an environment for testing, developing, and deploying motion
      planning algorithms
# in real-world robotics applications. The main focus is on simulation and analys
      is of kinematic and
# geometric information related to motion planning. OpenRAVE's stand-alone nature
       allows is to be easily
# integrated into existing robotics systems. An important target application is i
      ndustrial robotics automation.

#=================================================================================
      ==
# Copyright (C) 2009-2011 Rosen Diankov
#
# Distributed under the OSI-approved BSD License (the "License");
# see accompanying file Copyright.txt for details.
#
# This software is distributed WITHOUT ANY WARRANTY; without even the
# implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
# See the License for more information.
#==============================================================================
# (To distributed this file outside of CMake, substitute the full
#  License text for the above reference.)
#=================================================================================
      ==

set(_OpenRAVE_PATHS)
```

```
if(NOT OpenRAVE_DIR)
  if( WIN32 )
    # search in the registry
    set(_OpenRAVE_CONFIG_NAME "openrave-config.exe")
    get_filename_component(OpenRAVE_VERSION_STRING "[HKEY_LOCAL_MACHINE\\SOFTWARE
      \\OpenRAVE;]" NAME)
    message(STATUS "OpenRAVE ${OpenRAVE_VERSION_STRING} found in registry")
    if( OpenRAVE_VERSION_STRING )
      get_filename_component(_OpenRAVE_PATH "[HKEY_LOCAL_MACHINE\\SOFTWARE\\OpenR
      AVE\\${OpenRAVE_VERSION_STRING};InstallRoot]" ABSOLUTE)
      set(_OpenRAVE_PATHS ${_OpenRAVE_PATHS} ${_OpenRAVE_PATH})
    endif( OpenRAVE_VERSION_STRING )
  else(WIN32)
    set(_OpenRAVE_CONFIG_NAME "openrave-config")
  endif(WIN32)
  # search for the config path
  find_program(_OpenRAVE_CONFIG_EXECUTABLE NAMES ${_OpenRAVE_CONFIG_NAME} DOC "op
      enrave executable")
  if( _OpenRAVE_CONFIG_EXECUTABLE )
    get_filename_component(_OpenRAVE_PATH "${_OpenRAVE_CONFIG_EXECUTABLE}" PATH)
      # bin
    get_filename_component(_OpenRAVE_PATH "${_OpenRAVE_PATH}" PATH)
    set(_OpenRAVE_PATHS ${_OpenRAVE_PATHS} ${_OpenRAVE_PATH})
  endif( _OpenRAVE_CONFIG_EXECUTABLE )
endif(NOT OpenRAVE_DIR)

find_package(OpenRAVE NO_MODULE PATHS ${_OpenRAVE_PATHS})
```

12.3 ikfastloader.cpp

Author

Rosen Diankov

Usage:

```
ikloader [robot filename] [iktype]
```

Example:

```
ikloader robots/barrettwam.robot.xml Transform6D
```

Show how to load an ikfast solver from C++ by specifying the robot and iktype.

Full Example Code:

```
#include <openrave-core.h>
#include <vector>
#include <cstring>
#include <sstream>
```

```
#include <stdio.h>

#include <boost/iostreams/device/file_descriptor.hpp>
#include <boost/iostreams/stream.hpp>
#include <boost/format.hpp>

using namespace OpenRAVE;
using namespace std;

int main(int argc, char ** argv)
{
    if( argc < 3 ) {
        RAVELOG_INFO("ikloader robot iktype\n");
        return 1;
    }

    string robotname = argv[1];
    string iktype = argv[2];
    RaveInitialize(true); // start openrave core

    EnvironmentBasePtr penv = RaveCreateEnvironment(); // create the main environ
      ment

    {
        // lock the environment to prevent changes
        EnvironmentMutex::scoped_lock lock(penv->GetMutex());
        // load the scene
        RobotBasePtr probot = penv->ReadRobotXMLFile(robotname);
        if( !probot ) {
            penv->Destroy();
            return 2;
        }
        penv->AddRobot(probot);

        ModuleBasePtr pikfast = RaveCreateModule(penv,"ikfast");
        penv->AddModule(pikfast,"");
        stringstream ssin,ssout;
        ssin << "LoadIKFastSolver " << probot->GetName() << " " << iktype;
        // if necessary, add free inc for degrees of freedom
        //ssin << " " << 0.04f;
        // set the active manipulator
        probot->SetActiveManipulator(probot->GetManipulators().at(0)->GetName());

        if( !pikfast->SendCommand(ssout,ssin) ) {
            RAVELOG_ERROR("failed to load iksolver\n");
            penv->Destroy();
            return 1;
        }

        RAVELOG_INFO("testing random ik\n");
        vector<dReal> vsolution;
        if( !probot->GetActiveManipulator()->FindIKSolution(IkParameterization(pr
      obot->GetActiveManipulator()->GetEndEffectorTransform()),vsolution,true) ) {
            RAVELOG_INFO("failed to get solution\n");
        }
        else {
            stringstream ss; ss << "solution is: ";
```

```
            for(size_t i = 0; i < vsolution.size(); ++i) {
                ss << vsolution[i] << " ";
            }
            ss << endl;
            RAVELOG_INFO(ss.str());
        }
    }

    penv->Destroy(); // destroy
    return 0;
}
```

12.4 opencvsaving.cpp

Author

Rosen Diankov

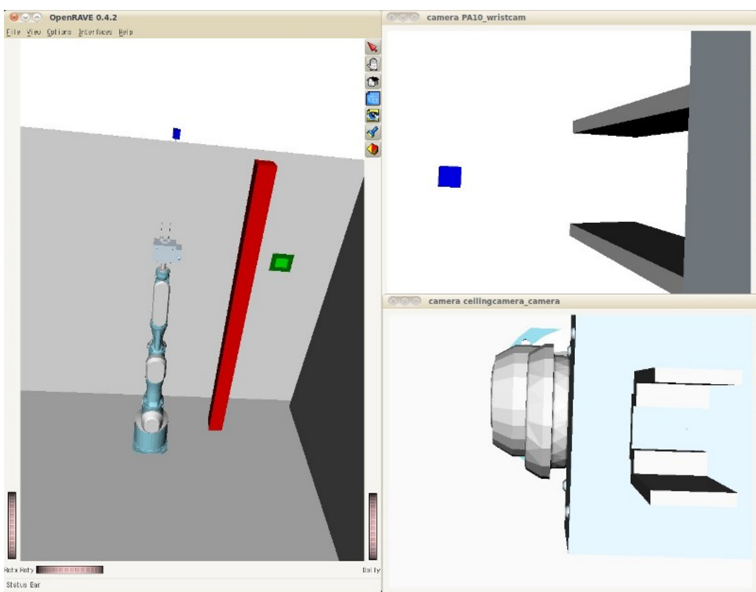


Figure 2: OpenRAVE Environment with two cameras.

This example shows how to enable all cameras loaded in an environment and convert their image data to the OpenCV IplImage structure. Then cvSaveImage is called for each image.

```
#include <openrave-core.h>
#include <vector>
```

```
#include <boost/thread/thread.hpp>
#include <boost/bind.hpp>
#include <opencv/cv.h>
#include <opencv/highgui.h>

#ifdef _WIN32
#define WIN32_LEAN_AND_MEAN
#include <winsock2.h>
#define usleep(micro) Sleep(micro/1000)
#endif

using namespace OpenRAVE;
using namespace std;

void SetViewer(EnvironmentBasePtr penv, const string& viewername)
{
    ViewerBasePtr viewer = RaveCreateViewer(penv,viewername);
    BOOST_ASSERT(!!viewer);

    // attach it to the environment:
    penv->AddViewer(viewer);

    // finally you call the viewer's infinite loop (this is why you need a separa
      te thread) :
    bool showgui = true;
    viewer->main(showgui);
}

class OpenRAVECamera
{
public:
    OpenRAVECamera(SensorBasePtr psensor)
    {
        pcamera=psensor;
        pdata = boost::static_pointer_cast<SensorBase::CameraSensorData>(pcamera-
      >CreateSensorData(SensorBase::ST_Camera));
        geom = *boost::static_pointer_cast<SensorBase::CameraGeomData>(pcamera->G
      etSensorGeometry(SensorBase::ST_Camera));
        img = cvCreateImage(cvSize(geom.width,geom.height),IPL_DEPTH_8U,3);
    }
    virtual ~OpenRAVECamera() {
        cvReleaseImage(&img);
    }

    SensorBasePtr pcamera;
    SensorBase::CameraGeomData geom;
    boost::shared_ptr<SensorBase::CameraSensorData> pdata;
    IplImage* img;
};

int main(int argc, char ** argv)
{
    RaveInitialize(true); // start openrave core
    EnvironmentBasePtr penv = RaveCreateEnvironment(); // create the main environ
      ment
    boost::thread thviewer(boost::bind(SetViewer,penv,"qtcoin"));
    penv->Load("data/pa10calib_envcamera.env.xml");
```

```
    std::vector<RobotBasePtr> vrobots;
    penv->GetRobots(vrobots);
    if( vrobots.size() > 0 ) {
        RAVELOG_INFO("moving the robot a little\n");
        Transform t = vrobots.at(0)->GetTransform();
        t.trans.x += 0.6;
        vrobots.at(0)->SetTransform(t);
    }

    // extract all the cameras
    std::vector<SensorBasePtr> allsensors;
    penv->GetSensors(allsensors);
    std::vector< boost::shared_ptr<OpenRAVECamera> > vcameras;
    for( std::vector<SensorBasePtr>::iterator itsensor = allsensors.begin(); itse
      nsor != allsensors.end(); ++itsensor) {
        if( (*itsensor)->Supports(SensorBase::ST_Camera) ) {
            (*itsensor)->Configure(SensorBase::CC_PowerOn);
            (*itsensor)->Configure(SensorBase::CC_RenderDataOn);
            vcameras.push_back(boost::shared_ptr<OpenRAVECamera>(new OpenRAVECame
      ra(*itsensor)));
        }
    }
    while(1) {
        // read the camera data and save the image
        for(size_t icamera = 0; icamera < vcameras.size(); ++icamera) {
            vcameras[icamera]->pcamera->GetSensorData(vcameras[icamera]->pdata);
            if( vcameras[icamera]->pdata->vimagedata.size() > 0 ) {
                char* imageData = vcameras[icamera]->img->imageData;
                uint8_t* src = &vcameras[icamera]->pdata->vimagedata.at(0);
                for(int i=0; i < vcameras[icamera]->geom.height; i++, imageData +
      = vcameras[icamera]->img->widthStep, src += 3*vcameras[icamera]->geom.width) {
                    for(int j=0; j<vcameras[icamera]->geom.width; j++) {
                        imageData[3*j] = src[3*j];
                        imageData[3*j+1] = src[3*j+1];
                        imageData[3*j+2] = src[3*j+2];
                    }
                }
                string filename = str(boost::format("camera%d.jpg")%icamera);
                RAVELOG_INFO(str(boost::format("saving image %s")%filename));
                cvSaveImage(filename.c_str(),vcameras[icamera]->img);
            }
        }
        usleep(200000);
    }

    return 0;
}
```

12.5 orcollision.cpp

Author

Rosen Diankov

Load a robot into the openrave environment, set it at [joint values] and check for self collisions. Returns number of contact points.

Usage:

```
orcollision [--list] [--checker checker_name] [--joints #values [values]] body_model
```

- **--list** - List all the loadable interfaces (ie, collision checkers).
- **--checker** - name Load a different collision checker instead of the default one.
- **--joints #values [values]** - Set the robot to specific joint values

Example:

```
orcollision --checker ode robots/barrettwam.robot.xml
```

Full Example Code:

```
#include <openrave-core.h>
#include <vector>
#include <cstring>
#include <sstream>

using namespace OpenRAVE;
using namespace std;


void printhelp()
{
    RAVELOG_INFO("orcollision [--list] [--checker checker_name] [--joints #values
       [values]] body_model\n");
}

void printinterfaces(EnvironmentBasePtr penv)
{
    std::map<InterfaceType, std::vector<std::string> > interfacenames;
    RaveGetLoadedInterfaces(interfacenames);
    stringstream ss;

    ss << endl << "Loadable interfaces: " << endl;
    for(std::map<InterfaceType, std::vector<std::string> >::iterator itinterface
      = interfacenames.begin(); itinterface != interfacenames.end(); ++itinterface) {
        ss << RaveGetInterfaceName(itinterface->first) << "(" << itinterface->sec
      ond.size() << "):" << endl;
        for(vector<string>::iterator it = itinterface->second.begin(); it != itin
      terface->second.end(); ++it)
            ss << " " << *it << endl;
        ss << endl;
    }
    RAVELOG_INFO(ss.str());
```

```
}

int main(int argc, char ** argv)
{
    if( argc < 2 ) {
        printhelp();
        return -1; // no robots to load
    }

    RaveInitialize(true); // start openrave core
    EnvironmentBasePtr penv = RaveCreateEnvironment(); // create the main environ
      ment
    vector<dReal> vsetvalues;

    // parse the command line options
    int i = 1;
    while(i < argc) {
        if((strcmp(argv[i], "-h") == 0)||(strcmp(argv[i], "-?") == 0)||(strcmp(ar
      gv[i], "/?") == 0)||(strcmp(argv[i], "--help") == 0)||(strcmp(argv[i], "-help") =
      = 0)) {
            printhelp();
            return 0;
        }
        else if( strcmp(argv[i], "--checker") == 0 ) {
            // create requested collision checker
            CollisionCheckerBasePtr pchecker = RaveCreateCollisionChecker(penv,ar
      gv[i+1]);
            if( !pchecker ) {
                RAVELOG_ERROR("failed to create checker %s\n", argv[i+1]);
                return -3;
            }
            penv->SetCollisionChecker(pchecker);
            i += 2;
        }
        else if( strcmp(argv[i], "--list") == 0 ) {
            printinterfaces(penv);
            return 0;
        }
        else if( strcmp(argv[i], "--joints") == 0 ) {
            vsetvalues.resize(atoi(argv[i+1]));
            for(int j = 0; j < (int)vsetvalues.size(); ++j)
                vsetvalues[j] = atoi(argv[i+j+2]);

            i += 2+vsetvalues.size();
        }
        else
            break;
    }

    if( i >= argc ) {
        RAVELOG_ERROR("not enough parameters\n");
        printhelp();
        return 1;
    }

    // load the scene
    if( !penv->Load(argv[i]) ) {
```

```
        return 2;
    }

    // lock the environment to prevent thigns from changes
    EnvironmentMutex::scoped_lock lock(penv->GetMutex());

    vector<KinBodyPtr> vbodies;
    penv->GetBodies(vbodies);
    // get the first body
    if( vbodies.size() == 0 ) {
        RAVELOG_ERROR("no bodies loaded\n");
        return -3;
    }

    KinBodyPtr pbody = vbodies.at(0);
    vector<dReal> values;
    pbody->GetDOFValues(values);

    // set new values
    for(int i = 0; i < (int)vsetvalues.size() && i < (int)values.size(); ++i) {
        values[i] = vsetvalues[i];
    }
    pbody->SetDOFValues(values,true);

    int contactpoints = 0;
    CollisionReportPtr report(new CollisionReport());
    penv->GetCollisionChecker()->SetCollisionOptions(CO_Contacts);
    if( pbody->CheckSelfCollision(report) ) {
        contactpoints = (int)report->contacts.size();
        stringstream ss;
        ss << "body in self-collision "
           << (!!report->plink1 ? report->plink1->GetName() : "") << ":"
           << (!!report->plink2 ? report->plink2->GetName() : "") << " at "
           << contactpoints << "contacts" << endl;
        for(int i = 0; i < contactpoints; ++i) {
            CollisionReport::CONTACT& c = report->contacts[i];
            ss << "contact" << i << ": pos=("
               << c.pos.x << ", " << c.pos.y << ", " << c.pos.z << "), norm=("
               << c.norm.x << ", " << c.norm.y << ", " << c.norm.z << ")" << endl
      ;
        }

        RAVELOG_INFOA(ss.str());
    }
    else {
        RAVELOG_INFO("body not in collision\n");
    }

    // get the transformations of all the links
    vector<Transform> vlinktransforms;
    pbody->GetLinkTransformations(vlinktransforms);

    penv->Destroy(); // destroy
    return contactpoints;
}
```

12.6 orconveyormovement.cpp

Author

Rosen Diankov

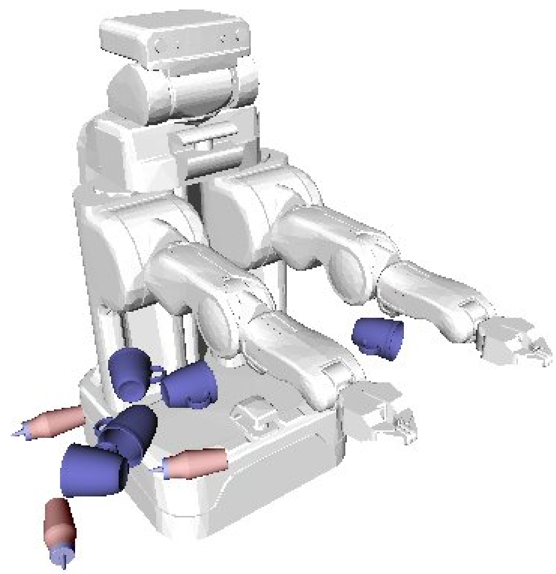


Figure 3: Parts moving on a conveyor belt.

Shows how to setup a simulation loop to move objects around a conveyor belt.

Full Example Code:

```
#include <openrave-core.h>
#include <vector>
#include <sstream>
#include <boost/thread/thread.hpp>
#include <boost/bind.hpp>

using namespace OpenRAVE;
using namespace std;

#ifdef _WIN32
#define WIN32_LEAN_AND_MEAN
#include <windows.h>
#define usleep(micro) Sleep(micro/1000)
#endif

class ConveyorBeltModule : public ModuleBase
```

```
{
    struct RegisteredBody
    {
        string filename;
        dReal appearanceprobability;     // probably of appearance in 1 second
    };

    struct InstancedBody
    {
        KinBodyPtr pbody;
        dReal timeleft;
    };
    SpaceSamplerBasePtr _psampler;
public:
    ConveyorBeltModule(EnvironmentBasePtr penv, std::istream& is) : ModuleBase(pe
      nv)
    {
        __description = "Handles conveyor belt movement";
        RegisterCommand("registerbody",boost::bind(&ConveyorBeltModule::RegisterB
      ody,this,_1,_2),"registers a body to be put into the environment");
        movevel = Vector(0,0.4,0);
        start = Vector(0.5,-1,0.6);
        _psampler = RaveCreateSpaceSampler(penv,"mt19937");
    }

    int main(const string& cmd)
    {
        return 0;
    }

    bool RegisterBody(ostream& sout, istream& sinput)
    {
        EnvironmentMutex::scoped_lock lock(GetEnv()->GetMutex());
        RegisteredBody body;
        sinput >> body.filename >> body.appearanceprobability;
        if( !sinput ) {
            return false;
        }
        _listregistered.push_back(body);
        return true;
    }

    bool SimulationStep(dReal fElapsedTime)
    {
        for(list<RegisteredBody>::iterator it = _listregistered.begin(); it != _l
      istregistered.end(); ++it) {
            // appearanceprobabiliy is in seconds, so have to transform
            dReal appearanceprobability = 1-pow(1-it->appearanceprobability,fElap
      sedTime);
            vector<dReal> vsample;
            _psampler->SampleSequence(vsample,4,IT_OpenStart);
            if( vsample.at(0) < appearanceprobability ) {
                KinBodyPtr pbody = GetEnv()->ReadKinBodyXMLFile(it->filename);
                GetEnv()->AddKinBody(pbody,true);
                InstancedBody b;
                for(int iter = 0; iter < 10; ++iter) {
                    Transform t;
```

```
                t.rot = geometry::quatFromAxisAngle<dReal>(Vector(0,0,1),vsam
ple.at(1)*2*PI);
                t.trans = start + Vector(vsample.at(2)-0.5,vsample.at(3)-0.5,
0)*0.4;
                pbody->SetTransform(t);
                if( !GetEnv()->CheckCollision(KinBodyConstPtr(pbody)) ) {
                    b.pbody = pbody;
                    break;
                }
            }

            if( !b.pbody ) {
                GetEnv()->Remove(pbody);
            }
            else {
                b.timeleft = 4.0;
                _listinstances.push_back(b);
            }
        }
    }

    list<InstancedBody>::iterator it = _listinstances.begin();
    while(it != _listinstances.end() ) {
        Transform t = it->pbody->GetTransform();
        t.trans += fElapsedTime*movevel;
        it->pbody->SetTransform(t);
        it->timeleft -= fElapsedTime;
        if( it->timeleft <= 0 ) {
            GetEnv()->Remove(it->pbody);
            it = _listinstances.erase(it);
        }
        else {
            ++it;
        }
    }
    return false;
}

static InterfaceBasePtr create(EnvironmentBasePtr penv, std::istream& is)
{
    return InterfaceBasePtr(new ConveyorBeltModule(penv,is));
}

private:
    Vector start, movevel;
    list<RegisteredBody> _listregistered;
    list<InstancedBody> _listinstances;
};

void SetViewer(EnvironmentBasePtr penv, const string& viewername)
{
    ViewerBasePtr viewer = RaveCreateViewer(penv,viewername);
    penv->AddViewer(viewer);
    viewer->main(true);
}

int main(int argc, char ** argv)
```

```
{
    // initialize openrave and register the conveyor module
    RaveInitialize(true);
    boost::shared_ptr<void> handle = RaveRegisterInterface(PT_Module,"conveyorbel
      t",OPENRAVE_MODULE_HASH,OPENRAVE_ENVIRONMENT_HASH,ConveyorBeltModule::create);
    EnvironmentBasePtr penv = RaveCreateEnvironment();

    // load the environment
    string scenefilename = "robots/pr2-beta-static.zae";
    string viewername = "qtcoin";
    boost::thread thviewer(boost::bind(SetViewer,penv,viewername)); // create the
       viewer
    penv->Load(scenefilename);

    // create the conveyor module and add a couple of bodies for simulation
    ModuleBasePtr p = RaveCreateModule(penv,"conveyorbelt");
    penv->AddModule(p,"");
    stringstream sout, sin("registerbody data/mug1.kinbody.xml 0.6");
    p->SendCommand(sout,sin);
    sin.clear();
    sin.str("registerbody data/ketchup.kinbody.xml 0.3");
    p->SendCommand(sout,sin);

    thviewer.join(); // wait for the viewer thread to exit
    penv->Destroy(); // destroy
    return 0;
}
```

12.7 orikfilter.cpp

Author

Rosen Diankov

Shows how to use set a custom inverse kinematics filter to add extra constraints.

Full Example Code:

```
#include <openrave-core.h>
#include <vector>
#include <sstream>
#include <boost/thread/thread.hpp>
#include <boost/bind.hpp>

using namespace OpenRAVE;
using namespace std;

#ifdef _WIN32
inline static uint32_t GetMilliTime()
{
    LARGE_INTEGER count, freq;
    QueryPerformanceCounter(&count);
    QueryPerformanceFrequency(&freq);
    return (uint32_t)((count.QuadPart * 1000) / freq.QuadPart);
```

```
}
#else

inline static void getWallTime(uint32_t& sec, uint32_t& nsec)
{
    struct timeval timeofday;
    gettimeofday(&timeofday,NULL);
    sec  = timeofday.tv_sec;
    nsec = timeofday.tv_usec * 1000;
}

inline static uint32_t GetMilliTime()
{
    uint32_t sec,nsec;
    getWallTime(sec,nsec);
    return (uint64_t)sec*1000 + (uint64_t)nsec/1000000;
}

#endif

// quit after 100 milliseconds
IkFilterReturn MyTimeoutFilter(std::vector<dReal>&, RobotBase::ManipulatorPtr, co
      nst IkParameterization&, uint32_t starttime)
{
    if( GetMilliTime()-starttime > 100 ) {
        RAVELOG_INFO("quitting\n");
        return IKFR_Quit;
    }
    return IKFR_Success;
}

int main(int argc, char ** argv)
{
    string scenefilename = "data/pr2test1.env.xml";
    RaveInitialize(true);
    EnvironmentBasePtr penv = RaveCreateEnvironment();
    penv->Load(scenefilename);

    vector<RobotBasePtr> vrobots;
    penv->GetRobots(vrobots);
    RobotBasePtr probot = vrobots.at(0);
    probot->SetActiveManipulator("leftarm_torso");
    RobotBase::ManipulatorPtr pmanip = probot->GetActiveManipulator();

    // load inverse kinematics using ikfast
    ModuleBasePtr pikfast = RaveCreateModule(penv,"ikfast");
    penv->AddModule(pikfast,"");
    stringstream ssin,ssout;
    vector<dReal> vsolution;
    ssin << "LoadIKFastSolver " << probot->GetName() << " " << (int)IKP_Transform
      6D;
    if( !pikfast->SendCommand(ssout,ssin) ) {
        RAVELOG_ERROR("failed to load iksolver\n");
    }
    if( !pmanip->GetIkSolver()) {
        penv->Destroy();
        return 1;
```

```
    }

    probot->SetActiveDOFs(pmanip->GetArmIndices());
    vector<dReal> vlower,vupper;

    while(1) {
        {
            EnvironmentMutex::scoped_lock lock(penv->GetMutex()); // lock environ
      ment

            // move robot randomly
            probot->GetActiveDOFLimits(vlower,vupper);
            vector<dReal> v(pmanip->GetArmIndices().size());
            for(size_t i = 0; i < vlower.size(); ++i) {
                v[i] = vlower[i] + (vupper[i]-vlower[i])*RaveRandomFloat();
            }
            probot->SetActiveDOFValues(v);
            bool bincollision = !penv->CheckCollision(probot) && !probot->CheckSe
      lfCollision();

            uint32_t starttime = GetMilliTime();
            UserDataPtr filterhandle = pmanip->GetIkSolver()->RegisterCustomFilte
      r(0,boost::bind(MyTimeoutFilter,_1,_2,_3,starttime));
            bool bsuccess = pmanip->FindIKSolution(pmanip->GetIkParameterization(
      IKP_Transform6D),v,IKFO_CheckEnvCollisions);
            RAVELOG_INFO("in collision: %d, real success %d, time passed: %d\n",b
      incollision,bsuccess,GetMilliTime()-starttime);
        }
    }

    RaveDestroy();
    return 0;
}
```

12.8 orloadviewer.cpp

Author

Rosen Diankov

Shows how to load a robot into the openrave environment and start a viewer.

Usage:

```
orloadviewer [--num n] [--scene filename] viewername
```

- **--num** - Number of environments/viewers to create simultaneously
- **--scene** - The filename of the scene to load.

Example:

```
./orloadviewer --scene data/lab1.env.xml qtcoin
```

Full Example Code:

```
#include <openrave-core.h>
#include <vector>
#include <cstring>
#include <sstream>

#include <boost/thread/thread.hpp>
#include <boost/bind.hpp>

using namespace OpenRAVE;
using namespace std;

void SetViewer(EnvironmentBasePtr penv, const string& viewername)
{
    ViewerBasePtr viewer = RaveCreateViewer(penv,viewername);
    BOOST_ASSERT(!!viewer);

    // attach it to the environment:
    penv->AddViewer(viewer);

    // finally you call the viewer's infinite loop (this is why you need a separa
      te thread):
    bool showgui = true;
    viewer->main(showgui);

}

int main(int argc, char ** argv)
{
    //int num = 1;
    string scenefilename = "data/lab1.env.xml";
    string viewername = "qtcoin";

    // parse the command line options
    int i = 1;
    while(i < argc) {
        if((strcmp(argv[i], "-h") == 0)||(strcmp(argv[i], "-?") == 0)||(strcmp(ar
      gv[i], "/?") == 0)||(strcmp(argv[i], "--help") == 0)||(strcmp(argv[i], "-help") =
      = 0)) {
            RAVELOG_INFO("orloadviewer [--num n] [--scene filename] viewername\n"
      );
            return 0;
        }
        //        else if( strcmp(argv[i], "--num") == 0 ) {
        //            num = atoi(argv[i+1]);
        //            i += 2;
        //        }
        else if( strcmp(argv[i], "--scene") == 0 ) {
            scenefilename = argv[i+1];
            i += 2;
        }
        else
            break;
    }
    if( i < argc ) {
```

```
        viewername = argv[i++];
    }

    RaveInitialize(true); // start openrave core
    EnvironmentBasePtr penv = RaveCreateEnvironment(); // create the main environ
      ment
    RaveSetDebugLevel(Level_Debug);

    boost::thread thviewer(boost::bind(SetViewer,penv,viewername));
    penv->Load(scenefilename); // load the scene
    thviewer.join(); // wait for the viewer thread to exit
    penv->Destroy(); // destroy
    return 0;
}
```

12.9 ormulticontrol.cpp

Author

Rosen Diankov

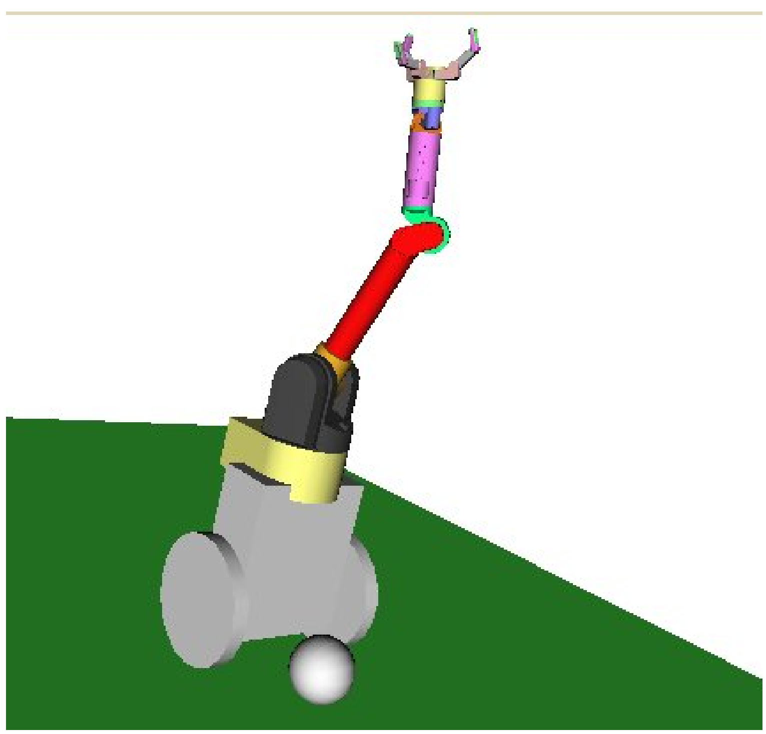


Figure 4: Base moves with velocity control, arm moves with position control.

Shows how to set two controllers for a robot using the MultiController class. The differential base moves with velocity control while the arm moves with position control.

Full Example Code:

```
#include <openrave-core.h>
#include <openrave/planningutils.h>
#include <vector>
#include <cstring>
#include <sstream>

#include <boost/thread/thread.hpp>
#include <boost/bind.hpp>

using namespace OpenRAVE;
using namespace std;

#ifdef _WIN32
#define WIN32_LEAN_AND_MEAN
#include <windows.h>
#define usleep(micro) Sleep(micro/1000)
#endif

void SetViewer(EnvironmentBasePtr penv, const string& viewername)
{
    ViewerBasePtr viewer = RaveCreateViewer(penv,viewername);
    penv->AddViewer(viewer);
    viewer->main(true);
}

int main(int argc, char ** argv)
{
    string scenefilename = "data/diffdrive_arm.env.xml";
    string viewername = "qtcoin";
    RaveInitialize(true);
    EnvironmentBasePtr penv = RaveCreateEnvironment();
    penv->SetDebugLevel(Level_Debug);

    boost::thread thviewer(boost::bind(SetViewer,penv,viewername)); // create the
       viewer
    usleep(400000); // wait for the viewer to init

    penv->Load(scenefilename);

    // attach a physics engine
    penv->SetPhysicsEngine(RaveCreatePhysicsEngine(penv,"ode"));
    penv->GetPhysicsEngine()->SetGravity(Vector(0,0,-9.8));

    vector<RobotBasePtr> vrobots;
    penv->GetRobots(vrobots);
    RobotBasePtr probot = vrobots.at(0);
    std::vector<dReal> q;

    vector<int> wheelindices, restindices;
    ControllerBasePtr wheelcontroller, armcontroller;
    // create the controllers, make sure to lock environment!
```

```
{
    EnvironmentMutex::scoped_lock lock(penv->GetMutex()); // lock environment


    MultiControllerPtr multi(new MultiController(penv));
    vector<int> dofindices(probot->GetDOF());
    for(int i = 0; i < probot->GetDOF(); ++i) {
        dofindices[i] = i;
    }
    probot->SetController(multi,dofindices,1); // control everything
    // set the velocity controller on all joints that have 'wheel' in their d
  escription
    for(std::vector<KinBody::JointPtr>::const_iterator itjoint = probot->GetJ
  oints().begin(); itjoint != probot->GetJoints().end(); ++itjoint) {
        if( (*itjoint)->GetName().find("wheel") != string::npos ) {
            for(int i = 0; i < (*itjoint)->GetDOF(); ++i) {
                wheelindices.push_back((*itjoint)->GetDOFIndex()+i);
            }
        }
        else {
            for(int i = 0; i < (*itjoint)->GetDOF(); ++i) {
                restindices.push_back((*itjoint)->GetDOFIndex()+i);
            }
        }
    }

    if(wheelindices.size() > 0 ) {
        wheelcontroller = RaveCreateController(penv,"odevelocity");
        multi->AttachController(wheelcontroller,wheelindices,0);
    }

    if( restindices.size() > 0 ) {
        armcontroller = RaveCreateController(penv,"idealcontroller");
        multi->AttachController(armcontroller,restindices,0);
    }
    else {
        RAVELOG_WARN("robot needs to have wheels and arm for demo to work\n")
  ;
    }
}

while(1) {
    {
        EnvironmentMutex::scoped_lock lock(penv->GetMutex()); // lock environ
  ment

        if( !!armcontroller ) {
            // set a trajectory on the arm and velocity on the wheels
            TrajectoryBasePtr traj = RaveCreateTrajectory(penv,"");
            probot->SetActiveDOFs(restindices);
            ConfigurationSpecification spec = probot->GetActiveConfigurationS
  pecification();
            int timeoffset = spec.AddDeltaTime();
            traj->Init(spec);
            probot->GetActiveDOFValues(q); // get current values
            vector<dReal> vdata(spec.GetDOF(),0);
            std::copy(q.begin(),q.end(),vdata.begin());
```

```
            traj->Insert(0,vdata);
            for(int i = 0; i < 4; ++i) {
                q.at(RaveRandomInt()%restindices.size()) += RaveRandomFloat()
      -0.5; // move a random axis
            }

            // check for collisions
            {
                RobotBase::RobotStateSaver saver(probot); // add a state save
      r so robot is not moved permenantly
                probot->SetActiveDOFValues(q);
                if( probot->CheckSelfCollision() ) { // don't check env colli
      sions since we have physics enabled
                    continue; // robot in collision at final point, so reject

                }
            }

            std::copy(q.begin(),q.end(),vdata.begin());
            vdata.at(timeoffset) = 2; // trajectory takes 2s
            traj->Insert(1,vdata);
            planningutils::RetimeActiveDOFTrajectory(traj,probot,true);
            armcontroller->SetPath(traj);
        }

        if( !!wheelcontroller ) {
            stringstream sout,ss; ss << "setvelocity ";
            for(size_t i = 0; i < wheelindices.size(); ++i) {
                ss << 2*(RaveRandomFloat()-0.5) << " ";
            }
            if( !wheelcontroller->SendCommand(sout,ss) ) {
                RAVELOG_WARN("failed to send velocity command\n");
            }
        }
    }

    // unlock the environment and wait for the arm controller to finish (whee
      l controller will never finish)
    if( !armcontroller ) {
        usleep(2000000);
    }
    else {
        while(!armcontroller->IsDone()) {
            usleep(1000);
        }
    }
  }

  thviewer.join(); // wait for the viewer thread to exit
  penv->Destroy(); // destroy
  return 0;
}
```

12.10 orplanning_ik.cpp

Author

Rosen Diankov

Shows how to use inverse kinematics and planners to move a robot's end-effector safely through the environment. The default manipulator is used for the robot.

Full Example Code:

```
#include <openrave-core.h>
#include <vector>
#include <sstream>
#include <boost/thread/thread.hpp>
#include <boost/bind.hpp>

using namespace OpenRAVE;
using namespace std;

#ifdef _WIN32
#define WIN32_LEAN_AND_MEAN
#include <windows.h>
#define usleep(micro) Sleep(micro/1000)
#endif

void SetViewer(EnvironmentBasePtr penv, const string& viewername)
{
    ViewerBasePtr viewer = RaveCreateViewer(penv,viewername);
    penv->AddViewer(viewer);
    viewer->main(true);
}

int main(int argc, char ** argv)
{
    string scenefilename = "data/pa10grasp2.env.xml";
    string viewername = "qtcoin";
    RaveInitialize(true);
    EnvironmentBasePtr penv = RaveCreateEnvironment();

    boost::thread thviewer(boost::bind(SetViewer,penv,viewername)); // create the
       viewer
    usleep(200000); // wait for the viewer to init
    penv->Load(scenefilename);
    usleep(100000); // wait for the viewer to init

    vector<RobotBasePtr> vrobots;
    penv->GetRobots(vrobots);
    RobotBasePtr probot = vrobots.at(0);

    // find a manipulator chain to move
    for(size_t i = 0; i < probot->GetManipulators().size(); ++i) {
        if( probot->GetManipulators()[i]->GetName().find("arm") != string::npos )
       {
            probot->SetActiveManipulator(i);
            break;
```

```
        }
    }
    RobotBase::ManipulatorPtr pmanip = probot->GetActiveManipulator();

    // load inverse kinematics using ikfast
    ModuleBasePtr pikfast = RaveCreateModule(penv,"ikfast");
    penv->AddModule(pikfast,"");
    stringstream ssin,ssout;
    vector<dReal> vsolution;
    ssin << "LoadIKFastSolver " << probot->GetName() << " " << (int)IKP_Transform
      6D;
    if( !pikfast->SendCommand(ssout,ssin) ) {
        RAVELOG_ERROR("failed to load iksolver\n");
    }
    if( !pmanip->GetIkSolver()) {
        penv->Destroy();
        return 1;
    }

    ModuleBasePtr pbasemanip = RaveCreateModule(penv,"basemanipulation"); // crea
      te the module
    penv->AddModule(pbasemanip,probot->GetName()); // load the module

    while(1) {
        {
            EnvironmentMutex::scoped_lock lock(penv->GetMutex()); // lock environ
      ment

            // find a new manipulator position and feed that into the planner. If
       valid, robot will move to it safely.
            Transform t = pmanip->GetEndEffectorTransform();
            t.trans += Vector(RaveRandomFloat()-0.5f,RaveRandomFloat()-0.5f,
      RaveRandomFloat()-0.5f);
            t.rot = quatMultiply(t.rot,quatFromAxisAngle(Vector(RaveRandomFloat()
      -0.5f,RaveRandomFloat()-0.5f,RaveRandomFloat()-0.5f)*0.2f));
            ssin.str("");
            ssin.clear();
            ssin << "MoveToHandPosition pose " << t;
            // start the planner and run the robot
            RAVELOG_INFO("%s\n",ssin.str().c_str());
            if( !pbasemanip->SendCommand(ssout,ssin) ) {
                continue;
            }
        }

        // unlock the environment and wait for the robot to finish
        while(!probot->GetController()->IsDone()) {
            usleep(1000);
        }
    }

    RaveDestroy();
    thviewer.join(); // wait for the viewer thread to exit
    return 0;
}
```

12.11 orplanning_module.cpp

Author

Rosen Diankov

Shows how to use a planner from a module to move the arm withut colliding into anything. The default values plan for all the joints of the robot.

Full Example Code:

```
#include <openrave-core.h>
#include <vector>
#include <sstream>
#include <boost/thread/thread.hpp>
#include <boost/bind.hpp>

using namespace OpenRAVE;
using namespace std;

#ifdef _WIN32
#define WIN32_LEAN_AND_MEAN
#include <windows.h>
#define usleep(micro) Sleep(micro/1000)
#endif

void SetViewer(EnvironmentBasePtr penv, const string& viewername)
{
    ViewerBasePtr viewer = RaveCreateViewer(penv,viewername);
    penv->AddViewer(viewer);
    viewer->main(true);
}

int main(int argc, char ** argv)
{
    string scenefilename = "data/wamtest1.env.xml";
    string viewername = "qtcoin";
    RaveInitialize(true);
    EnvironmentBasePtr penv = RaveCreateEnvironment();

    boost::thread thviewer(boost::bind(SetViewer,penv,viewername)); // create the
       viewer
    usleep(200000); // wait for the viewer to init
    penv->Load(scenefilename);
    usleep(100000); // wait for the viewer to init

    vector<RobotBasePtr> vrobots;
    penv->GetRobots(vrobots);
    RobotBasePtr probot = vrobots.at(0);
    vector<dReal> vlower,vupper,v(probot->GetDOF());
    probot->GetDOFLimits(vlower,vupper);

    // set all dofs as active
    vector<int> vindices(probot->GetDOF());
    for(size_t i = 0; i < vindices.size(); ++i) {
        vindices[i] = i;
```

```
    }
    probot->SetActiveDOFs(vindices);

    ModuleBasePtr pbasemanip = RaveCreateModule(penv,"basemanipulation"); // crea
      te the module
    penv->AddModule(pbasemanip,probot->GetName()); // load the module

    while(1) {
        {
            EnvironmentMutex::scoped_lock lock(penv->GetMutex()); // lock environ
      ment

            // find a set of free joint values for the robot
            {
                RobotBase::RobotStateSaver saver(probot); // save the state
                while(1) {
                    for(size_t i = 0; i < vlower.size(); ++i) {
                        v[i] = vlower[i] + (vupper[i]-vlower[i])*RaveRandomFloat(
      );
                    }
                    probot->SetActiveDOFValues(v);
                    if( !penv->CheckCollision(probot) && !probot->CheckSelfCollis
      ion() ) {
                        break;
                    }
                }
                // robot state is restored
            }

            stringstream cmdin,cmdout;
            cmdin << "MoveActiveJoints goal ";
            for(size_t i = 0; i < v.size(); ++i) {
                cmdin << v[i] << " ";
            }

            // start the planner and run the robot
            RAVELOG_INFO("%s\n",cmdin.str().c_str());
            if( !pbasemanip->SendCommand(cmdout,cmdin) ) {
                continue;
            }
        }

        // unlock the environment and wait for the robot to finish
        while(!probot->GetController()->IsDone()) {
            usleep(1000);
        }
    }

    thviewer.join(); // wait for the viewer thread to exit
    penv->Destroy(); // destroy
    return 0;
}
```

12.12 orplanning_planner.cpp

Author

Rosen Diankov

Shows how to use a planner by directly creating the planner and setting the module parameters. The default values plan for the arm joints of a particular manipulator.

Full Example Code:

```
#include <openrave-core.h>
#include <vector>
#include <sstream>
#include <boost/thread/thread.hpp>
#include <boost/bind.hpp>
#include <boost/format.hpp>

using namespace OpenRAVE;
using namespace std;

#ifdef _WIN32
#define WIN32_LEAN_AND_MEAN
#include <winsock2.h>
#define usleep(micro) Sleep(micro/1000)
#endif

void SetViewer(EnvironmentBasePtr penv, const string& viewername)
{
    ViewerBasePtr viewer = RaveCreateViewer(penv,viewername);
    penv->AddViewer(viewer);
    viewer->main(true);
}

int main(int argc, char ** argv)
{
    string scenefilename = "data/hanoi_complex2.env.xml";
    string viewername = "qtcoin";
    RaveInitialize(true);
    EnvironmentBasePtr penv = RaveCreateEnvironment();
    RaveSetDebugLevel(Level_Debug);

    boost::thread thviewer(boost::bind(SetViewer,penv,viewername)); // create the
       viewer
    usleep(200000); // wait for the viewer to init
    penv->Load(scenefilename);
    usleep(100000); // wait for the viewer to init

    vector<RobotBasePtr> vrobots;
    penv->GetRobots(vrobots);
    RobotBasePtr probot = vrobots.at(0);
    // find the longest manipulator chain to move
    RobotBase::ManipulatorPtr pmanip = probot->GetManipulators().at(0);
    for(size_t i = 1; i < probot->GetManipulators().size(); ++i) {
        if( pmanip->GetArmIndices().size() < probot->GetManipulators()[i]->GetArm
      Indices().size() ) {
```

```
        pmanip = probot->GetManipulators()[i];
    }
}
RAVELOG_INFO(str(boost::format("planning with manipulator %s\n")%pmanip->GetN
  ame()));

probot->SetActiveDOFs(pmanip->GetArmIndices());
vector<dReal> vlower,vupper;
probot->GetActiveDOFLimits(vlower,vupper);

// create a planner
PlannerBasePtr planner = RaveCreatePlanner(penv,"birrt");
PlannerBase::PlannerParametersPtr params(new PlannerBase::PlannerParameters()
  );
params->_nMaxIterations = 4000; // max iterations before failure
params->SetRobotActiveJoints(probot); // set planning configuration space to
  current active dofs
params->vgoalconfig.resize(probot->GetActiveDOF());

while(1) {
    GraphHandlePtr pgraph;
    {
        EnvironmentMutex::scoped_lock lock(penv->GetMutex()); // lock environ
  ment

        // find a set of free joint values for the robot
        {
            RobotBase::RobotStateSaver saver(probot); // save the state
            while(1) {
                for(size_t i = 0; i < vlower.size(); ++i) {
                    params->vgoalconfig[i] = vlower[i] + (vupper[i]-vlower[i]
  )*RaveRandomFloat();
                }
                probot->SetActiveDOFValues(params->vgoalconfig);
                if( !penv->CheckCollision(probot) && !probot->CheckSelfCollis
  ion() ) {
                    break;
                }
            }
            // robot state is restored
        }

        RAVELOG_INFO("starting to plan\n");
        probot->GetActiveDOFValues(params->vinitialconfig);
        if( !planner->InitPlan(probot,params) ) {
            continue;
        }

        // create a new output trajectory
        TrajectoryBasePtr ptraj = RaveCreateTrajectory(penv,"");
        if( !planner->PlanPath(ptraj) ) {
            RAVELOG_WARN("plan failed, trying again\n");
            continue;
        }

        // draw the end effector of the trajectory
        {
```

```
            RobotBase::RobotStateSaver saver(probot); // save the state of th
e robot since will be setting joint values
            vector<RaveVector<float> > vpoints;
            vector<dReal> vtrajdata;
            for(dReal ftime = 0; ftime <= ptraj->GetDuration(); ftime += 0.01
) {
                ptraj->Sample(vtrajdata,ftime,probot->GetActiveConfigurationS
pecification());
                probot->SetActiveDOFValues(vtrajdata);
                vpoints.push_back(pmanip->GetEndEffectorTransform().trans);
            }
            pgraph = penv->drawlinestrip(&vpoints[0].x,vpoints.size(),sizeof(
vpoints[0]),1.0f);
        }

        // send the trajectory to the robot
        probot->GetController()->SetPath(ptraj);
    }


    // wait for the robot to finish
    while(!probot->GetController()->IsDone()) {
        usleep(1000);
    }
  }

  thviewer.join(); // wait for the viewer thread to exit
  penv->Destroy(); // destroy
  return 0;
}
```

12.13 orpythonbinding.cpp

Author

Rosen Diankov

Shows how to creating python bindings with an OpenRAVE C++ plugin. The demo registers a python function to be called inside the environment simulation thread using a Module interface.

The compilation procedure will produce a orpythonbinding shared object or DLL, which can then be directly included into python.

The following python example will register 'mysimfunction' with the enviornment thread, and run it until it returns true.

```
from openravepy import *
env=openravepy.Environment()
RaveSetDebugLevel(DebugLevel.Debug)
import orpythonbinding
orpythonbinding.Init(RaveGlobalState())
totaltime = 0
def mysimfunction(elapsedtime):
 global totaltime
```

```
    totaltime += elapsedtime
    print 'this is the time',totaltime
 # return True to end the thread
    return totaltime > 5

   module = orpythonbinding.RegisterSimulationFunction(RaveGetEnvironmentId(env),mysimfunction)
   while True:
    sleep(1)
```

Full Example Code:

```
#include <openrave/openrave.h>

#include <boost/python.hpp>
#include <boost/python/exception_translator.hpp>
#include <boost/python/stl_iterator.hpp>
#include <pyconfig.h>

#include <exception>
#include <boost/shared_ptr.hpp>
#include <boost/format.hpp>
#include <boost/assert.hpp>

#include <vector>
#include <cstring>
#include <sstream>

using namespace OpenRAVE;
using namespace std;

class FunctionUserData : public UserData
{
public:
    virtual ~FunctionUserData() {
    }
    boost::python::object simulationfn;
};

class PythonBindingModule : public ModuleBase
{
public:
    PythonBindingModule(EnvironmentBasePtr penv, std::istream&) : ModuleBase(penv
      ) {
        SetUserData(UserDataPtr(new FunctionUserData()));
    }
    virtual ~PythonBindingModule() {
        RAVELOG_DEBUG("destroying python binding\n");
    }

    virtual bool SimulationStep(dReal fElapsedTime) {
        boost::shared_ptr<FunctionUserData> p = boost::dynamic_pointer_cast<Funct
      ionUserData>(GetUserData());
        bool ret = false;
        if( !!p ) {
            PyGILState_STATE gstate = PyGILState_Ensure();
```

```
            try {
                ret = p->simulationfn(fElapsedTime);
            }
            catch(...) {
                RAVELOG_WARN("unknown exception in python callback, please regist
      er again:\n");
                PyErr_Print();
                ret = true;
            }
            PyGILState_Release(gstate);
            if( ret ) {
                GetEnv()->Remove(shared_from_this());
            }
        }
        return ret;
    }
};

boost::shared_ptr<void> g_PythonBindingInterfaceHandle;

InterfaceBasePtr PythonBindingCreateInterface(EnvironmentBasePtr penv, std::istre
      am& istream)
{
    return InterfaceBasePtr(new PythonBindingModule(penv,istream));
}

InterfaceBasePtr RegisterSimulationFunction(int environmentid, boost::python::obj
      ect simulationfn)
{
    ModuleBasePtr module = RaveCreateModule(RaveGetEnvironment(environmentid), "P
      ythonBinding");
    if( !!module ) {
        boost::shared_ptr<FunctionUserData> p = boost::dynamic_pointer_cast<Funct
      ionUserData>(module->GetUserData());
        p->simulationfn = simulationfn;
        module->GetEnv()->AddModule(module,"");
    }
    return InterfaceBasePtr(module);
}

void Init(UserDataPtr globalstate)
{
    RaveInitializeFromState(globalstate);
    if( !g_PythonBindingInterfaceHandle ) {
        g_PythonBindingInterfaceHandle = RaveRegisterInterface(PT_Module, "Python
      Binding", OPENRAVE_MODULE_HASH,OPENRAVE_ENVIRONMENT_HASH, PythonBindingCreateInte
      rface);
    }
}

BOOST_PYTHON_MODULE(orpythonbinding)
{
    boost::python::def("Init", Init, boost::python::args("globalstate"), "initial
      izes the python bindings with the openrave global state");
    boost::python::def("RegisterSimulationFunction", RegisterSimulationFunction,
      boost::python::args("environmentid","simulationfn"));
};
```

12.14 orshowsensors.cpp

Author

Rosen Diankov

Shows how to toggle sensor power and rendering options

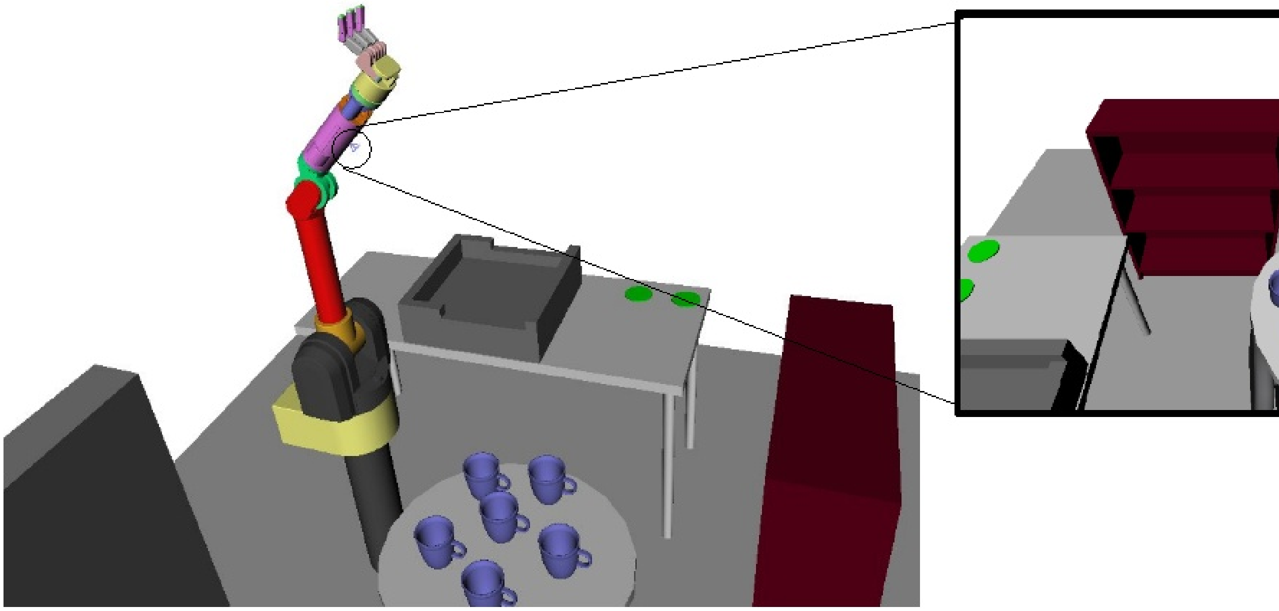


Figure 5: Camera Sensor.

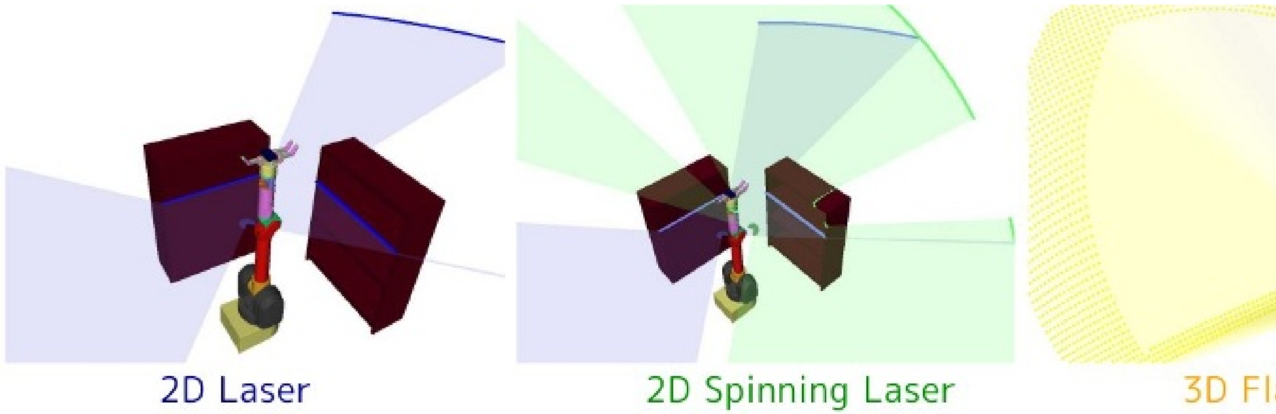


Figure 6: Laser Sensor.

Full Example Code:

```
#include <openrave-core.h>
#include <vector>

#include <boost/thread/thread.hpp>
#include <boost/bind.hpp>

#ifdef _WIN32
#define WIN32_LEAN_AND_MEAN
#include <winsock2.h>
#define usleep(micro) Sleep(micro/1000)
#endif

using namespace OpenRAVE;
using namespace std;

void SetViewer(EnvironmentBasePtr penv, const string& viewername)
{
    ViewerBasePtr viewer = RaveCreateViewer(penv,viewername);
    BOOST_ASSERT(!!viewer);

    // attach it to the environment:
    penv->AddViewer(viewer);

    // finally you call the viewer's infinite loop (this is why you need a separa
      te thread):
    bool showgui = true;
    viewer->main(showgui);
}

int main(int argc, char ** argv)
{
    RaveInitialize(true); // start openrave core
```

```
    EnvironmentBasePtr penv = RaveCreateEnvironment(); // create the main environ
      ment
    boost::thread thviewer(boost::bind(SetViewer,penv,"qtcoin"));
    penv->Load("data/testwamcamera.env.xml");
    size_t ienablesensor = 0;
    // get all the sensors, this includes all attached robot sensors
    std::vector<SensorBasePtr> sensors;
    penv->GetSensors(sensors);
    while(1) {
        for(size_t isensor = 0; isensor < sensors.size(); ++isensor) {
            sensors[isensor]->Configure(isensor == ienablesensor ? SensorBase::CC
      _PowerOn : SensorBase::CC_PowerOff);
            sensors[isensor]->Configure(isensor == ienablesensor ? SensorBase::CC
      _RenderDataOn : SensorBase::CC_RenderDataOff);
        }
        ienablesensor = (ienablesensor+1)%sensors.size();
        usleep(5000000); // 5s
    }
    return 0;
}
```

12.15 ortrajectory.cpp

Author

Rosen Diankov

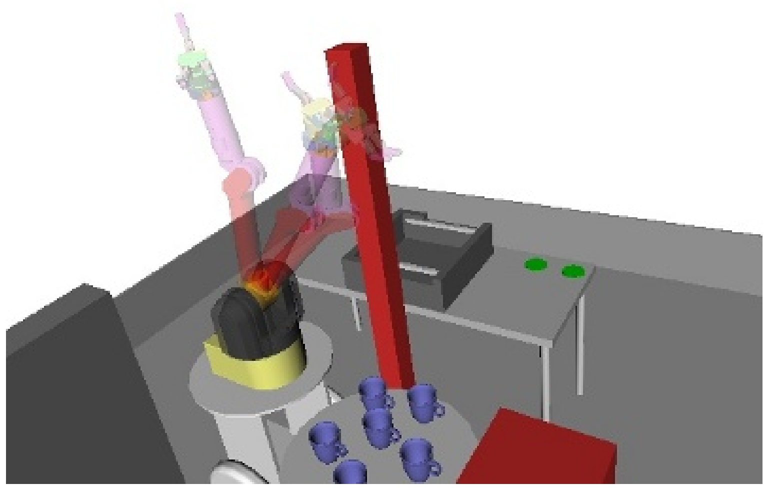


Figure 7: Robot moving in random configurations.

Shows how to send a cubicaly interpolated trajectory to the robot controller. The actual trajectory consists of two points: the current configuration and the target configuration.

```
    TrajectoryBasePtr traj = RaveCreateTrajectory(penv,"");
    traj->Init(probot->GetActiveConfigurationSpecification());
    probot->GetActiveDOFValues(q); // get current values
    traj->Insert(0,q);
    q[0] = 0.5;
    traj->Insert(1,q);
    planningutils::RetimeActiveDOFTrajectory(probot,traj);
```

The demo also adds a collision check at the target point to make sure robot is going to a collision free configuration.

```
    {
        RobotBase::RobotStateSaver saver(probot); // add a state saver so robot i
      s not moved permenantly
        probot->SetDOFValues(q);
        if( penv->CheckCollision(RobotBaseConstPtr(probot)) ) {
            continue; // robot in collision at final point, so reject
        }
    }
```

In order for the path itself to be collision free, we would have to use planners.

Full Example Code:

```
#include <openrave-core.h>
#include <vector>
#include <cstring>
#include <sstream>

#include <boost/thread/thread.hpp>
#include <boost/bind.hpp>
#include <openrave/planningutils.h>

using namespace OpenRAVE;
using namespace std;

#ifdef _WIN32
#define WIN32_LEAN_AND_MEAN
#include <windows.h>
#define usleep(micro) Sleep(micro/1000)
#endif

void SetViewer(EnvironmentBasePtr penv, const string& viewername)
{
    ViewerBasePtr viewer = RaveCreateViewer(penv,viewername);
    penv->AddViewer(viewer);
    viewer->main(true);
}

int main(int argc, char ** argv)
{
    string scenefilename = "data/lab1.env.xml";
    string viewername = "qtcoin";
    RaveInitialize(true);
```

```
    EnvironmentBasePtr penv = RaveCreateEnvironment();
    penv->SetDebugLevel(Level_Debug);

    boost::thread thviewer(boost::bind(SetViewer,penv,viewername)); // create the
       viewer
    usleep(300000); // wait for the viewer to init

    penv->Load(scenefilename);
    vector<RobotBasePtr> vrobots;
    penv->GetRobots(vrobots);
    RobotBasePtr probot = vrobots.at(0);
    std::vector<dReal> q;

    while(1) {
        {
            EnvironmentMutex::scoped_lock lock(penv->GetMutex()); // lock environ
      ment

            TrajectoryBasePtr traj = RaveCreateTrajectory(penv,"");
            traj->Init(probot->GetActiveConfigurationSpecification());
            probot->GetActiveDOFValues(q); // get current values
            traj->Insert(0,q);
            q[RaveRandomInt()%probot->GetDOF()] += RaveRandomFloat()-0.5; // move
       a random axis

            // check for collisions
            {
                RobotBase::RobotStateSaver saver(probot); // add a state saver so
       robot is not moved permenantly
                probot->SetDOFValues(q);
                if( penv->CheckCollision(RobotBaseConstPtr(probot)) ) {
                    continue; // robot in collision at final point, so reject
                }
            }

            traj->Insert(1,q);
            planningutils::RetimeActiveDOFTrajectory(traj,probot);
            probot->GetController()->SetPath(traj);
            // setting through the robot is also possible: probot->SetMotion(traj
      );
        }
        // unlock the environment and wait for the robot to finish
        while(!probot->GetController()->IsDone()) {
            usleep(1000);
        }
    }

    thviewer.join(); // wait for the viewer thread to exit
    penv->Destroy(); // destroy
    return 0;
}
```

12.16 plugincpp.cpp

Author

Rosen Diankov

Creates a simple OpenRAVE::ModuleBase interface.

Full Example Code:

```
#include <openrave/openrave.h>
#include <openrave/plugin.h>
#include <boost/bind.hpp>

using namespace std;
using namespace OpenRAVE;

class MyModule : public ModuleBase
{
public:
    MyModule(EnvironmentBasePtr penv) : ModuleBase(penv)
    {
        __description = "A very simple plugin.";
        RegisterCommand("numbodies",boost::bind(&MyModule::NumBodies,this,_1,_2),
      "returns bodies");
        RegisterCommand("load",boost::bind(&MyModule::Load, this,_1,_2),"loads a
      given file");
    }
    virtual ~MyModule() {
    }

    void Destroy() {
        RAVELOG_INFO("module unloaded from environment\n");
    }

    int main(const string& cmd)
    {
        RAVELOG_INFO("module initialized cmd; %s\n", cmd.c_str());
        return 0;
    }

    bool NumBodies(ostream& sout, istream& sinput)
    {
        vector<KinBodyPtr> vbodies;
        GetEnv()->GetBodies(vbodies);
        sout << vbodies.size();     // publish the results
        return true;
    }

    bool Load(ostream& sout, istream& sinput)
    {
        string filename;
        sinput >> filename;
        bool bSuccess = GetEnv()->Load(filename.c_str());     // load the file
        return bSuccess;
    }
```

```
};

InterfaceBasePtr CreateInterfaceValidated(InterfaceType type, const std::string&
      interfacename, std::istream& sinput, EnvironmentBasePtr penv)
{
    if((type == PT_Module)&&(interfacename == "mymodule")) {
        return InterfaceBasePtr(new MyModule(penv));
    }
    return InterfaceBasePtr();
}

void GetPluginAttributesValidated(PLUGININFO& info)
{
    info.interfacenames[PT_Module].push_back("MyModule");
}

OPENRAVE_PLUGIN_API void DestroyPlugin()
{
    RAVELOG_INFO("destroying plugin\n");
}
```