ROBOOP

A Robotics Object Oriented Package in C++

version 1.31

Documentation

Richard Gourdeau
Département de génie électrique
École Polytechnique de Montréal
C.P. 6079, Succ. Centre-Ville,
Montréal, Québec, Canada, H3C 3A7
email: richard.gourdeau@polymtl.ca

December 14, 2006

Contents

Chapter 1

Introduction

1.1 Description

This package (ROBOOP[1]) is a C++ robotics object oriented programming toolbox suitable for synthesis, and simulation of robotic manipulator models in an environment that provides "MATLAB like" features for the treatment of matrices. Its is a portable tool that does not require the use of commercial software. A class named `Robot` provides the implementation of the kinematics, the dynamics and the linearized dynamics of serial robotic manipulators. A class named `Stewart` provides the implementation of the kinematics, the dynamics for Stewart type parallel manipulators.

1.2 Requirements

This work uses the matrix library NEWMAT11 [2] developed by Robert Davies. Hence, the requirement for the ROBOOP are the same as for the NEWMAT11. Although make files are only provided for the Borland C++ 4.5 and 5.x, Visual C++ 6.0, Visual C++ 7.0 (.NET), and GNU G++ compilers (OpenWatcom C++ support has been suspended until the compiler provides "full" STL implementation), other compilers supporting the STL could be used. See the file `nm11.htm` in the `newmat` directory for more details.

The library Boost is used by ROBOOP. Under most Linux distributions and Cygwin, Boost is a standard package (just install it). For Borland C++

[1] Program source and documentation are available from the URL: `http://www.cours.polymtl.ca/roboop/`

[2] available from the site `http://www.robertnz.net/`

and Visual C++, you can extract the following file (boost_inc.zip) in the roboop/source directory [3]

In order to use the graphic features of this package, the software gnuplot[4] (version 3.5 on later) must be installed in the PATH of your computer. The binary name is wgnuplot.exe under Windows 95/98/NT/2000 (Win32) and gnuplot under most of other platforms, you should edit the file gnugraph.h if the binary name is different.

1.3 Compiling

1.3.1 Linux

Under Linux, you can compile using one of the three following ways (in the roboop directory):

1. Using the command

   ```
   make -f makefile.gcc
   ```

2. If you have CMake installed then use

   ```
   cmake .
   make
   ```

3. If you have Bakefile installed then use

   ```
   bakefile -f gnu roboop.bkl
   make
   ```

1.3.2 MS Windows

Borland Compiler : you can compile using one of the three following ways:

1. Using the command

   ```
   make -f makefile.bc5
   ```

2. If you have CMake installed then use the CMake program from the Start menu to generate a Borland makefile, then from the prompt (in the roboop directory) execute the command

[3] simpler but will not provide you with all the Boost features

[4] gnuplot is freely available from the following location: http://www.gnuplot.info/

```
make
```

3. If you have Bakefile installed then use (in the `roboop` directory)

   ```
   bakefile -f borland roboop.bkl
   make
   ```

Cygwin : you can compile using one of the three following ways (in the `roboop` directory):

1. Using the command

   ```
   make -f makefile.gw32
   ```

2. If you have CMake installed then use

   ```
   cmake .
   make
   ```

3. If you have Bakefile installed then use

   ```
   ln -s /usr/include/boost-1_33_1/boost/ /usr/include/boost
   bakefile -f gnu roboop.bkl
   make
   ```

Visual C++ : you can compile using one of the following ways:

1. Using the command

   ```
   nmake -f makefile.vcpp
   ```

2. Opening the Visual C++ 6.0 Workspace `roboop.dsw` or the Visual C++ 7.0 Solution `roboop.sln` and building the targets.
3. If you have CMake installed then use the `CMake` program from the Start menu to generate NMake makefiles, then from the prompt (in the `roboop` directory) execute the command

   ```
   nmake
   ```

4. If you have CMake installed then use the `CMake` program from the Start menu to generate one of the different Visual Studio project formats available, then by opening the Visual C++ Workspace or Solution generated and building the targets.
5. If you have Bakefile installed then use (in the `roboop` directory)

   ```
   bakefile -f msvc roboop.bkl
   nmake
   ```

or

```
bakefile -f msvc6proj roboop.bkl
```

and by opening the Visual C++ Workspace generated and building the targets.

1.3.3 Mac OSX

You can compile using one of the following ways (in the `roboop` directory):

1. Using the command

   ```
   make -f makefile.gccOSX
   ```

2. CMake and Bakefile have not been tested yet but might work with the Linux directives!

1.4 Copyright

1.5 Version history

version 1.31 (2006/12/14)

The project can now use CMake or Bakefile for automated `makefile` generation. In future releases, *hand made* makefiles and project files will be replaced by the output of CMake or Bakefile.

Corrected bug in irotk (reported by Chris Lightcap).

The program files in this version are the following revisions:

```
Id: bench.cpp,v 1.20 2005/07/01 16:16:35
Id: clik.cpp,v 1.6 2006/05/16 16:11:15
Id: comp_dq.cpp,v 1.17 2004/07/06 02:16:36
Id: comp_dqp.cpp,v 1.16 2004/07/06 02:16:36
Id: config.cpp,v 1.20 2006/05/16 19:24:26
Id: controller.cpp,v 1.3 2005/11/15 19:06:13
Id: control_select.cpp,v 1.7 2006/05/16 19:24:26
Id: delta_t.cpp,v 1.17 2005/07/01 16:11:45
Id: demo_2dof_pd.cpp,v 1.2 2006/05/16 16:27:43
Id: demo.cpp,v 1.34 2006/05/16 16:27:43
Id: dynamics.cpp,v 1.34 2006/05/19 18:32:30
Id: dynamics_sim.cpp,v 1.6 2006/05/19 21:05:57
Id: gnugraph.cpp,v 1.44 2006/05/19 17:49:58
Id: homogen.cpp,v 1.15 2006/11/15 18:35:17
Id: invkine.cpp,v 1.8 2006/05/16 16:11:15
Id: kinemat.cpp,v 1.31 2004/08/16 00:37:53
Id: quaternion.cpp,v 1.18 2005/11/15 19:25:58
Id: robot.cpp,v 1.50 2006/05/16 19:24:26
Id: rtest.cpp,v 1.15 2005/07/01 17:44:53
Id: sensitiv.cpp,v 1.13 2004/07/06 02:16:37
Id: stewart.cpp,v 1.6 2006/05/16 19:24:26
Id: trajectory.cpp,v 1.8 2006/05/16 19:24:26
Id: utils.cpp,v 1.26 2006/05/16 16:11:15
Id: clik.h,v 1.6 2006/05/16 16:11:15
Id: config.h,v 1.18 2006/05/16 19:24:26
Id: controller.h,v 1.5 2006/05/16 16:11:15
Id: control_select.h,v 1.4 2006/05/16 16:11:15
Id: dynamics_sim.h,v 1.4 2006/05/16 16:11:15
Id: gnugraph.h,v 1.13 2006/05/16 19:24:26
Id: quaternion.h,v 1.12 2005/11/15 19:25:58
Id: robot.h,v 1.52 2006/05/16 16:11:15
Id: stewart.h,v 1.2 2006/05/16 16:11:15
Id: trajectory.h,v 1.10 2006/05/16 19:24:26
Id: utils.h,v 1.10 2006/05/16 16:11:15
Id: makefile.bc5,v 1.18 2006/08/17 17:36:17
Id: makefile.gcc,v 1.22 2006/08/17 17:36:17
Id: makefile.gccOSX,v 1.4 2006/08/17 17:36:17
Id: makefile.gw32,v 1.15 2006/08/17 17:36:17
Id: makefile.vcpp,v 1.11 2006/08/17 17:36:17
```

version 1.30 (2006/08/17)

Upgraded the matrix library to NEWMAT11 (beta) April 2006 enabling compilation under GNU g++ 4.1.x.

The program files in this version are the following revisions:

version 1.29 (2006/05/19)

OpenWatcom support is (temporally) suspended. Fixed gear ratio bug for viscous friction (reported by Carmine Lia). Fix `set_q`, `set_qp` bug in `xdot` (reported by Philip Gruebele)

The following changes have been contributed by Etienne Lachance

- "Clean up" of some header files.
- Member functions add and select are now in template form.
- Using Boost shared pointers in `gnugraph`.
- The inverse kinematics function (`inv_kin`) should return the solution without changing the robot position (reported by J.D. Yamokoski).
- Functions `Rhino_DH`, `Puma_DH`, `Schilling_DH`, `Rhino_mDH`, `Puma_mDH` and `Schilling_mDH` use `const Robot_basic` reference instead of `const Robot_basic` pointer.
- Prevent exceptions from leaving `Robot_basic` destructor.
- Catch exception by reference instead of by value.

version 1.28 (2005/12/07)

The following changes have been contributed by Etienne Lachance

- Removing unnecessary copy constructor and the assignment operator (operator=) in many classes.
- In the `Quaternion` class, the operator* and operator/ are now non-member functions when one of the operand is a real, it now supports q2 = c * q1 and ¿ q2 = q1 * c

version 1.27 (2005/10/11)

It is now possible to turn off warning messages in the `Config` class.

version 1.26 (2005/07/05)

- New Class `Stewart` contributed by Samuel Belanger (intergated by Etienne Lachance and Richard Gourdeau): new files `stewart.h` and `stewart.cpp` and modified `bench.cpp`.

- Fixed max() bug for VC++ 6.0 (`utils.cpp`).
- Typos in Doxygen documentation.

version 1.25 (2005/06/13) Fixed `catch(bad_alloc)` in constructors.

The following changes have been contributed by Etienne Lachance

- The desired joint acceleration was missing in the computed torque method (bug reported by Carmine Lia).
- Added missing file message in `trajectory.cpp`

The following changes have been contributed by Carmine Lia

- Added `defined(__MINGW32__)` for temp files in `gnugraph.cpp`.
- Added `pinv` in `utils.cpp`.

version 1.24 (2005/03/18)

The following changes have been contributed by Brian Galardo, Jean-Pascal Joary, Etienne Lachance:

- Added member functions `Robot::inv_schilling`, `mRobot::inv_schilling` and `mRobot_min_para::inv_schilling` for the Schilling Titan II robot arm,
- Fixed previous bug on Rhino and Puma inverse kinematics.

by Etienne Lachance:

- Some "clean-up" in the `config.h` and `config.cpp` files,

and by Stephen Webb :

- minor bug in constructor `Robot_basic(const Robot_basic & x)`.

version 1.23 (2004/09/18)

The following change has been contributed by Etienne Lachance:

- Configuration files can use degrees for the angles with the option `angle_in_degree` set to 1.

version 1.22 (2004/09/10)

The following change has been contributed by Etienne Lachance:

- In `config.cpp`: parameter value can now contain space and fixed print member function.

Carl Glen Henshaw provided a makefile for MAC OS X.

version 1.21 (2004/08/16)

The following changes have been contributed by Etienne Lachance

- Fixed some missing `use_namespace #define`.
- Merge all `select_*` and `add_*` functions into overloaded `select()` and `add()` functions.
- made `gnuplot.cpp` and `config.cpp` independent of `robot.h` and `utils.h`.
- New constructors for `Robot` and `mRobot` based on input matrices (this change is NOT backward compatible)

The following changes have been contributed by Ethan Tira-Thompson

- Supports for `Link::immobile` flag so jacobians and deltas are 0 for immobile joints.
- Jacobians will only contain entries for mobile joints - otherwise NaNs result in later processing.
- Added parameters to jacobian functions to generate for frames other than the end effector.
- Can now do inverse kinematics for frames other than end effector.
- Tolerance in `inv_kin` based on `USING_FLOAT` from newmat's `include.h`

The program files in this version are the following revisions:

version 1.20 (2004/07/02)

The following changes have been contributed by Ethan Tira-Thompson

- Added support for newmat's `use_namespace #define`, using `ROBOOP` namespace.
- Fixed some problem using `float` as `Real` type.

The following changes have been contributed by Etienne Lachance

- Added the following class: `Dynamics`, `Trajectory_Select`, `Proportional_Derivative` and `Control_Select`.
- Added a new demo program, call `demo_2dof_pd`. This new demo program shows how to use the class mentioned above.

- Protection added on input vector of the `trans` function.
- Added a `joint_offset` logic. This idea has been proposed by Ethan Tira-Thompson.
- Added Doxygen documentation.
- Replace files `impedance.*` by `controller.*`.

version 1.19 (2004/05/12) Upgraded the matrix library from NEWMAT10 to NEWMAT11 (beta). Visual C++ .NET and Borland C++ Builder 6 compilers are now supported. Updated documentation.

version 1.18 (2004/05/05) ROBOOP is relicensed to the GNU Lesser General Public License. Updated documentation.

The following changes have been contributed by Vincent Drolet and Etienne Lachance:

- Added the following members function in class Robot: `inv_kin_rhino`, `inv_kin_puma` and `robotType_inv_kin`.

version 1.17 (2004/04/02) Numerous warning messages were corrected under VC++. Updated documentation.

The following changes have been contributed by Etienne Lachance:

- Added class `Impedance` which implements the impedance controller.
- Added function `perturb_robot`.
- Added class `Resolve_acc` which implements the resolve rate acceleration position controller.
- Added class `Computed_torque_method` which implements the computed torque method position controller.
- Class *Config* can now write data into a configuration file.
- Fixed bugs in `Quaternion` class member functions: `exponential` and `logarithm`.
- Added `Quaternion` class member function `power`.
- Added the following `Quaternion` class non member functions: `Omega`, `Slerp`, `Slerp_prime`, `Squad` and `Squad_prime`.
- Provided `Spl_Quaternion` class to generate quaternions cubic splines.

- Added class Spl_Cubic to generate cubic splines.
- Added class Spl_path to generate 3D cubic splines.
- Provided CLIK class for closed loop inverse kinematics.
- Added member functions G and C in all robot classes.

version 1.16 (2003/09/24) The OpenWatcom C++ compiler is now supported. Updated documentation.

version 1.15 (2003/06/18) The following changes have been contributed by Etienne Lachance:

- Updated documentation.
- Definitions in file gnugraph.cpp are now in gnugraph.h.
- Class Plot2d, GNUcurve are now using STL string instead of char*.
- Added member functions jacobian_dot() and jacobian_DLS_inv() in all robot classes.
- Added class Config to read configuration file.
- Replaced Robot_basic(const char *filename) by Robot_basic(const string & filename). The new constructor uses the class Config.
- Provided Plot_file class to generate graphics from a data file.
- Added the following Quaternion class member functions: exponential, logarithm, dot_product, dot, E.
- Fixed bugs in IO_matrix_file class.
- Developed linearized equations for modified DH notations. The equations are implemented in dq_torque, dqp_torque, dtau_dq and dtau_dqp.
- Added examples in demo.cpp related to IO_matrix_file, Plot_file and Config.

version 1.14 (2003/04/17) Updated documentation. The Watcom compiler is no longer supported (problems with STL and streams). The following changes have been contributed by Etienne Lachance:

- The classes RobotMotor and mRobotMotor no longer exist and are now integrated in the Robot and mRobot classes.
- The Robot and mRobot classes are now derived from the Robot_basic virtual class.

- Removed class `mlink`. DH and modified DH parameters are now included in `link`.
- Added `kine_pd()`.
- Created a new `torque` member function that allowed to have load on last link.
- Fixed bug in modified DH dynamics.
- Added a class `Quaternion`.
- Added the program `rtest` to compare results with Peter Corke MATLAB toolbox.
- Added member function `set_plot2d` to generate plots using the `Plot2d` class.
- Added utility class `IO_matrix_file` dealing with data files (not documented yet).

version 1.13 (2002/08/09) Moved the arrays of `ColumnVector` to the constructors for the dynamics and linearized dynamics for a $\approx 10\%$ gain in speed (thanks to Etienne Lachance for the suggestion). Added the `mRobot` and `mRobotMotor` classes using the modified Denavit-Hartenberg notation. Updated documentation.

version 1.12 (2002/02/04) Upgraded the matrix library from NEWMAT09 to NEWMAT10.

version 1.11 (2001/06/06) Fixed bugs for prismatic joints in the dynamics routines (reported by Hassan Abedi). Updated documentation.

version 1.10 (2001/04/30) Changed the license to GNU General Public License. Workspace for MS Visual C++ 6.0. New makefiles using implicit rules. New class `RobotMotor` that includes motors parameters (rotor inertia, gear ratio and friction coefficients). Updated documentation.

version 1.09 (98/09/27) Makefile for MS Visual C++ 6.0.

version 1.08 (98/06/1) Changes to `robot.cpp` and `robot.h` to avoid the warning messages:

```
initialization of non-const reference '*' from rvalue '*'
```

Fixed function `ieulzxz` in `homogen.cpp` thanks to Kilian Pohl.

version 1.07 (98/05/12) The bench.cpp program is more portable. Simpler makefile for Borland C++. New targets in makefiles (clean and veryclean). Removed the CVS Log tags from the sources. Compiler option -O now works under gcc 2.7.2 thanks to the new newmat.h provided by Robert Davies.

version 1.06 (97/11/21) The function inv_kin modified to use the Jacobian by default in the iterative procedure ($\approx 1.8\times$ faster). Updated documentation.

version 1.05 (97/11/17) Added make file for GNU G++ under Windows 95/NT using Cygnus GNU-Win32 compiler. Added optimization flags under GNU G++. Updated documentation.

version 1.04 (97/11/14) Added make file for GNU G++ and graphic support through gnuplot (2d plots). Updated documentation.

version 1.03 (97/11/01) Added adaptive step size integration. Changes to the documentation.

version 1.02 (97/10/21) Upgraded the matrix library from NEWMAT08A to NEWMAT09. New directory structure : newmat08 is replaced by newmat. Conditional compilation of delete [] for pre 2.1 C++ compilers has been removed since NEWMAT09 no longer supports these compilers. Minor changes to the documentation.

version 1.01 (97/01/17) Conditional compilation of delete [] for pre 2.1 C++ compilers. Changes to the documentation.

version 1.0 (96/12/15) First public release of the package.

1.6 Files in the distribution

readme	txt	readme file
makefile	gcc	make file for GNU G++ Linux
makefile	gccOSX	make file for GNU G++ MAC OS X
makefile	gw32	make file for Cygwin (Win32)
makefile	bc5	make file for Borland C++ 4.5, 5.x (Win32)
makefile	vcpp	make file for Visual C++ 5.0 and 6.0(Win32)
CMakeLists	txt	Confguration file for CMake
roboop	bkl	Configuration file for Bakefile
roboop	dsw	workspace for Visual C++ 6.0 (Win32)
bench	dsp	project file used by `roboop.dsw`
demo	dsp	project file used by `roboop.dsw`
demo_2dof_pd	dsp	project file used by `roboop.dsw`
newmat	dsp	project file used by `roboop.dsw`
roboop	dsp	project file used by `roboop.dsw`
rtest	dsp	project file used by `roboop.dsw`
roboop	sln	solution for Visual C++ 7.0 (Win32)
bench	vcproj	project file used by `roboop.sln`
demo	vcproj	project file used by `roboop.sln`
demo_2dof_pd	vcproj	project file used by `roboop.sln`
newmat	vcproj	project file used by `roboop.sln`
roboop	vcproj	project file used by `roboop.sln`
rtest	vcproj	project file used by `roboop.sln`
roboop	wpj	project file for OpenWatcom C++ 1.2 IDE (Win32)
bench	tgt	target file used by `roboop.wpj`
demo	tgt	target file used by `roboop.wpj`
demo_2dof_pd	tgt	target file used by `roboop.wpj`
newmat	tgt	target file used by `roboop.wpj`
roboop	tgt	target file used by `roboop.wpj`
rtest	tgt	target file used by `roboop.wpj`
demo	txt	output of the demo program
newmat		directory of the matrix library NEWMAT11 see the file `nm11.htm`
docs		documentation directory
gnugpl	txt	GNU General Public License
gnulgpl	txt	GNU Lesser General Public License
robot	ps	documentation in `postscript` format
robot	pdf	documentation in PDF format
doxy		Doxygen documentation directory
roboop_doxygen		Doxygen configuration file

source		the ROBOOP program source directory
CMakeLists	txt	Confguration file for CMake
robot	h	header file
clik	h	header file for CLIK
config	h	header file for configuration class
controller	h	header file for controllers
control_select	h	header file for Control_Select class
dynamics_sim	h	header file for Dynamics class
gnugraph	h	header file for the graphics
quaternion	h	header file for the quaternions
stewart	h	header file for the Stewart classs
trajectory	h	header file for the splines
utils	h	header file utility functions
bench	cpp	benchmark program file
clik	cpp	closed loop inverse kinematics CLIK
comp_dq	cpp	simplified version of delta_t with no dqp and dqpp
comp_dqp	cpp	simplified version of delta_t with no dq and dqpp
config	cpp	configuration class members functions
controller	cpp	some controllers functions
control_select	cpp	controller selection functions
delta_t	cpp	compute torque variation w/r to dq, dqp and dqpp
demo	cpp	demo program file
demo_2dof_pd	cpp	demo program file
dynamics	cpp	dynamics functions
dynamics_sim	cpp	simulation dynamics functions
gnugraph	cpp	graphics functions
homogen	cpp	homogeneous transform functions
impedance	cpp	impedance controller
invkine	cpp	inverse kinematics functions
kinemat	cpp	kinematics functions
quaternion	cpp	quaternions functions
robot	cpp	constructors and other stuff
rtest	cpp	testing program file
test	txt	testing data file
sensitiv	cpp	partial derivatives of robot dynamics
stewart	cpp	implemantation of the Stewart classs
trajectory	cpp	translation and rotation splines
utils	cpp	miscellaneous

conf		configuration files directory
pd_2dof	conf	PD controller parameters for the 2 dof robot
puma560_dh	conf	PUMA robot parameters standard D-H
puma560_mdh	conf	PUMA robot parameters modified D-H
q_2dod	dat	desired trajectory for the 2 dof robot
rhino560_dh	conf	RHINO robot parameters standard D-H
rhino560_mdh	conf	RHINO robot parameters modified D-H
rr_dh	conf	2 dof robot parameters standard D-H
stewart	conf	a Stewart platform parameters file

1.7 Doxygen documentation

Source code now has Doxygen compatible documentation. To obtain the documentation (under Linux) simply run doxygen roboop_doxygen in the doxy directory. It will creates `html` and `latex` directories.

The main html page can be accessed using the `index.html` file. To obtain the latex documentation simply run the `Makefile` in the `latex` directory.

Chapter 2

Reference manual

This package uses data types defined by the NEWMAT11 matrix library:

- `Real` : the type for floating point values. It can be either a `float` or a `double` as defined in the header file `include.h` in the `newmat` directory.
- `Matrix` : the type for matrices as defined in the NEWMAT11 documentation.
- `ColumnVector` : a type for column vectors derived from `Matrix`.
- `ReturnMatrix` : the type used by functions for returning any type of matrix (`Matrix, ColumnVector, RowVector,` etc).

The file `demo.cpp` presents examples for the use of some functions in the package. The time required to compute some functions for a 6 dof robot can be obtained with the file `bench.cpp`.

2.1 3D homogeneous transforms

In this section, functions dealing with 4×4 homogeneous transform matrices are described.

eulzxz

Syntax

```
ReturnMatrix eulzxz(const ColumnVector & a);
```

Description

Given a column vector `a`

$$\begin{bmatrix} \gamma_1 \\ \beta \\ \gamma_2 \end{bmatrix} \tag{2.1}$$

this function returns the homogeneous transform matrix given by

$$\boldsymbol{Rot}(z, \gamma_1)\boldsymbol{Rot}(x, \beta)\boldsymbol{Rot}(z, \gamma_2) \tag{2.2}$$

Note: the column vector `a` must have a length of at least 3. Only the first 3 elements are used.

Return Value

`Matrix`

ieulzxz

Syntax

```
ReturnMatrix ieulzxz(const Matrix & R);
```

Description

Given a homogeneous transform matrix `R`, this function returns a column vector

$$\begin{bmatrix} \gamma_1 \\ \beta \\ \gamma_2 \end{bmatrix} \tag{2.3}$$

such that the 3×3 rotation bloc of the matrix

$$\boldsymbol{Rot}(z, \gamma_1)\boldsymbol{Rot}(x, \beta)\boldsymbol{Rot}(z, \gamma_2) \tag{2.4}$$

is equal to the 3×3 rotation bloc of the matrix `R`.

Return Value

`ColumnVector`.

irotk

Syntax

```
ReturnMatrix irotk(const Matrix & R);
```

Description

Given a homogeneous transform matrix `R`, this function returns a column vector

$$\begin{bmatrix} \boldsymbol{k} \\ \theta \end{bmatrix} \tag{2.5}$$

with $\boldsymbol{k}$ a unit vector such that the 3×3 rotation bloc of the matrix

$$\boldsymbol{Rot}(\boldsymbol{k}, \theta) \tag{2.6}$$

is equal to the 3×3 rotation bloc of the matrix `R`.

Return Value

`ColumnVector`.

irpy

Syntax

```
ReturnMatrix irpy(const Matrix & R);
```

Description

Given a homogeneous transform matrix `R`, this function returns a column vector

$$\begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} \tag{2.7}$$

such that the 3×3 rotation bloc of the matrix

$$\boldsymbol{Rot}(z,\gamma)\boldsymbol{Rot}(y,\beta)\boldsymbol{Rot}(x,\alpha) \tag{2.8}$$

is equal to the 3×3 rotation bloc of the matrix `R`.

Return Value

`ColumnVector`.

rotd

Syntax

```
ReturnMatrix rotd(const Real theta,
                  const ColumnVector & k1,
                  const ColumnVector & k2);
```

Description

This function returns the matrix of a rotation of an angle `theta` around the oriented line segment defined by the points `k1` and `k2`.
Note: the column vectors `k1` and `k2` must have a length of at least 3. Only the first 3 elements are used.

Return Value

`Matrix`

rotk

Syntax

```
ReturnMatrix rotk(const Real theta,
                  const ColumnVector & k);
```

Description

This function returns the matrix of a rotation of an angle `theta` around the vector `k`.

$$\boldsymbol{Rot}(\boldsymbol{k}, \theta) \tag{2.9}$$

Note: the column vector `k` must have a length of at least 3. Only the first 3 elements are used.

Return Value

`Matrix`

rpy

Syntax

```
ReturnMatrix rpy(const ColumnVector & a);
```

Description

Given a column vector `a`

$$\begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} \tag{2.10}$$

this function returns the homogeneous transform matrix given by

$$\boldsymbol{Rot}(z, \gamma)\boldsymbol{Rot}(y, \beta)\boldsymbol{Rot}(x, \alpha) \tag{2.11}$$

Note: the column vector `a` must have a length of at least 3. Only the first 3 elements are used.

Return Value

`Matrix`

rotx, roty, rotz

Syntax

```
ReturnMatrix rotx(const Real alpha);
ReturnMatrix roty(const Real beta);
ReturnMatrix rotz(const Real gamma);
```

Description

These functions return the elementary rotation matrices:

$$\boldsymbol{Rot}(x,\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.12}$$

$$\boldsymbol{Rot}(y,\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.13}$$

$$\boldsymbol{Rot}(z,\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 & 0 \\ \sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.14}$$

Return Value

`Matrix`

trans

Syntax

```
ReturnMatrix trans(const ColumnVector & a);
```

Description

Given a column vector `a`, this function returns the following matrix:

$$\boldsymbol{Trans}(a) \quad = \quad \begin{bmatrix} 1 & 0 & 0 & a_1 \\ 0 & 1 & 0 & a_2 \\ 0 & 0 & 1 & a_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.15}$$

Note: the column vector `a` must have a length of at least 3. Only the first 3 elements are used.

Return Value

`Matrix`

2.2 The Quaternion class

The `Quaternion` class deals with quaternions. Unit quaternions are used to represent rotations. It is composed of two elements: a scalar s (`Real s_`) and a vector $\boldsymbol{v}$ (`ColumnVector v_`) representing a quaternion (see[1]).

$$q = w + xi + yj + zk \quad (2.16)$$

$$= (s, v) \quad (2.17)$$

An object of this class can be initialize with no parameter ($s = 1$ and $\boldsymbol{v} = \boldsymbol{0}$), from an other unit quaternion, from an angle of rotation around a unit vector, from a rotation matrix, from a quaternion object or from the four components of a quaternion. The constructors does not guarantee that quaternions will be unit.

constructors

Syntax

```
Quaternion();
Quaternion(const Quaternion & q);
Quaternion(const Real angle_in_rad, const ColumnVector & axis);
Quaternion(const Real s, const Real v1, const Real v2, const Real v3);
Quaternion(const Matrix & R);
Quaternion & operator=(const Quaternion & q);
```

Description

`Quaternion` object constructors, copy constructor and equal operator.

Return Value

None

operators

Syntax

```
Quaternion   operator+(const Quaternion & q)const;
Quaternion   operator-(const Quaternion & q)const;
Quaternion   operator*(const Quaternion & q)const;
Quaternion   operator*(const ColumnVector & vec)const;
Quaternion   operator*(constReal c)const;
Quaternion   operator/(const Quaternion & q)const;
Quaternion   operator/(constReal c)const;
```

Description

The operators $+$, $-$, $*$ and / for quaternion are implemented. The operators $*$ and / will generate unit quaternions only if the quaternions involve are unity.

Return Value

`Quaternion`

conjugate and inverse

Syntax

```
Quaternion   conjugate()const;
Quaternion   i()const;
```

Description

Compute the conjugate of the quaternion (or the inverse if it's a unit quaternion). The conjugate is defined as

$$q^* = w - xi - yj - zk \tag{2.18}$$

$$= (s, -v) \tag{2.19}$$

Return Value

`Quaternion`

exponential and logarithm

Syntax

```
Quaternion   exp()const;
Quaternion   Log()const;
Quaternion   power(const Real t)const;
```

Description

A unit quaternion can be represented by $q = cos(\theta) + usin(\theta)$. Euler's identity for complex numbers generalizes to quaternions $exp(u\theta) = cos(\theta) + usin(\theta)$, where $exp(x)$ is replace by $exp(u\theta)$ and uu is replace by -1. With this identity we obtain the exponential of the quaternion $q = (0, \theta v)$, where q is not necessary a unit quaternion. It is then possible to define the logarithm and the power of a unit quaternion [2].

$$Log(q) = Log(\cos(\theta) + u\sin(\theta)) = Log(exp(u\theta)) = u\theta \tag{2.20}$$

$$q^t = \cos(t\theta) + u\sin(t\theta) \tag{2.21}$$

$Log(q)$ is not necessary a unit quaternion even if q is a unit quaternion.

Return Value

`Quaternion` for `exp, Log`

dot_product

Syntax

```
Real dot_prod(const Quaternion & q)const;
```

Description

Compute the dot product of quaternions.

Return Value

`Real`

quaternion time derivative

Syntax

```
Quaternion   dot(const ColumnVector & w, const short sign)const;
ReturnMatrix E(const short sign)const;
```

Description

The quaternion time derivative is obtain from the quaternion propagation law [2].

$$\dot{s} = -\frac{1}{2}v^T w \tag{2.22}$$

$$\dot{v} = \frac{1}{2}E(s,v)w \tag{2.23}$$

where

$$\begin{array}{ll} E = \eta I - S(\epsilon) & \text{in base frame} \\ E = \eta I + S(\epsilon) & \text{in body frame} \end{array} \tag{2.24}$$

The choice of reference system (base or body) for $\boldsymbol{w}$ is assign by ***sign***. A value of 1 is for base frame while -1 is for body frame.

Return Value

`Quaternion` for `dot`
`Matrix` for `E`

unit and norm

Syntax

```
Quaternion & unit();
Real norm()const;
```

Description

`unit()` makes the quaternion a unit quaternion, `norm()` computes and returns the norm of the quaternion. `norm_sqr()` computes and returns the square norm of the quaternion.

Return Value

`Quaternion` for `unit()`
`Real` for `norm()` and `norm_sqr()`

s and v

Syntax

```
Real s()const;
void set_s(const Real s);
ReturnMatrix v()const;
void set_v(const ColumnVector & v);
```

Description

The functions `s()` and `v()` returns one of the components of a quaternion (s or $\boldsymbol{v}$), while `set_s()` and `set_v()` can assign a value to one of the components.

Return Value

None for `set_s()` and `set_v()`
`Real` for `s()`
`Matrix` for `v()`

Rotation matrices

Syntax

```
ReturnMatrix R() const;
ReturnMatrix T() const;
```

Description

Returns a rotation matrix from the quaternion (`R()` returns a 3×3 matrix and `T()` returns a 4×4 matrix).

Return Value

`Matrix`

Omega, ω

Syntax

```
ReturnMatrix Omega(const Quaternion & q, const Quaternion & q_dot);
```

Description

Omega is not a member function of the class `Quaternion`. The function returned the angular velocity obtain from a quaternion and it's time derivative. Like the member function `dot`, it use the quaternions propagation law [2].

Return Value

`ColumnVector`

Slerp

Syntax

```
Quaternion Slerp(const Quaternion & q0, const Quaternion & q1,
                 const Real t);
```

Description

Slerp stands for Spherical Linear Interpolation. Slerp is not a member function of the class Quaternion. The quaternions q_0 and q_1 needs to be unit quaternions. It returns a unit quaternion. As the parameter t uniformly varies between 0 and 1, the values $q(t)$ are required to uniformly vary along the circular arc from q_0 to q_1.

It is customary to choose the sign G on q_1 so that $q_0 \cdot Gq_1 \geq 0$ (the angle between q_0 and Gq_1 is acute). This choice avoids extra spinning caused by the interpolated rotations [2]. For unit quaternions Slerp is defined as

$$q = \begin{cases} q_0(q_0^{-1}q_1)^t & \text{if } q_0 \cdot q_1 \geq 0 \\ q_0(q_0^{-1}(-q_1))^t & \text{otherwise} \end{cases} \tag{2.25}$$

Return Value

Quaternion

Slerp_prime

Syntax

```
Quaternion Slerp_prime(const Quaternion & q0, const Quaternion & q1,
                       const Real t);
```

Description

Slerp_prime represent the Slerp derivative. Slerp_prime is not a member function of the class Quaternion. The quaternions q_0 and q_1 needs to be unit quaternions. It does not necessary returns a unit quaternion.

It is customary to choose the sign G on q_1 so that $q_0 \cdot Gq_1 \geq 0$ (the angle between q_0 and Gq_1 is acute). This choice avoids extra spinning caused by the interpolated rotations [2]. For unit quaternions Slerp is defined as

$$q = \begin{cases} Slerp(q_0, q_1, t) Log(q_0^{-1} q_1) & \text{if } q_0 \cdot q_1 \geq 0 \\ Slerp(q_0, q_1, t) Log(q_0^{-1}(-q_1)) & \text{otherwise} \end{cases} \tag{2.26}$$

Return Value

Quaternion

Squad

Syntax

```
Quaternion Squad(const Quaternion & p, const Quaternion & a,
                 const Quaternion & b, const Quaternion & r,
                 const Real t);
```

Description

Squad stands for Spherical Cubic Interpolation. Squad is not a member function of the class Quaternion. The quaternions p, a, b and r needs to be unit quaternions. It returns a unit quaternion.

Squad uses an iterative of three slerps. Suppose four quaternions, p, a, b and r as the ordered vertices of quadrilateral. Interpolate c along p to q using slerp and d along a to b also using slerp. Now interpolate q along c to d [2]. Squad is defined as

$$q = Slerp(Slerp(p, r, t), Slerp(a, b, t), 2t(1 - t)); \tag{2.27}$$

Return Value

Quaternion

Squad_prime

Syntax

```
Quaternion Squad_prime(const Quaternion & p, const Quaternion & a,
                       const Quaternion & b, const Quaternion & q,
                       const Real t);
```

Description

Squad_prime represent the Squad derivative. Squad_prime is not a member function of the class Quaternion.

Return Value

Quaternion

2.3 The Robot and mRobot classes

The `Robot` and `mRobot` classes are composed of the following data elements:

- the number of degree of freedom n (`int dof`);
- the gravity acceleration vector ($-\boldsymbol{g}$) expressed in the base frame (`ColumnVector gravity`);
- one array of dimension n of `Link` object elements (`Link *links`);

and the member functions providing the different algorithms implementation (see tables 2.2–2.17).

The `Link` class (see table 2.1) encapsulates all the data and functionality required to characterize a single "link" as it is defined by Denavit and Hartenberg (standard notation [3], or modified notation [4]). It is initialized by providing the joint type (`int joint_type`: revolute=0, prismatic=1) and the parameters θ, d, a, α (`Real theta, d, a, alpha`) and a boolean value `Bool DH` (`true`=standard `false`=modified) It also contains the inertial parameters data: mass m (`Real m`), center of mass position vector $\boldsymbol{r}$ (`ColumnVector r`) and inertia tensor matrix $\boldsymbol{I}_c$ (`Matrix I`). In this case, $\boldsymbol{r}$ is given with respect to the link coordinate frame and $\boldsymbol{I}_c$ is with respect to a coordinate frame parallel to the link coordinate frame and located at the center of mass of m. The dynamic model takes into account the motors inertia, gear ratio and frictions. The values `Im` and `Gr` representing respectively the motors rotor inertia I_m and gear ratio G_r; `B` and `Cf` representing respectively the motors viscous B and Coulomb friction C_f coefficients:

$$\tau_f = B\dot{q} + C_f \mathrm{sign}(\dot{q})$$

On initialization, the constructor sets up the matrices $\boldsymbol{R}$ and $\boldsymbol{p}$ such that

$$\boldsymbol{R} = \begin{bmatrix} \cos\theta & -\cos\alpha\sin\theta & \sin\alpha\sin\theta \\ \sin\theta & \cos\alpha\cos\theta & -\sin\alpha\cos\theta \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix} \tag{2.28}$$

$$\boldsymbol{p} = \begin{bmatrix} a\cos\theta \\ a\sin\theta \\ d \end{bmatrix} \tag{2.29}$$

for the standard D-H notation and

$$\boldsymbol{R} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \cos\alpha\sin\theta & \cos\alpha\cos\theta & -\sin\alpha \\ \sin\alpha\sin\theta & \sin\alpha\cos\theta & \cos\alpha \end{bmatrix} \tag{2.30}$$

Table 2.1: The `Link` class data parameters

Kinematic		Inertial		Motor	
`int`	`joint_type`	`Real`	`m`	`Real`	`Im`
`Real`	`theta, d, a, alpha`	`ColumnVector`	`r`	`Real`	`Gr`
`Real`	`joint_offset`	`Matrix`	`I`	`Real`	`B`
`ColumnVector`	`p`			`Real`	`Cf`
`Matrix`	`R,`				
`Bool`	`DH`				
`Real`	`theta_min, theta_max`				
`Real`	`joint_offset`				

$$\boldsymbol{p} \;=\; \begin{bmatrix} a \\ -d\sin\alpha \\ d\cos\alpha \end{bmatrix} \tag{2.31}$$

for the modified D-H notation.

If the link corresponds to a revolute (prismatic) joint, then only θ (d) can be changed after the link definition. This is done through the member function `transform` which sets the new value of q (θ or d) and updates the matrices $\boldsymbol{R}$ and $\boldsymbol{p}$ which compose the link homogeneous transform:

$$\boldsymbol{T} \;=\; \begin{bmatrix} \boldsymbol{R} & \boldsymbol{p} \\ 0 & 1 \end{bmatrix} \tag{2.32}$$

Only the changing elements are computed since the data of an instance of a class is persistent throughout the scope of definition of the instance (see [5]). In standard notation, the elements (3,2) and (3,3) of $\boldsymbol{R}$ provide storage for $\cos\alpha$ and $\sin\alpha$ which are computed only once. In modified notation, the elements (3,3) and (2,3) of $\boldsymbol{R}$ provide storage for $\cos\alpha$ and $\sin\alpha$. So as to make the implementation faster, only the elements of $\boldsymbol{R}$ and $\boldsymbol{p}$ involving θ (d) are updated with a revolute (prismatic) joint.

2.3.1 `Robot` and `mRobot` object initialization

The `Robot` and `mRobot` classes provide a default constructor that creates a 1 dof robot. A $n_{dof} \times 19$ matrix containing the kinematic and inertial parameters (as for the `Robot` class) can be supplied upon initialization. A

$n_{dof} \times 19$ matrix containing the kinematic and inertial parameters (as for the `Robot` class) can be supplied along with a $n_{dof} \times 4$ matrix providing the motors inertia, gear ratio and friction coefficients. A $n_{dof} \times 23$ matrix (kinematic, inertial and motor parameters) can also be used. The structure of the initialization matrix is:

Column	Variable	Description
1	σ	joint type (revolute=0, prismatic=1)
2	θ	Denavit-Hartenberg parameter
3	d	Denavit-Hartenberg parameter
4	a	Denavit-Hartenberg parameter
5	α	Denavit-Hartenberg parameter
6	θ_{min}	minimum value of joint variable
7	θ_{max}	maximum value of joint variable
8	θ_{off}	joint offset
9	m	mass of the link
10	c_x	center of mass along axis x
11	c_y	center of mass along axis y
12	c_z	center of mass along axis z
13	I_{xx}	element xx of the inertia tensor matrix
14	I_{xy}	element xy of the inertia tensor matrix
15	I_{xz}	element xz of the inertia tensor matrix
16	I_{yy}	element yy of the inertia tensor matrix
17	I_{yz}	element yz of the inertia tensor matrix
18	I_{zz}	element zz of the inertia tensor matrix
19	I_m	motor rotor inertia
20	Gr	motor gear ratio
21	B	motor viscous friction coefficient
22	C_f	motor Coulomb friction coefficient
23	`immobile`	flag for the kinematics and inverse kinematics (if true joint is locked, if false joint is free)

constructors

Syntax

Standard notation:

```
Robot(const int ndof=1);
Robot(const Matrix & initrobot);
Robot(const Matrix & initrobot, const Matrix & initmotor);
Robot(const Robot & x);
Robot(const string & filename, const string & robotName);
```

Modified notation:

```
mRobot(const int ndof=1);
mRobot(const Matrix & initrobot_motor);
mRobot(const Matrix & initrobot, const Matrix & initmotor);
mRobot(const mRobot & x);
mRobot(const string & filename, const string & robotName);
```

Description

`Robot` and `mRobot` object constructors, copy constructor and equal operator.

Return Value

None

get_q, get_qp, get_qpp

Syntax

```
ReturnMatrix get_q(void);
Real get_q(const int i);
ReturnMatrix get_qp(void);
Real get_qp(const int i);
ReturnMatrix get_qp(void);
Real get_qp(const int i);
```

Description

These functions return a column vector containing the joint variables (`get_q`), velocities (`get_qp`) or accelerations (`get_qpp`) when called with no argument. It returns the scalar value for the i^{th} joint variable when called with an integer argument.

Return Value

`ColumnVector` or `Real`

set_q, set_qp, set_qpp

Syntax

```
void set_q(const ColumnVector & q);
void set_q(const Matrix & q);
void set_q(const Real q, const int i);
void set_qp(const ColumnVector & qp);
void set_qp(const Matrix & qp);
void set_qp(const Real qp, const int i);
void set_qpp(const ColumnVector & qpp);
void set_qpp(const Matrix & qpp);
void set_qpp(const Real qpp, const int i);
```

Description

These functions set the joint variables (velocities or accelerations) or the i^{th} joint variable (velocity or acceleration) to `q` (`qp` or `qpp`).

Return Value

None

2.3.2 Kinematics

The forward kinematic model defines the relation:

$$ {}^{0}\boldsymbol{T}_n = \boldsymbol{G}(\boldsymbol{q}) \tag{2.33} $$

where ${}^{0}\boldsymbol{T}_n$ is the homogeneous transform representing the position and orientation of the manipulator tool (frame n) in the base frame 0. The inverse kinematic model is defined by

$$ \boldsymbol{q} = \boldsymbol{G}^{-1}({}^{0}\boldsymbol{T}_n) \tag{2.34} $$

In general, this equation allows multiple solutions.

inv_kin

Syntax

```
ReturnMatrix inv_kin(const Matrix & Tobj, const int mj = 0);
ReturnMatrix inv_kin(const Matrix & Tobj, const int mj,
                     const int endlink, bool & converge);
```

Description

The inverse kinematic model is computed using a Newton-Raphson technique. If `mj == 0`, it is based on the following [6]:

$$
\begin{aligned}
{}^0\boldsymbol{T}_n(\boldsymbol{q}^*) &= {}^0\boldsymbol{T}_n(\boldsymbol{q}+\delta\boldsymbol{q}) \approx {}^0\boldsymbol{T}_n(\boldsymbol{q})\delta\boldsymbol{T}(\delta\boldsymbol{q}) = \boldsymbol{T}_{obj} && (2.35)\\
\delta\boldsymbol{T}(\delta\boldsymbol{q}) &= ({}^0\boldsymbol{T}_n(\boldsymbol{q}))^{-1}\boldsymbol{T}_{obj} = \boldsymbol{I} + \boldsymbol{\Delta} && (2.36)\\
\boldsymbol{\Delta} &= \begin{bmatrix} 0 & -\delta_z & \delta_y & d_x \\ \delta_z & 0 & -\delta_x & d_y \\ -\delta_y & \delta_x & 0 & d_z \\ 0 & 0 & 0 & 0 \end{bmatrix} && (2.37)\\
{}^n\delta\boldsymbol{\chi} &= \begin{bmatrix} d_x & d_y & d_z & \delta_x & \delta_y & \delta_z \end{bmatrix}^T && (2.38)\\
{}^n\delta\boldsymbol{\chi} &\approx {}^n\boldsymbol{J}(\boldsymbol{q})\delta\boldsymbol{q} && (2.39)
\end{aligned}
$$

If `mj == 1`, it is based on the following Taylor expansion [6, 7]:

$$
{}^0\boldsymbol{T}_n(\boldsymbol{q}^*) = {}^0\boldsymbol{T}_n(\boldsymbol{q}+\delta\boldsymbol{q}) \approx {}^0\boldsymbol{T}_n(\boldsymbol{q}) + \sum_{i=1}^{n} \frac{\partial {}^0\boldsymbol{T}_n}{\partial q_i}\delta q_i \qquad (2.40)
$$

The function `dTdqi` computes these partial derivatives.

Given the desired position represented by the homogeneous transform `Tobj`, this function return the column vector of joint variables that is corresponding to this position. On return, the value `converge` is true if the procedure has converge to values that give the correct position and false otherwise.

Note: `mj == 0` is faster ($\approx 1.8\times$) than `mj == 1`. Also, `mj == 1` might converge when `mj == 0` does not.

Return Value

`ColumnVector`

inv_kin_rhino

Syntax

```
ReturnMatrix inv_kin_rhino(const Matrix & Tobj,
                           bool & converge)
```

Description

This function performs the Rhino robot inverse kinematics.

Return Value

`ColumnVector`

inv_kin_puma

Syntax

```
ReturnMatrix inv_kin_puma(const Matrix & Tobj,
                          bool & converge)
```

Description

This function performs the Puma robot inverse kinematics.

Return Value

`ColumnVector`

jacobian

Syntax

```
ReturnMatrix jacobian(const int ref=0);
ReturnMatrix jacobian(const int endlink, const int ref)const;
```

Description

The manipulator Jacobian defines the relation between the velocities in joint space $\dot{\boldsymbol{q}}$ and in the Cartesian space $\dot{\boldsymbol{\chi}}$ expressed in frame i:

$$ {}^{i}\dot{\boldsymbol{\chi}} = {}^{i}\boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} \tag{2.41} $$

or the relation between small variations in joint space $\delta\boldsymbol{q}$ and small displacements in the Cartesian space $\delta\boldsymbol{\chi}$:

$$ {}^{i}\delta\boldsymbol{\chi} \approx {}^{i}\boldsymbol{J}(\boldsymbol{q})\delta\boldsymbol{q} \tag{2.42} $$

The manipulation Jacobian expressed in the base frame is given by (see [8])

$$ {}^{0}\boldsymbol{J}(\boldsymbol{q}) = \left[\begin{array}{cccc} {}^{0}\boldsymbol{J}_1(\boldsymbol{q}) & {}^{0}\boldsymbol{J}_2(\boldsymbol{q}) & \cdots & {}^{0}\boldsymbol{J}_n(\boldsymbol{q}) \end{array} \right] \tag{2.43} $$

with

$$ {}^{0}\boldsymbol{J}_i(\boldsymbol{q}) = \left[\begin{array}{c} \boldsymbol{z}_{i-1} \times {}^{i-1}\boldsymbol{p}_n \\ \boldsymbol{z}_{i-1} \end{array} \right] \text{ for a revolute joint} \tag{2.44} $$

$$ {}^{0}\boldsymbol{J}_i(\boldsymbol{q}) = \left[\begin{array}{c} \boldsymbol{z}_{i-1} \\ 0 \end{array} \right] \text{ for a prismatic joint} \tag{2.45} $$

where $\boldsymbol{z}_{i-1}$ and ${}^{i-1}\boldsymbol{p}_n$ are expressed in the base frame and $\times$ is the vector cross product. Expressed in the i^{th} frame, the Jacobian is given by

$$ {}^{i}\boldsymbol{J}(\boldsymbol{q}) = \left[\begin{array}{cc} ({}^{0}\boldsymbol{R}_i)^T & 0 \\ 0 & ({}^{0}\boldsymbol{R}_i)^T \end{array} \right] {}^{0}\boldsymbol{J}(\boldsymbol{q}) \tag{2.46} $$

This function returns ${}^{i}\boldsymbol{J}(\boldsymbol{q})$ ($i = 0$ when not specified) for the `endlink` (last link when not specified).

Return Value

`Matrix`

jacobian_dot

Syntax

```
ReturnMatrix jacobian_dot(const int ref=0);
```

Description

The manipulator Jacobian time derivative can be used to compute the end effector acceleration due to joints velocities [9]:

$$^{i}\ddot{\boldsymbol{x}} \;=\; {}^{i}\dot{\boldsymbol{J}}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} \tag{2.47}$$

The Jacobian time derivative expressed in the base frame is given by [9]

$$^{0}\dot{\boldsymbol{J}}(\boldsymbol{q},\dot{\boldsymbol{q}}) \;=\; \left[\begin{array}{cccc} {}^{0}\dot{\boldsymbol{J}}_1(\boldsymbol{q},\dot{\boldsymbol{q}}) & {}^{0}\dot{\boldsymbol{J}}_2(q,\dot{\boldsymbol{q}}) & \cdots & {}^{0}\dot{\boldsymbol{J}}_n(\boldsymbol{q},\dot{\boldsymbol{q}}) \end{array}\right] \tag{2.48}$$

with

$$^{0}\dot{\boldsymbol{J}}_i(\boldsymbol{q},\dot{\boldsymbol{q}}) \;=\; \left[\begin{array}{c} \boldsymbol{\omega}_{i-1}\times\boldsymbol{z}_i \\ \boldsymbol{\omega}_{i-1}\times^{i-1}\boldsymbol{p}_n + \boldsymbol{z}_i\times^{i-1}\dot{\boldsymbol{p}}_n \end{array}\right] \text{ for a revolute joint} \tag{2.49}$$

$$^{0}\dot{\boldsymbol{J}}_i(\boldsymbol{q},\dot{\boldsymbol{q}}) \;=\; \left[\begin{array}{c} 0 \\ 0 \end{array}\right] \text{ for a prismatic joint} \tag{2.50}$$

where $\boldsymbol{z}_i$ and $^{i-1}\boldsymbol{p}_n$ are expressed in the base frame and $\times$ is the vector cross product. Expressed in the i^{th} frame, the Jacobian time derivative is given by

$$^{i}\dot{\boldsymbol{J}}(\boldsymbol{q},\dot{\boldsymbol{q}}) = \left[\begin{array}{cc} ({}^{0}\boldsymbol{R}_i)^T & 0 \\ 0 & ({}^{0}\boldsymbol{R}_i)^T \end{array}\right] {}^{0}\dot{\boldsymbol{J}}(\boldsymbol{q},\dot{\boldsymbol{q}}) \tag{2.51}$$

This function returns $^{i}\dot{\boldsymbol{J}}(\boldsymbol{q},\dot{\boldsymbol{q}})$(i=0 when not specified).

Return Value

`Matrix`

jacobian_DLS_inv

Syntax

```
ReturnMatrix jacobian_DLS_inv(const Real eps, const Real lambda_max,
                              const int ref=0);
```

Description

This function returns the inverse Jacobian Matrix for 6 dof manipulator based on the Damped Least-Squares scheme [10]. Using the singular value decomposition, the Jacobian matrix is

$$J = \sum_{i=1}^{6} \sigma_i u_i v_i^T \tag{2.52}$$

where v_i and u_i are the input and output vectors, and σ_i are the singular values ordered so that $\sigma_i \geq \sigma_2 \geq \cdots \sigma_r \geq 0$, with r being the rank of J. Based on the Damped Least-Squares the inverse Jacobian can be written as

$$J^{-1} = \sum_{i=1}^{6} \frac{\sigma_i}{\sigma_i^2 + \lambda^2} v_i u_i^T \tag{2.53}$$

where λ is the damping factor. A singular region can be selected on the basis of the smallest singular value of J. Outside the region the exact solution is returned, while inside the region a configuration-varying damping factor is introduced to obtain the desired approximate solution. This region is defined as

$$\lambda^2 = \begin{cases} 0 & \text{if } \sigma_6 \geq \epsilon \\ \left(1 - (\frac{\sigma_6}{\epsilon})^2\right)\lambda_{max}^2 & \text{otherwise} \end{cases} \tag{2.54}$$

Return Value

`Matrix`

kine

Syntax

```
void kine(Matrix & Rot, ColumnVector & pos);
void kine(Matrix & Rot, ColumnVector & pos, const int j);
ReturnMatrix kine(void);
ReturnMatrix kine(const int j);
```

Description

The forward kinematic model is provided by implementing the following recursion:

$$
\begin{aligned}
{}^0\boldsymbol{R}_i &= {}^0\boldsymbol{R}_{i-1}\,{}^{i-1}\boldsymbol{R}_i && (2.55)\\
{}^0\boldsymbol{p}_i &= {}^0\boldsymbol{p}_{i-1} + {}^0\boldsymbol{R}_{i-1}\boldsymbol{p}_i && (2.56)
\end{aligned}
$$

where

$$
{}^0\boldsymbol{T}_i = \begin{bmatrix} {}^0\boldsymbol{R}_i & {}^0\boldsymbol{p}_i \\ 0 & 1 \end{bmatrix} \tag{2.57}
$$

The overloaded function `kine` can return the orientation and position or the equivalent homogeneous transform for the last (if not supplied) or the i^{th} link. For example:

```
Robot myrobot(init_matrix);
Matrix Thomo, R;
ColumnVector p;
/* forward kinematics up to the last link */
Thomo = myrobot.kine();
/* forward kinematics up to the 2nd link  */
Thomo = myrobot.kine(2);
/* forward kinematics up to the last link, outputs R and p */
myrobot.kine(R,p);
/* forward kinematics up to the 2nd link, outputs R and p  */
myrobot.kine(R,p,2);
```

are valid calls to the function `kine`.

Return Value

`Matrix` or None (in this case `Rot` and `pos` are modified on output)

kine_pd

Syntax

```
ReturnMatrix kine_pd(const int ref=0);
void kine_pd(Matrix & Rot, ColumnVector & pos,
             ColumnVector & pos_dot, const int ref=0);
```

Description

The forward kinematic model is provided by implementing the following recursion (similar to `kine`):

$$
\begin{aligned}
{}^{0}\boldsymbol{R}_i &= {}^{0}\boldsymbol{R}_{i-1}\,{}^{i-1}\boldsymbol{R}_i && (2.58)\\
{}^{0}\boldsymbol{p}_i &= {}^{0}\boldsymbol{p}_{i-1} + {}^{0}\boldsymbol{R}_{i-1}\boldsymbol{p}_i && (2.59)
\end{aligned}
$$

$$
\begin{array}{lc}
{}^{0}\dot{\boldsymbol{p}}_i = {}^{0}\dot{\boldsymbol{p}}_{i-1} + {}^{0}\boldsymbol{R}_i\boldsymbol{\omega}_i \times {}^{0}\boldsymbol{R}_{i-1}\boldsymbol{p}_i & \text{DH notation} \\
{}^{0}\dot{\boldsymbol{p}}_i = {}^{0}\dot{\boldsymbol{p}}_{i-1} + {}^{0}\boldsymbol{R}_{i-1}(\boldsymbol{\omega}_{i-1} \times \boldsymbol{p}_i) & \text{modified DH notation}
\end{array} \qquad (2.60)
$$

where

$$
{}^{0}\boldsymbol{T}_i = \begin{bmatrix} {}^{0}\boldsymbol{R}_i & {}^{0}\boldsymbol{p}_i \\ 0 & 1 \end{bmatrix} \qquad (2.61)
$$

Return Value

`Matrix` or None (in this case `Rot`, `pos` `pos_dot` are modified on output)

dTdqi

Syntax

```
void dTdqi(Matrix & dRot, ColumnVector & dp, const int i);
ReturnMatrix dTdqi(const int i);
```

Description

This function computes the partial derivatives:

$$\frac{\partial^0 \boldsymbol{T}_n}{\partial q_i} = {}^0\boldsymbol{T}_{i-1}\boldsymbol{Q}_i\,{}^{i-1}\boldsymbol{T}_n \tag{2.62}$$

in standard notation and

$$\frac{\partial^0 \boldsymbol{T}_n}{\partial q_i} = {}^0\boldsymbol{T}_{i}\boldsymbol{Q}_i\,{}^{i}\boldsymbol{T}_n \tag{2.63}$$

in modified notation, with

$$\boldsymbol{Q}_i = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \text{ for a revolute joint} \tag{2.64}$$

$$\boldsymbol{Q}_i = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \text{ for a prismatic joint} \tag{2.65}$$

Return Value

`Matrix` or None (in this case `dRot` and `dp` are modified on output)

2.3.3 Dynamics

The robotics manipulator dynamic model is given by (see appendix A or [4])

$$\boldsymbol{\tau} = \boldsymbol{D}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}}) + \boldsymbol{G}(\boldsymbol{q}) \tag{2.66}$$

acceleration

Syntax

```
ReturnMatrix acceleration(const ColumnVector & q,
                          const ColumnVector & qp,
                          const ColumnVector & tau);

ReturnMatrix acceleration(const ColumnVector & q,
                          const ColumnVector & qp,
                          const ColumnVector & tau_cmd,
                          const ColumnVector & Fext,
                          const ColumnVector & Next)
```

Description

This function computes $\ddot{\boldsymbol{q}}$ from $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$ and $\boldsymbol{\tau}$ which is the forward dynamics problem. Walker and Orin [11] presented methods to compute the inverse dynamics. A simplified RNE version computing

$$\boldsymbol{\tau} = \boldsymbol{D}(\boldsymbol{q})\ddot{\boldsymbol{q}} \tag{2.67}$$

is implemented in the function `torque_novelocity`. By evaluating this equation n times, one can compute $\boldsymbol{D}(\boldsymbol{q})$ (the `inertia` function), use the full RNE to compute $\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}}) + \boldsymbol{G}(\boldsymbol{q})$ and then solve the equation :

$$\ddot{\boldsymbol{q}} = \boldsymbol{D}^{-1}(\boldsymbol{q})\left[\boldsymbol{\tau} - \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}}) - \boldsymbol{G}(\boldsymbol{q})\right] \tag{2.68}$$

Return Value

`ColumnVector`

inertia

Syntax

```
ReturnMatrix inertia(const ColumnVector & q);
```

Description

This function computes the robot inertia matrix $\boldsymbol{D(q)}$. A simplified RNE version computing

$$\boldsymbol{\tau} = \boldsymbol{D(q)}\ddot{\boldsymbol{q}} \tag{2.69}$$

is implemented in the function `torque_novelocity`. By evaluating this equation n times, one can compute $\boldsymbol{D(q)}$.

Return Value

`Matrix`

torque

Syntax

```
ReturnMatrix torque(const ColumnVector & q,
                    const ColumnVector & qp,
                    const ColumnVector & qpp);

ReturnMatrix torque(const ColumnVector & q,
                    const ColumnVector & qp,
                    const ColumnVector & qpp,
                    const ColumnVector & Fext,
                    const ColumnVector & Next);
```

Description

This function computes $\boldsymbol{\tau}$ from $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$ and $\ddot{\boldsymbol{q}}$ which is the inverse dynamics problem. The recursive Newton-Euler (RNE) formulation is one of the most computationally efficient algorithm [12, 13] used to solve this problem (see appendix A). The second form allows the inclusion the contribution of a load applied at the last link.

Return Value

`ColumnVector`

torque_novelocity

Syntax

```
ReturnMatrix torque_novelocity(const ColumnVector & q,
                               const ColumnVector & qpp);

ReturnMatrix torque_novelocity(const ColumnVector & q,
                               const ColumnVector & qpp,
                               const ColumnVector & Fext,
                               const ColumnVector & Next);
```

Description

This function computes $\boldsymbol{\tau}$ from $\boldsymbol{q}$ and $\ddot{\boldsymbol{q}}$ when $\dot{\boldsymbol{q}} = 0$ and gravity is set to zero.

Return Value

`ColumnVector`

G and C

Syntax

```
ReturnMatrix G();
ReturnMatrix C();
```

Description

The function G() computes $\boldsymbol{\tau}$ from the gravity effect, while C() computes $\boldsymbol{\tau}$ from the Coriolis and centrifugal effects.

Return Value

`ColumnVector` for G and C

2.3.4 Linearized dynamics

Murray and Neuman [13] have developed an efficient recursive linearized Newton-Euler formulation that can be used to compute (see appendix A)

$$\delta\boldsymbol{\tau} = \boldsymbol{D}(\boldsymbol{q})\delta\ddot{\boldsymbol{q}} + \boldsymbol{S_1}(\boldsymbol{q},\dot{\boldsymbol{q}})\delta\dot{\boldsymbol{q}} + \boldsymbol{S_2}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}})\delta\boldsymbol{q} \tag{2.70}$$

delta_torque

Syntax

```
void delta_torque(const ColumnVector & q,
                  const ColumnVector & qp,
                  const ColumnVector & qpp,
                  const ColumnVector & dq,
                  const ColumnVector & dqp,
                  const ColumnVector & dqpp,
                  ColumnVector & torque,
                  ColumnVector & dtorque);
```

Description

This function computes

$$\delta\boldsymbol{\tau} = \boldsymbol{D}(\boldsymbol{q})\delta\ddot{\boldsymbol{q}} + \boldsymbol{S_1}(\boldsymbol{q},\dot{\boldsymbol{q}})\delta\dot{\boldsymbol{q}} + \boldsymbol{S_2}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}})\delta\boldsymbol{q} \tag{2.71}$$

Return Value

None (`torque` and `dtorque` are modified on output)

dq_torque

Syntax

```
void dq_torque(const ColumnVector & q,
               const ColumnVector & qp,
               const ColumnVector & qpp,
               const ColumnVector & dq,
               ColumnVector & torque,
               ColumnVector & dtorque);
```

Description

This function computes

$$\boldsymbol{S_2}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}})\delta\boldsymbol{q} \tag{2.72}$$

Return Value

None (`torque` and `dtorque` are modified on output)

dqp_torque

Syntax

```
void dqp_torque(const ColumnVector & q,
                const ColumnVector & qp,
                const ColumnVector & dqp,
                ColumnVector & torque,
                ColumnVector & dtorque);
```

Description

This function computes

$$\boldsymbol{S_1}(\boldsymbol{q},\dot{\boldsymbol{q}})\delta\dot{\boldsymbol{q}} \tag{2.73}$$

Return Value

None (`torque` and `dtorque` are modified on output)

dtau_dq

Syntax

```
ReturnMatrix dtau_dq(const ColumnVector & q,
                     const ColumnVector & qp,
                     const ColumnVector & qpp);
```

Description

This function computes

$$\frac{\partial \boldsymbol{\tau}}{\partial \boldsymbol{q}} = \boldsymbol{S_2}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}) \tag{2.74}$$

Return Value

`Matrix`

dtau_dqp

Syntax

```
ReturnMatrix dtau_dqp(const ColumnVector & q,
                      const ColumnVector & qp);
```

Description

This function computes

$$\frac{\partial \boldsymbol{\tau}}{\partial \dot{\boldsymbol{q}}} = \boldsymbol{S_1}(\boldsymbol{q},\dot{\boldsymbol{q}}) \tag{2.75}$$

Return Value

`Matrix`

perturb_robot

Syntax

```
void perturb_robot(Robot_basic & robot, const double f = 0.1);
```

Description

This function, which is not a member of any class, modifies randomly the robot parameters. The parameter variation in percentage is described by `f`.

Return Value

None

2.4 The Spl_Cubic class

Spl_Cubic deals with parametric cubic splines [9].

Constructor

Syntax

```
Spl_cubic(){};
Spl_cubic(const Matrix & pts);
Spl_cubic(const Spl_cubic & x);
Spl_cubic & operator=(const Spl_cubic & x);
```

Description

Spl_Cubic object constructor, copy constructor and equal operator.

Return Value

None

s, ds and dds

Syntax

```
short interpolating(const Real t, ColumnVector & s);
short first_derivative(const Real t, ColumnVector & ds);
short second_derivative(const Real t, ColumnVector & dds);
```

Description

These functions interpolate the spline at time t to sets the quaternion s, ds and dds.

Return Value

Status, as a short int.

0 successful

NOT_IN_RANGE (regarding t)

BAD_DATA

2.5 The Spl_path class

Spl_path uses three instances of the class Spl_Cubic for path X, Y, Z interpolation.

Constructor

Syntax

```
Spl_path():Spl_cubic(){};
Spl_path(const string & filename);
Spl_path(const Matrix & x);
Spl_path(const Spl_path & x);
Spl_path & operator=(const Spl_path & x);
```

Description

Spl_path object constructor, copy constructor and equal operator.

Return Value

None

p, dp, ddp

Syntax

```
short p(const Real time, ColumnVector & p);
short p_pdot(const Real time, ColumnVector & p, ColumnVector & pdot);
short p_pdot_pddot(const Real time, ColumnVector & p, ColumnVector & dp,
                   ColumnVector & ddp);
```

Description

These functions interpolate the spline at time t to sets the quaternion p (position), dp (velocity) and ddp (acceleration).

Return Value

Status, as a short int.

0 successful

NOT_IN_RANGE (regarding t)

BAD_DATA

2.6 The Spl_Quaternion class

Spl_Quaternion deals with parametric quaternions cubic splines.

Constructor

Syntax

```
Spl_Quaternion(){}
Spl_Quaternion(const string & filename);
Spl_Quaternion(const quat_map & quat);
Spl_Quaternion(const Spl_Quaternion & x);
Spl_Quaternion & operator=(const Spl_Quaternion & x);
```

Description

Spl_Quaternion object constructor, copy constructor and equal operator.

Return Value

None

quat and quat_w

Syntax

```
short quat(const Real t, Quaternion & q);
short quat_w(const Real t, Quaternion & q, ColumnVector & w);
```

Description

These functions interpolate the spline at time t to sets the quaternion q and the angular velocity ω.

Return Value

Status, as a short int.

> 0 successful
>
> NOT_IN_RANGE (regarding t)

2.7 The `Trajectory_Select` class

This class deals with trajectory selection logic.

Constructor

Syntax

```
Trajectory_Select();
Trajectory_Select(const string & filename);
Trajectory_Select(const Trajectory_Select & x);
Trajectory_Select & operator=(const Trajectory_Select & x);
```

Description

Trajectory_Select object constructor, copy constructor and equal operator.

Return Value

None

set_trajectory

Syntax

```
void set_trajectory(const string & filename);
```

Description

This function reads the trajectory file (filename) and assign the spline data in class Spl_path or in class Spl_Quaternion.

Return Value

None

2.8 The `CLIK` class

The *CLICK* class deals with closed-loop inverse kinematics algorithm based on the unit quaternion [14].

Constructor

Syntax

```
Clik(){}
Clik(const Robot & robot_, const Matrix & Kp_, const Matrix & Ko_,
     const Real eps_=0.04, const Real lambda_max_=0.04,
     const Real dt=1.0);
Clik(const mRobot & mrobot_, const Matrix & Kp_, const Matrix & Ko_,
     const Real eps_=0.04, const Real lambda_max_=0.04,
     const Real dt=1.0);
Clik(const mRobot_min_para & mrobot_min_para_, const Matrix & Kp_,
     const Matrix & Ko_, const Real eps_=0.04,
     const Real lambda_max_=0.04, const Real dt=1.0);
Clik(const Clik & x);
Clik & operator=(const Clik & x);
```

Description

CLIK object constructor, copy constructor and equal operator.

Return Value

None

q_qdot

Syntax

```
void q_qdot(const Quaternion & qd, const ColumnVector & pd,
            const ColumnVector & pddot, const ColumnVector & wd,
            ColumnVector & q, ColumnVector & qp);
```

Description

This function sets the desired orientation joint position q and the desired joint velocity qp.

Return Value

None

2.9 The Proportional_Derivative class

The *Proportional_Derivative* class deals with the well known proportional derivative position controller.

Constructor

Syntax

```
Proportional_Derivative(const short dof = 1);
Proportional_Derivative(const Robot_basic & robot, const DiagonalMatrix & Kp,
                        const DiagonalMatrix & Kd);
Proportional_Derivative(const Proportional_Derivative & x);
```

Description

Proportional_Derivative object constructor, copy constructor and equal operator.

Return Value

None

torque_cmd

Syntax

```
ReturnMatrix torque_cmd(Robot_basic & robot, const ColumnVector & qd,
                        const ColumnVector & qpd);
```

Description

This function sets the output torque for a desired joint position vector, q_d, and a desired joint velocity vector, $\dot{q}_d$.

Return Value

`Matrix`

K_d, K_p

Syntax

```
short set_Kd(const DiagonalMatrix & Kd);
short set_Kp(const DiagonalMatrix & Kp);
```

Description

These functions sets the joint position error gain matrix, K_d, and the joint velocity error gain matrix, K_p.

Return Value

Status, as a short int.

> 0 successful
>
> WRONG_SIZE (regarding the input vector)

2.10 The `Computed_torque_method` class

The *Computed_torque_method* class deals with the well known computed torque method position controller [8].

Constructor

Syntax

```
Computed_torque_method();
Computed_torque_method(const Robot_basic & robot,
                       const DiagonalMatrix & Kd, const DiagonalMatrix & Kp);
Computed_torque_method(const Computed_torque_method & x);
Computed_torque_method & operator=(const Computed_torque_method & x);
```

Description

Computed_torque_method object constructor, copy constructor and equal operator.

Return Value

None

torque_cmd

Syntax

```
ReturnMatrix torque_cmd(Robot_basic & robot, const ColumnVector & qd,
                        const ColumnVector & qpd);
```

Description

This function sets the output torque for a desired joint position vector, q_d, and a desired joint velocity vector, $\dot{q}_d$.

Return Value

`Matrix`

K_d, K_p

Syntax

```
short set_Kp(const DiagonalMatrix & Kp);
short set_Kd(const DiagonalMatrix & Kd);
```

Description

These functions sets the joint position error gain matrix, K_p, and the joint velocity error gain matrix, K_d.

Return Value

Status, as a short int.

0 successful

WRONG_SIZE (regarding the input vector)

2.11 The `Resolve_acc` class

The *Resolve_acc* class deals with the resolve rate acceleration controller [15].

Constructor

Syntax

```
Resolved_acc();
Resolved_acc(const Robot_basic & robot,
             const double Kvp, const double Kpp,
             const double Kvo, const double Kpo);
Resolved_acc(const Resolved_acc & x);
Resolved_acc & operator=(const Resolved_acc & x);
```

Description

Resolve_acc object constructor, copy constructor and equal operator.

Return Value

None

torque_cmd

Syntax

```
ReturnMatrix torque_cmd(Robot_basic & robot, const ColumnVector & pdpp,
                        const ColumnVector & pdp, const ColumnVector & pd,
                        const ColumnVector & wdp, const ColumnVector & wd,
                        const Quaternion & qd, const short link_pc,
                        const Real dt);
```

Description

This function sets the output torque for the following desired end effector vector: acceleration, velocity, position, angular acceleration, angular velocity and angular position.

Return Value

`Matrix`

K_{pp}, K_{vp}, K_{po}, K_{vo}

Syntax

```
void set_Kpp(const double Kpp);
void set_Kvp(const double Kvp);
void set_Kpo(const double Kpo);
void set_Kvo(const double Kvo);
```

Description

These functions sets the end effector position error gain, K_{pp}, the velocity error gain, K_{vp}, the orientation error gain K_{po}, and the orientation angular rate gain, K_{vo}.

Return Value

None

2.12 The `Impedance` class

The *Impedance* class deals with the impedance controller [16]. This class should be use with the class *Resolve_acc*. *Resolve_acc* will make sure the end effector follow the compliant trajectory generated by *Impedance*. The end effector impedance is defined in terms of its translational and rotational part [16].

Constructor

Syntax

```
Impedance();
Impedance(const Robot_basic & robot, const DiagonalMatrix & Mp,
          const DiagonalMatrix & Dp, const DiagonalMatrix & Kp,
          const Matrix & Km,         const DiagonalMatrix & Mo,
          const DiagonalMatrix & Do, const DiagonalMatrix & Ko);
Impedance(const Impedance & x);
Impedance & operator=(const Impedance & x);
```

Description

Impedance object constructor, copy constructor and equal operator.

Return Value

None

control

Syntax

```
short control(const ColumnVector & pdpp, const ColumnVector & pdp,
              const ColumnVector & pd, const ColumnVector & wdp,
              const ColumnVector & wd, const Quaternion & qd,
              const ColumnVector & f, const ColumnVector & n,
              const Real dt);
```

Description

This function generate the compliant trajectory for a desired trajectory.

Return Value

Status, as a short int.

> 0 successful
>
> WRONG_SIZE (regarding the input vector)

M_p, D_p, K_p, M_o, D_o, K_o

Syntax

```
short set_Mp(const DiagonalMatrix & Mp);
short set_Mp(Real MP_i, const short i);
short set_Dp(const DiagonalMatrix & Dp);
short set_Dp(Real Dp_i, const short i);
short set_Kp(const DiagonalMatrix & Kp);
short set_Kp(Real Kp_i, const short i);
short set_Mo(const DiagonalMatrix & Mo);
short set_Mo(Real Mo_i, const short i);
short set_Do(const DiagonalMatrix & Do);
short set_Do(Real Do_i, const short i);
short set_Ko(const DiagonalMatrix & Ko);
short set_Ko(Real Ko_i, const short i);
```

Description

These functions sets the translational and rotational impedance parameters.

Return Value

Status, as a short int.

0 successful

WRONG_SIZE (regarding the input vector)

2.13 The `Control_Select` class

The *Control_Select* class deals with the controllers selection logic. It can be use to select any controllers mentioned above by reading the input file.

Constructor

Syntax

```
Control_Select();
Control_Select(const string & filename);
Control_Select(const Control_Select & x);
Control_Select & operator=(const Control_Select & x);
```

Description

Control_Select object constructor, copy constructor and equal operator.

Return Value

None

get_dof

Syntax

```
int get_dof();
```

Description

This function return the degree of freedom used in the selection.

Return Value

```
int
```

set_control

Syntax

```
void set_control(const string & filename);
```

Description

This function set the active controller.

Return Value

None

2.14 The `Stewart` class

Coming soon ... (based on [17]).

2.15 The `IO_matrix_file` class

Read and write functions are provided by the class IO_matrix_file. It is possible to read or write data at every iteration of the simulation using an instance of this class.

Constructor

Syntax

```
IO_matrix_file(const string & filename);
```

Description

IO_matrix_file object constructor.

Return Value

None

write

Syntax

```
short write(const vector<Matrix> & data);
short write(const vector<Matrix> & data, const vector<string> & data_title);
```

Description

This member function appends `data` to a file (specified by the constructor, and opened by write() when first called). `data_title` is used to write a header description at the beginning of the file. If it is not specified, a default description $datai, i = 1, 2, \cdots, n$ will be added. The header contains the number of iterations, the number of vectors and the data parameters, as follows:

nb_iterations 1269
nb_vector 2
nb_rows 1 nb_cols 1 time (s)
nb_rows 6 nb_cols 1 q(i) (rad)

Return Value

A short integer return the status:

0 successful,

IO_COULD_NOT_OPEN_FILE

IO_DATA_EMPTY

read

Syntax

```
short read(const vector<Matrix> & data);
short read(const vector<Matrix> & data, const vector<string> & data_title);
short read_all(vector<Matrix> & data, vector<string> & data_title);
```

Description

These member functions read `data` from a file (specified by the constructor, and opened when first called). `read()` reads the values corresponding to only one iteration, while `read_all()` reads the entire file at once.
These member functions are meant to read a file that was written using `write()`.

Return Value

Status, as a short int.

0 successful

IO_DATA_EMPTY

IO_COULD_NOT_OPEN_FILE

2.16 Graphics

Graphics are provided through calls to the gnuplot [1] software. Instances of the class `Plot2d` and `Plot_file` are used to generate the data and command files required by the call to gnuplot. A plot can be generated using the `set_plot2d` function.

[1] gnuplot is freely available from the following location: http://www.gnuplot.info/

`Plot2d` class

Constructor

Syntax

```
Plot2d(void);
```

Description

Upon initialization, a `Plot2d` object contain an empty graph. Data, title, label and other goodies can be added using the following member functions:

- `addcommand`;
- `addcurve`;
- `dump`;
- `gnuplot`;
- `settitle`;
- `setxlabel`;
- `setylabel`.

Return Value

`None`

addcommand

Syntax

```
void addcommand(const char * gcom);
```

Description

This function adds the command specified by the string `gcom` to the gnuplot command file. Ex: `mygraph.addcommand("set grid")`.
Note: see the gnuplot documentation for the list of commands.

Return Value

None

addcurve

Syntax

```
void addcurve(const Matrix & data,
              const char * label = "",
              int type = LINES);
```

Description

This function add the curves specified by the $n \times 2$ matrix `data` to the plot using the string `label` for the legend and `type` for the curve line type. Defined line types are:

- LINES;
- POINTS;
- LINESPOINTS;
- IMPULSES;
- DOTS;
- STEPS;
- BOXES.

See the `gnuplot` documentation for the description of these line types.

Return Value

None

dump

Syntax

```
void dump(void);
```

Description

This function dumps the current content of the object to `stdout`.

Return Value

None

gnuplot

Syntax

```
void gnuplot(void);
```

Description

This function calls gnuplot with the current content of the object.

Return Value

None

settitle

Syntax

```
void settitle(const char * t);
```

Description

This function sets the title of the graph to the string `t`.

Return Value

None

setxlabel

Syntax

```
void setxlabel(const char * t);
```

Description

This function sets the axis X label of the graph to the string `t`.

Return Value

None

setylabel

Syntax

```
void setylabel(const char * t);
```

Description

This function sets the axis Y label of the graph to the string `t`.

Return Value

None

`Plot_file` class

An instance of this class allows the creation of graphics from a data file. This file has to be created with an instance of the class IO_matrix_file.

Constructor

Syntax

```
Plot_file(const string & filename);
```

Description

Plot_file object constructor.

Return Value

```
None
```

graph

Syntax

```
short graph(const string & title_graph, const string & label, const short x,
            const short y, const short x_start, const short y_start,
            const short y_end);
```

Description

Create a graphic from a data file (specified by constructor). `title_graph` and `label` are used to provide the graphic title and label names in the legend. `x` refers to the index in the "`vector<Matrix> & data`" (in class `IO_Matrix_file`) corresponding to the `x` axis (ex: time), while `y` refers to the index in the "`vector<Matrix> & data`" corresponding to the `y` axis (ex: joints positions). `x_start`, `y_start` and `y_end` specify which rows of `data` to use.

Return Value

Status, as a short int.

0 successful

X_Y_DATA_NO_MATCH

PROBLEM_FILE_READING

set_plot2d

Syntax

```
void set_plot2d(const char *title_graph, const char *x_axis_title,
                const char *y_axis_title, const char *label, int type,
                const Matrix &xdata, const Matrix &ydata,
                int start_y, int end_y);

void set_plot2d(const char *title_graph, const char *x_axis_title,
                const char *y_axis_title, const std::vector<char *> label,
                int type, const Matrix &xdata, const Matrix &ydata,
                const std::vector<int> & data_select);
```

Description

This function generates a plot using a range (`start_y, end_y`) or a selection of columns (`data_select`) of the `ydtata` while setting the titles and labels.

Return Value

None

2.17 Config class

Config

Syntax

```
Config(const string & filename, const bool bPrintErrorMessages = true);
Config(const Config & x);
Config & operator=(const Config & x);
```

Description

This class provides a function to read a configuration.

Return Value

`None`

Reading and writing

Syntax

```
short read_conf();
short write_conf(const string filename, const string file_title,
                 const int space_between_column);
```

Description

The member function read_conf reads a configuration file (specified by constructor). The member function *write_conf* writes the configuration data in a file. A configuration file is divided in sections, which contain different parameters with their values. A section starts by [*section_name*] and contains one or more parameters an their values: *parameter_name: value* The ":" is mandatory between the name of the parameter and it's value. Lines beginning with a # and white/empty lines are ignored . The following example contains one section named *PUMA560_mDH*.

```
[PUMA560_mDH]
DH:         0
Fix:        1
MinPara:    0
dof:        6
Motor:      0
```

Return Value

Status, as a short int.

0 successful

CAN_NOT_OPEN_FILE

select

Syntax

```
short select(const string section, const string parameter,
                    string & value) const;
short select(const string section, const string parameter,
                  bool & value) const;
short select(const string section, const string parameter,
                   short & value) const;
short select(const string section, const string parameter,
                 int & value) const;
short select(const string section, const string parameter,
                    double & value) const;
short select(const string section, const string parameter,
                    float & value) const;
```

Description

These member functions are use to assign to the variable `value` the value of the parameter `parameter` from section section `section`.

Return Value

Status, as a short int.

> 0 successful
>
> SECTION_OR_PARAMETER_DOES_NOT_EXIST

add

Syntax

```
void add(const string section, const string parameter,
                const string value);
void add(const string section, const string parameter,
              const bool value);
void add(const string section, const string parameter,
               const short value);
void add(const string section, const string parameter,
             const int value);
void add(const string section, const string parameter,
                const double value);
void add(const string section, const string parameter,
                const float value);
```

Description

These member functions are use to add data into the data file structure. They will create the section and the parameter if it does not already exist.

Return Value

None

2.18 Miscellaneous

odeint

Syntax

```
void odeint(ReturnMatrix (*xdot)(Real time, const Matrix & xin),
            Matrix & xo,
            Real to,
            Real tf,
            Real eps,
            Real h1,
            Real hmin,
            int & nok,
            int & nbad,
            RowVector & tout,
            Matrix & xout,
            Real dtsav);
```

Description

This function performs the numerical integration of

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}(t), t) \tag{2.76}$$

using an adaptive step size based on 4^{th} order Runge-Kutta scheme. It carries out the integration of `xdot` with the initial conditions given by `xo`, from time `to` to `tf` with accuracy `eps` saving the results at `dtsav` increments. After the function call, `tout` is set as

$$\begin{bmatrix} t_0 & t_1 & \cdots & t_{nsteps} \end{bmatrix} \tag{2.77}$$

`xout` as

$$\begin{bmatrix} \boldsymbol{x}_0 & \boldsymbol{x}_1 & \cdots & \boldsymbol{x}_{nsteps} \end{bmatrix} \tag{2.78}$$

`xo` as $\boldsymbol{x}_{nsteps}$, `nok` and `nbad` to the number of good and bad steps taken. The function `odeint` is adapted from [18].

Return Value

None (`xo`, `tout` and `xout` are modified on output)

Runge_Kutta4

Syntax

```
void Runge_Kutta4(ReturnMatrix (*xdot)(Real time, const Matrix & xin),
                  const Matrix & xo,
                  Real to,
                  Real tf,
                  int nsteps,
                  RowVector & tout,
                  Matrix & xout);
```

Description

This function performs the numerical integration of

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}(t), t) \tag{2.79}$$

using a fixed step size 4^{th} order Runge-Kutta scheme. It carries out the integration of `xdot` with the initial conditions given by `xo`, from time `to` to `tf` with `nsteps`. After the function call, `tout` is set as

$$\left[\begin{array}{cccc} t_0 & t_1 & \cdots & t_{nsteps} \end{array} \right] \tag{2.80}$$

and `xout` as

$$\left[\begin{array}{cccc} \boldsymbol{x}_0 & \boldsymbol{x}_1 & \cdots & \boldsymbol{x}_{nsteps} \end{array} \right] \tag{2.81}$$

Return Value

None (`tout` and `xout` are modified on output)

Integ_Trap

Syntax

```
ReturnMatrix Integ_Trap(const ColumnVector & present, ColumnVector & past,
                        Real dt);
```

Description

This function performs the trapezoidal integration of the vector ***past*** to vector ***present*** over ***dt***.

Return Value

`Matrix`

pinv

Syntax

```
ReturnMatrix pinv(const Matrix & M);
```

Description

This function computes the pseudo inverse of the matrix M using SVD. If $A = U^*QV$ is a singular value decomposition of A, then $A^\dagger = V^*Q^\dagger U$ where X^* is the conjugate transpose of X and

$$Q^\dagger = \begin{bmatrix} 1/\sigma_1 & & & \\ & 1/\sigma_2 & & \\ & & \ddots & \\ & & & 0 \end{bmatrix}$$

where the $1/\sigma_i$ are replaced by 0 when $1/\sigma_i < tol$.

Return Value

`Matrix`

vec_dot_prod

Syntax

```
Real vec_dot_prod(const ColumnVector & x, const ColumnVector & y);
```

Description

This function performs the vector dot product on `x` and `y`.

Return Value

`ColumnVector`

x_prod_matrix

Syntax

```
ReturnMatrix x_prod_matrix(const ColumnVector & x);
```

Description

This function computes the cross product matrix $S(x)$ of `x` such that $S(x)y = x \times y$.

Return Value

`Matrix`

2.19 Summary of functions

Table 2.2: Homogeneous transforms

Homogeneous Transforms	
eulzxz	transform of Euler angles
ieulzxz	Euler angles of a transform
irotk	rotation around a unit vector of a transform
irpy	roll-pitch-yaw angles of a transform
rotd	transform of a rotation around a line segment
rotk	transform of a rotation around a unit vector
rpy	transform of roll-pitch-yaw angles
rotx	transform of a rotation around X axis
roty	transform of a rotation around Y axis
rotz	transform of a rotation around Z axis
trans	transform of a translation

Table 2.3: Quaternion class member functions

Quaternions	
+, -, *, /, =	operators on quaternions
conjugate, i	conjugate (or inverse) of a quaternion
exp, Log, power	exponential, logarithm and power of a quaternion
dot_prod	dot product of a quaternion
dot, E	quaternion time derivative
unit	make a quaternion a unit quaternion
norm, norm_sqr	compute the norm and the square norm of a quaternion
s, v	returns the scalar and the vector of a quaternion
set_s, set_v	assign values to the scalar and vector part of a quaternion
R, T	returns the equivalent rotation matrix (3×3 or 4×4)

Table 2.4: Quaternion non member functions

Functions	
Omega	returns angular velocity
Slerp	Spherical Linear Interpolation
Slerp_prime	Spherical Linear Interpolation derivative
Squad	Spherical Cubic Interpolation
Squad_prime	Spherical Cubic Interpolation derivative

Table 2.5: Spl_Quaternion class member function

Spl_Quaternion	
quat	interpolate the spline at time t to sets the quaternion q.
quat_w	interpolate the spline at time t to sets the quaternion q and angular velocity ω.

Table 2.6: Spl_Cubic class member function

Spl_Cubic	
interpolating	interpolate the spline at time t.
first_derivative	interpolate the spline first derivative at time t.
second_derivative	interpolate the spline second derivative at time t.

Table 2.7: Spl_path class member function

Spl_path	
p	interpolate the spline at time t to sets the position.
p_pdot	interpolate the spline at time t to sets position and velocity.
p_pdot_pddot	interpolate the spline at time t to sets position, velocity and acceleration.

Table 2.8: CLIK class member function

CLIK	
q_qdot	sets the desired joint position and joint velocity

Table 2.9: Computed_torque_method class member function

Computed_torque_method	
torque_cmd	sets the output torque
set_Kd	sets the derivative error gain
set_Kp	sets the position error gain

Table 2.10: Resolve_acc class member function

Resolve_acc	
torque_cmd	sets the output torque
set_Kvp	sets the translational velocity error gain
set_Kpp	sets the translational position error gain
set_Kvo	sets the rotational velocity error gain
set_Kpo	sets the rotational position error gain

Table 2.11: Impedance class member function

Impedance	
control	sets the compliant trajectory
set_Mp	sets the translational impedance inertia matrix
set_Dp	sets the translational impedance damping matrix
set_Kp	sets the translational impedance stiffness matrix
set_Mo	sets the rotational impedance inertia matrix
set_Do	sets the rotational impedance damping matrix
set_Ko	sets the rotational impedance stiffness matrix

Table 2.12: IO_matrix_file class member functions

IO_matrix_file	
write	create and write data to a file
read	read data from a file
read_all	read entire data file at once

Table 2.13: Plot2d class member functions

Plot2d	
addcommand	add a `gnuplot` command the 2d graph
addcurve	add a curve to the 2d graph
dump	dump the content of the graph to `stdout`
gnuplot	plot the graph through a call to `gnuplot`
settitle	sets graph title
setxlabel	sets axis X label
setylabel	sets axis Y label
set_plot2d	"wrapper" function for `Plot2d`

Table 2.14: Plot_file class member functions

Plot_file	
graph	create a graphics from a data file

Table 2.15: Config class member functions

Config	
read_conf	read configuration file
select	assign the value of parameter from a section
add	specify the value of parameter for a section

Table 2.16: `Robot` (and `mRobot`) class member functions

Joint Variables	
get_q	get the robot joint variables position
get_qp	get the robot joint variables velocity
get_qpp	get the robot joint variables acceleration
set_q	set the robot joint variables position
set_qp	set the robot joint variables velocity
set_qpp	set the robot joint variables acceleration
Robot Kinematics	
inv_kin inv_kin_rhino inv_kin_puma	inverse kinematics Rhino inverse kinematics Puma inverse kinematics
jacobian jacobian_dot jacobian_DLS_inv	robot Jacobian robot Jacobian derivative robot Jacobian DLS inverse
kine, kine_pd	forward kinematics
dTdqi	partial derivative of forward kinematics
Robot Dynamics	
acceleration	forward dynamics
inertia	robot inertia matrix
torque	inverse dynamics
torque_novelocity	inverse dynamics without velocity and gravity
G	gravity effects
C	Coriolis and centrifugal effects
Robot Linearized Dynamics	
delta_torque	$\delta\boldsymbol{\tau} = \boldsymbol{D}(\boldsymbol{q})\delta\ddot{\boldsymbol{q}} + \boldsymbol{S_1}(\boldsymbol{q},\dot{\boldsymbol{q}})\delta\dot{\boldsymbol{q}} + \boldsymbol{S_2}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}})\delta\boldsymbol{q}$
dq_torque	$\boldsymbol{S_2}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}})\delta\boldsymbol{q}$
dqp_torque	$\boldsymbol{S_1}(\boldsymbol{q},\dot{\boldsymbol{q}})\delta\dot{\boldsymbol{q}}$
dtau_dq	$\frac{\partial\boldsymbol{\tau}}{\partial\boldsymbol{q}} = \boldsymbol{S_2}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}})$
dtau_dqp	$\frac{\partial\boldsymbol{\tau}}{\partial\dot{\boldsymbol{q}}} = \boldsymbol{S_1}(\boldsymbol{q},\dot{\boldsymbol{q}})$

Table 2.17: Miscellaneous

Miscellaneous	
odeint	adaptive step size Runge-Kutta integrator
Runge_Kutta4	fixed step size 4^{th} order Runge-Kutta integrator
Integ_Trap	trapezoidal integration
pinv	matrix pseudo inverse
vec_dot_prod	vector dot product
vec_x_prod	vector cross product
x_prod_matrix	cross product matrix
perturb_robot	perturb robot parameters

Chapter 3

Reporting bugs, contributions and comments

I intend to support this library. By this, I mean that bugs will be fixed as fast as time allows me and that new functionalities will be introduced in future releases. If you find a bug or think some part of the documentation could be improved, let me know and I will try to include the corrections in the next release. Comments regarding the documentation will not be treated as fast as bug reports. I will not, however, help users with problems related to assignments and homework. You can use your Web browser to send comments or bug report with the URL:
`http://www.cours.polymtl.ca/roboop/`.
If you don't have access to a Web browser, send email to
`richard.gourdeau@polymtl.ca`.

3.1 Reporting bugs

When reporting bugs, please send the following information (see the file `bugs.txt`):

```
VERSION OF THE PACKAGE (see the readme.txt file):

OS:

COMPILER:

DESCRIPTION OF THE BUG:
```

```
SAMPLE CODE THAT MAKE THE BUG APPARENT:
```

or use the URL: `http://www.cours.polymtl.ca/roboop/`.

3.2 Making a contribution to the package

If you have written some code you think might be useful for other users of the package, I will be happy to integrate it in future releases. Makefiles for compilers not included in this distribution would be greatly appreciated. Contact me for more details: `richard.gourdeau@polymtl.ca`.

3.3 Citing the package

If you are using the ROBOOP package, please let me know. If you want to cite this package in some of your work, please use [19] or the following BibTeX entry:

```
@ARTICLE{Gourdeau97,
  author = {Richard Gourdeau},
  month = sep,
  year = 1997,
  title = {Object Oriented Programming for Robotic
           Manipulators Simulation},
  journal = {IEEE Robotics and Automation Magazine},
  volume = 4,
  number = 3,
  pages = {21--29}
}
```

Chapter 4

Credits and acknowledgments

I would like to thank Robert Davies for making his NEWMAT11 library available.

The hardware and software used to develop the initial releases of this package were funded through NSERC grants OGP0138478 and EQP0172766.

I would like to thank Etienne Lachance for his contributions since the 1.13 release and Samuel Belanger for the initial version of the Stewart class.

Chapter 5

Future developments

In future releases, we hope to include the following:

- functions for basic control laws (sliding modes, etc);
- make files for other compilers.

Bibliography

[1] Jack C. K. Chou, "Quaternion kinematic and dynamic differential equations", *IEEE Trans. of Robotics and Automation*, vol. 1, no. 8, pp. 53–64, Feb. 1992.

[2] M. Lillholm E.B. Dam, M. Koch and, "Quaternions, interpolation and animation", Tech. Rep. DIKU-TR-98/5, University of Copenhagen, July 1998.

[3] J. Denavit and R. S. Hartenberg, "A kinematic notation for lower pair mechanisms based on matrices", *ASME Jour. of Applied Mechanics*, pp. 215–221, June 1955.

[4] J. J. Craig, *Introduction to Robotics: Mechanics and Control*, Addison-Wesley Publising Company, 2nd edition, 1989.

[5] Bruce Eckel, *C++ inside & out*, Osborne, McGraw-Hill, 1993.

[6] B. Gorla and M. Renaud, *Modèles des robots manipulateurs, application à leur commande*, Cepadues-éditions, Toulouse, mai 1984.

[7] J. J. Uicker, "Dynamic force analysis of spatial linkages", *ASME Jour. of Applied Mechanics*, vol. 34, pp. 418–424, June 1967.

[8] K. S. Fu, R. C. Gonzalez, and C. S. G. Lee, *Robotics: Control, Sensing, Vision, and Intelligence*, McGraw-Hill, New York, 1987.

[9] Jorge Angeles, *Fundamentals of Robotic Mechanical Systems: Theory, Methods and Algorithms*, Mechanical Engineering Series. Springer-Verlag, 1997.

[10] S. Chiaverini, B. Siciliano, and O. Egeland, "Review of the damped least-squares inverse kinematics with experiments on an industrial robot manipulator", *IEEE Trans. on Control Systems Technology*, vol. 2, no. 2, pp. 123–134, June 1994.

[11] M. W. Walker and D. E. Orin, "Efficient dynamic computer simulation of robotic mechanisms", *ASME Jour. of Dynamic Systems, Measurement, and Control*, vol. 104, pp. 205–211, 1982.

[12] J. Y. S. Luh, M. W. Walker, and R. P. C. Paul, "On-line computational scheme for mechanical manipulators", *ASME Jour. of Dynamic Systems, Measurement, and Control*, vol. 102, pp. 69–76, June 1980.

[13] J. J. Murray and C. P. Neuman, "Linearization and sensitivity models of the Newton-Euler dynamic robot model", *ASME Jour. of Dynamic Systems, Measurement, and Control*, vol. 108, pp. 272–276, Sept. 1986.

[14] S. Chiaverini and B. Siciliano, "The unit quaternion: A useful tool for inverse kinematics of robot manipulators", *Systems Analysis, Modeling and Simulation*, vol. 35, pp. 45–60, 1999.

[15] F. Caccavale, C. Natale, B. Siciliano, and L. Villani, "Resolved-acceleration control of robot manipulators: A critical review with experiments", *Robotica*, vol. 16, pp. 565–573, 1998.

[16] F. Caccavale and B. Siciliano, "Six-dof impedance control based on angle/axis representations", *IEEE Trans. of Robotics and Automation*, vol. 15, pp. 289–300, 1999.

[17] K. Harib and K. Srinivasan, "Kinematic and dynamic analysis of Stewart platform-based machine tool structures", *Robotica*, vol. 21, pp. 541–554, 2003.

[18] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling, *Numerical Recipes in C, The Art of Scientific Computing*, Cambridge University Press, 1988.

[19] Richard Gourdeau, "Object oriented programming for robotic manipulators simulation", *IEEE Robotics and Automation Magazine*, vol. 4, no. 3, pp. 21–29, Sept. 1997.

Appendix A

Recursive Newton-Euler algorithms, DH notation

In order to apply the RNE as presented in [13], let us define the following variables (referenced in the i^{th} coordinate frame if applicable):

- σ_i is the joint type; $\sigma_i = 1$ for a revolute joint and $\sigma_i = 0$ for a prismatic joint;
- $\boldsymbol{p}_i = \begin{bmatrix} a_i & d_i \sin\alpha_i & d_i \cos\alpha_i \end{bmatrix}^T$ is the position of the i^{th} with respect to the $i-1^{th}$ frame;
- $\boldsymbol{z}_0 = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$

A.1 Recursive Newton-Euler formulation

- Forward Iterations for $i = 1, 2, \ldots, n$.

  Initialize: $\omega_0 = \dot{\omega}_0 = 0$ and $\dot{\boldsymbol{v}}_0 = -\boldsymbol{g}$.

$$\omega_i = \boldsymbol{R}_i^T[\omega_{i-1} + \sigma_i \boldsymbol{z}_0 \dot{\theta}_i] \tag{A.1}$$

$$\dot{\omega}_i = \boldsymbol{R}_i^T\{\dot{\omega}_{i-1} + \sigma_i[\boldsymbol{z}_0\ddot{\theta}_i + \omega_{i-1} \times (\boldsymbol{z}_0\dot{\theta}_i)]\} \tag{A.2}$$

$$\begin{aligned}\dot{\boldsymbol{v}}_i = {} & \boldsymbol{R}_i^T\{\dot{\boldsymbol{v}}_{i-1} + (1-\sigma_i)[\boldsymbol{z}_0\ddot{d}_i + 2\omega_{i-1} \times (\boldsymbol{z}_0\dot{d}_i)]\} \\ & + \dot{\omega}_i \times \boldsymbol{p}_i + \omega_i \times (\omega_i \times \boldsymbol{p}_i)\end{aligned} \tag{A.3}$$

- Backward Iterations for $i = n, n-1, \ldots, 1$.

Initialize: $\boldsymbol{f}_{n+1} = \boldsymbol{n}_{n+1} = 0$.

$$\dot{\boldsymbol{v}}_{ci} = \boldsymbol{v}_i + \omega_i \times \boldsymbol{r}_i + \omega_i \times (\omega_i \times \boldsymbol{r}_i) \quad (A.4)$$
$$\boldsymbol{F}_i = m_i \dot{\boldsymbol{v}}_{ci} \quad (A.5)$$
$$\boldsymbol{N}_i = \boldsymbol{I}_{ci}\dot{\omega}_i + \omega_i \times (\boldsymbol{I}_{ci}\omega_i) \quad (A.6)$$
$$\boldsymbol{f}_i = \boldsymbol{R}_{i+1}[\boldsymbol{f}_{i+1}] + \boldsymbol{F}_i \quad (A.7)$$
$$\boldsymbol{n}_i = \boldsymbol{R}_{i+1}[\boldsymbol{n}_{i+1}] + \boldsymbol{p}_i \times \boldsymbol{f}_i + \boldsymbol{N}_i + \boldsymbol{r}_i \times \boldsymbol{F}_i \quad (A.8)$$
$$\tau_i = \sigma_i \boldsymbol{n}_i^T(\boldsymbol{R}_i^T \boldsymbol{z}_0) + (1-\sigma_i)\boldsymbol{f}_i^T(\boldsymbol{R}_i^T \boldsymbol{z}_0) \quad (A.9)$$

A.2 Recursive linearized Newton-Euler formulation

With

$$\boldsymbol{p}_{di} = \frac{\partial \boldsymbol{p}_i}{\partial d_i} = \begin{bmatrix} 0 & \sin\alpha_i & \cos\alpha_i \end{bmatrix}^T \quad (A.10)$$
$$\boldsymbol{Q} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad (A.11)$$

one can use the following

- Forward Iterations for $i = 1, 2, \ldots, n$.

  Initialize: $\delta\omega_0 = \delta\dot{\omega}_0 = \delta\dot{\boldsymbol{v}}_0 = 0$.

$$\delta\omega_i = \boldsymbol{R}_i^T\{\delta\omega_{i-1} + \sigma_i[\boldsymbol{z}_0\delta\dot{\theta}_i - \boldsymbol{Q}(\omega_{i-1} + \dot{\theta}_i)\delta\theta_i]\} \quad (A.12)$$
$$\begin{aligned}\delta\dot{\omega}_i &= \boldsymbol{R}_i^T\{\delta\dot{\omega}_{i-1} + \sigma_i[\boldsymbol{z}_0\delta\ddot{\theta}_i + \delta\omega_{i-1} \times (\boldsymbol{z}_0\dot{\theta}_i) + \omega_{i-1} \times (\boldsymbol{z}_0\delta\dot{\theta}_i)] \\ &\quad -\sigma_i\boldsymbol{Q}[\omega_{i-1} + \boldsymbol{z}_0\ddot{\theta}_i + \omega_{i-1} \times (\boldsymbol{z}_0\dot{\theta}_i)]\delta\theta_i\} \end{aligned} \quad (A.13)$$
$$\begin{aligned}\delta\dot{\boldsymbol{v}}_i &= \boldsymbol{R}_i^T\{\delta\dot{\boldsymbol{v}}_{i-1} - \sigma_i\boldsymbol{Q}\dot{\boldsymbol{v}}_{i-1}\delta\theta_i \\ &\quad +(1-\sigma_i)[\boldsymbol{z}_0\delta\ddot{d}_i + 2\delta\omega_{i-1} \times (\boldsymbol{z}_0\dot{d}_i) + 2\omega_{i-1} \times (\boldsymbol{z}_0\delta\dot{d}_i)]\} \\ &\quad +\delta\dot{\omega}_i \times \boldsymbol{p}_i + \delta\omega_i \times (\omega_i \times \boldsymbol{p}_i) + \omega_i \times (\delta\omega_i \times \boldsymbol{p}_i) \\ &\quad +(1-\sigma_i)(\dot{\omega}_i \times \boldsymbol{p}_{di} + \omega_i \times (\omega_i \times \boldsymbol{p}_{di}))\delta d_i \end{aligned} \quad (A.14)$$

- Backward Iterations for $i = n, n-1, \ldots, 1$.

  Initialize: $\delta\boldsymbol{f}_{n+1} = \delta\boldsymbol{n}_{n+1} = 0$.

$$\delta\dot{\boldsymbol{v}}_{ci} = \delta\boldsymbol{v}_i + \delta\omega_i \times \boldsymbol{r}_i + \delta\omega_i \times (\omega_i \times \boldsymbol{r}_i) + \omega_i \times (\delta\omega_i \times \boldsymbol{r}_i) \quad (A.15)$$

$$
\begin{aligned}
\delta \boldsymbol{F}_i &= m_i \delta \dot{\boldsymbol{v}}_{ci} && \text{(A.16)} \\
\delta \boldsymbol{N}_i &= \boldsymbol{I}_{ci} \delta \dot{\omega}_i + \delta \omega_i \times (\boldsymbol{I}_{ci} \omega_i) + \omega_i \times (\boldsymbol{I}_{ci} \delta \omega_i) && \text{(A.17)} \\
\delta \boldsymbol{f}_i &= \boldsymbol{R}_{i+1} [\delta \boldsymbol{f}_{i+1}] + \delta \boldsymbol{F}_i + \sigma_{i+1} \boldsymbol{Q} \boldsymbol{R}_{i+1} [\boldsymbol{f}_{i+1}] \delta \theta_{i+1} && \text{(A.18)} \\
\delta \boldsymbol{n}_i &= \boldsymbol{R}_{i+1} [\delta \boldsymbol{n}_{i+1}] + \delta \boldsymbol{N}_i + \boldsymbol{p}_i \times \delta \boldsymbol{f}_i + \boldsymbol{r}_i \times \delta \boldsymbol{F}_i \\
&\quad + (1 - \sigma_i)(\boldsymbol{p}_{di} \times \boldsymbol{f}_i) \delta d_i + \sigma_{i+1} \boldsymbol{Q} \boldsymbol{R}_{i+1} [\boldsymbol{n}_{i+1}] \delta \theta_{i+1} && \text{(A.19)} \\
\delta \tau_i &= \sigma_i [\delta \boldsymbol{n}_i^T (\boldsymbol{R}_i^T \boldsymbol{z}_0) - \boldsymbol{n}_i^T (\boldsymbol{R}_i^T \boldsymbol{Q} \boldsymbol{z}_0) \delta \theta_i] \\
&\quad + (1 - \sigma_i) [\delta \boldsymbol{f}_i^T (\boldsymbol{R}_i^T \boldsymbol{z}_0)] && \text{(A.20)}
\end{aligned}
$$

Appendix B

Recursive Newton-Euler algorithms, modified DH notation

In order to apply the RNE, let us define the following variables (referenced in the i^{th} coordinate frame if applicable):

- σ_i is the joint type; $\sigma_i = 1$ for a revolute joint and $\sigma_i = 0$ for a prismatic joint;
- $\boldsymbol{p}_i = \begin{bmatrix} a_{i-1} & -d_i sin\alpha_{i-1} & d_i cos\alpha_{i-1} \end{bmatrix}^T$ is the position of the i^{th} with respect to the $i-1^{th}$ frame;
- $\boldsymbol{z}_0 = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$

B.1 Recursive Newton-Euler formulation

- Forward Iterations for $i = 1, 2, \ldots, n$.

  Initialize: $\omega_0 = \dot{\omega}_0 = 0$ and $\dot{\boldsymbol{v}}_0 = -\boldsymbol{g}$.

$$\omega_i = \boldsymbol{R}_i^T \omega_{i-1} + \sigma_i \boldsymbol{z}_0 \dot{\theta}_i \quad (B.1)$$

$$\dot{\omega}_i = \boldsymbol{R}_i^T \dot{\omega}_{i-1} + \sigma_i \boldsymbol{R}_i^T \omega_{i-1} \times \boldsymbol{z}_0 \dot{\theta}_i + \sigma_i \boldsymbol{z}_0 \ddot{\theta}_i \quad (B.2)$$

$$\begin{aligned} \dot{\boldsymbol{v}}_i = {} & \boldsymbol{R}_i^T (\dot{\omega}_{i-1} \times \boldsymbol{p}_i + \omega_{i-1} \times (\omega_{i-1} \times \boldsymbol{p}_i) + \dot{\boldsymbol{v}}_{i-1}) \\ & + (1 - \sigma_i)(2\omega_i \times \boldsymbol{z}_0 \dot{d}_i + \boldsymbol{z}_0 \ddot{d}_i) \end{aligned} \quad (B.3)$$

- Backward Iterations for $i = n, n-1, \ldots, 1$.

  Initialize: $\boldsymbol{f}_{n+1} = \boldsymbol{n}_{n+1} = 0$.

$$\dot{\boldsymbol{v}}_{ci} = \dot{\omega}_i \times \boldsymbol{r}_i + \omega_i \times (\omega_i \times \boldsymbol{r}_i) + \boldsymbol{v}_i \tag{B.4}$$
$$\boldsymbol{F}_i = m_i \dot{\boldsymbol{v}}_{ci} \tag{B.5}$$
$$\boldsymbol{N}_i = \boldsymbol{I}_{ci}\ddot{\omega}_i \ + \omega_i \times \boldsymbol{I}_{ci}\omega_i \tag{B.6}$$
$$\boldsymbol{f}_i = \boldsymbol{R}_{i+1}\boldsymbol{f}_{i+1} + \boldsymbol{F}_i \tag{B.7}$$
$$\boldsymbol{n}_i = \boldsymbol{N}_i + \boldsymbol{R}_{i+1}\boldsymbol{n}_{i+1} + \boldsymbol{r}_i \times \boldsymbol{F}_i + \boldsymbol{p}_{i+1} \times \boldsymbol{R}_{i+1}\boldsymbol{f}_{i+1} \tag{B.8}$$
$$\tau_i = \sigma_i \boldsymbol{n}_i \boldsymbol{z}_) + (1-\sigma_i)\boldsymbol{f}_i^T \boldsymbol{z}_0 \tag{B.9}$$

B.2 Recursive linearized Newton-Euler formulation

With

$$\boldsymbol{p}_{di} = \frac{\partial \boldsymbol{p}_i}{\partial d_i} = \begin{bmatrix} 0 & -\sin\alpha_{i-1} & \cos\alpha_{i-1} \end{bmatrix}^T \tag{B.10}$$
$$\boldsymbol{Q} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{B.11}$$

one can use the following

- Forward Iterations for $i = 1, 2, \ldots, n$.

  Initialize: $\delta\omega_0 = \delta\dot{\omega}_0 = \delta\dot{\boldsymbol{v}}_0 = 0$.

$$\delta\omega_i = \boldsymbol{R}_i^T \delta\omega_{i-1} + \sigma_i(\boldsymbol{z}_0 \delta\dot{\theta}_i \ - \ \boldsymbol{Q}\boldsymbol{R}_i^T \omega_i \delta\theta_i) \tag{B.12}$$
$$\begin{aligned}\delta\dot{\omega}_i = {} & \boldsymbol{R}_i^T \delta\dot{\boldsymbol{w}}_{i-1} + \sigma_i[\boldsymbol{R}_i^T \delta\omega_{i-1} \times \boldsymbol{z}_0 \dot{\theta}_i \\ & + \boldsymbol{R}_i^T \omega_{i-1} \times \boldsymbol{z}_0 \delta\dot{\theta}_i + \boldsymbol{z}_0 \ddot{\theta}_i \\ & - (\boldsymbol{Q}\boldsymbol{R}_i^T \dot{\omega}_{i-1} + \boldsymbol{Q}\boldsymbol{R}_i^T \omega_{i-1} \times \omega z_0 \dot{\theta}_i)\delta\theta_i]\end{aligned} \tag{B.13}$$
$$\begin{aligned}\delta\dot{\boldsymbol{v}}_i = {} & \boldsymbol{R}_i^T \Big(\delta\dot{\omega}_{i-1} \times \boldsymbol{p}_i + \delta\omega_{i-1} \times (\omega_{i-1} \times \boldsymbol{p}_i) \\ & + \omega_{i-1} \times (\delta\omega_{i-1} \times \boldsymbol{p}_i) + \delta\dot{\boldsymbol{v}}_i\Big) \\ & + (1-\sigma_i)\Big(2\delta\omega_i \times \boldsymbol{z}_0 \dot{\boldsymbol{d}}_i + 2\omega_i \times \boldsymbol{z}_0 \delta\dot{\boldsymbol{d}}_i + \boldsymbol{z}_0 \delta\ddot{\boldsymbol{d}}_i\Big) \\ & - \sigma_i \boldsymbol{Q}\boldsymbol{R}_i^T \Big(\dot{\omega}_{i-1} \times \boldsymbol{p}_i + \omega_{i-1} \times (w_{i-1} \times p_i) + \dot{\boldsymbol{v}}_i\Big)\delta\theta_i \\ & + (1-\sigma_i)\boldsymbol{R}_i^T \Big(\dot{\omega}_{i-1} \times \boldsymbol{p}_{di} + \omega_{i-1} \times (\omega_{i-1} \times \boldsymbol{p}_{di})\Big)\delta\boldsymbol{d}_i\end{aligned} \tag{B.14}$$

- Backward Iterations for $i = n, n-1, \ldots, 1$.

  Initialize: $\delta \boldsymbol{f}_{n+1} = \delta \boldsymbol{n}_{n+1} = 0$.

$$\begin{aligned}
\delta \dot{\boldsymbol{v}}_{ci} &= \delta \dot{\boldsymbol{v}}_i + \delta \dot{\omega}_i \times \boldsymbol{r}_i + \delta \omega_i \times (\omega_i \times \boldsymbol{r}_i) \\
&\quad + \omega_i \times (\delta \omega_i \times \boldsymbol{r}_i) && \text{(B.15)} \\
\delta \boldsymbol{F}_i &= m_i \delta \dot{\boldsymbol{v}}_{ci} && \text{(B.16)} \\
\delta \boldsymbol{N}_i &= \boldsymbol{I}_{ci} \delta \dot{\omega}_i + \delta \omega_i \times (\boldsymbol{I}_{ci} \omega_i) + \omega_i \times (\boldsymbol{I}_{ci} \delta \omega_i) && \text{(B.17)} \\
\delta \boldsymbol{f}_i &= \boldsymbol{R}_{i+1} \delta \boldsymbol{f}_{i+1} + \delta \boldsymbol{F}_i + \sigma_{i+1} \boldsymbol{R}_{i+1} \boldsymbol{Q} \boldsymbol{f}_{i+1} \delta \theta_{i+1} && \text{(B.18)} \\
\delta \boldsymbol{n}_i &= \delta \boldsymbol{N}_i + \boldsymbol{R}_{i+1} \delta \boldsymbol{n}_{i+1} + \boldsymbol{r}_i \times \delta \boldsymbol{F}_i && \text{(B.19)} \\
&\quad + \boldsymbol{p}_{i+1} \times \boldsymbol{R}_{i+1} \delta \boldsymbol{f}_{i+1} \\
&\quad + \sigma_{i+1} \Big(\boldsymbol{R}_{i+1} \boldsymbol{Q} \boldsymbol{n}_{i+1} + \boldsymbol{p}_{i+1} \times \boldsymbol{R}_{i+1} \boldsymbol{Q} \boldsymbol{f}_{i+1} \Big) \delta \theta_{i+1} \\
&\quad + (1 - \sigma_{i+1}) \boldsymbol{p}_{di+1} \boldsymbol{p}_{di+1} \times \boldsymbol{R}_{i+1} \boldsymbol{f}_{i+1} \delta d_{i+1} \\
\delta \boldsymbol{\tau}_i &= \sigma \delta \boldsymbol{n}_i^T \boldsymbol{z}_0 + (1 - \sigma_i) \delta \boldsymbol{f}_i^T \boldsymbol{z}_0 && \text{(B.20)}
\end{aligned}$$

Appendix C

GNU Lesser General Public License

Content of the file `GNUlgpl.txt`.

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999
Copyright (C) 1991, 1999 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.
[This is the first released version of the Lesser GPL. It also counts as the successor of the GNU Library Public License, version 2, hence the version number 2.1.]

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages—typically libraries—of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

GNU LESSER GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

   A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a) The modified work must itself be a software library.

   b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

   c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

   d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

   Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

   This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

   If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

   However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

   When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

   If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

   Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

   You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

   a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

   a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

    11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

    If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

    It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

    Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

    NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the library's name and a brief idea of what it does.>
Copyright (C) <year>  <name of author>

This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public
License along with this library; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
```

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the
library 'Frob' (a library for tweaking knobs) written by James Random Hacker.

<signature of Ty Coon>, 1 April 1990
Ty Coon, President of Vice
```

That's all there is to it!