ROBOOP
A Robotics Object Oriented Package in C++
version 1.09

Documentation

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

September 27, 1998

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.

1.2 Requirements

This work uses the matrix library `NEWMAT09` [2] developed by Robert Davies. Hence, the requirement for the ROBOOP are the same as for the `NEWMAT09`. Although make files are only provided for the Borland C++ 4.5, Visual C++ 6.0, Watcom C++ 10.0 and GNU G++ compilers, the following compilers could be used:

- AT&T C++ 2.1;3.0.1 on a Sun;
- Borland C++ 3.1, 4.5, 5.0;
- Gnu G++ 2.3.3, 2.5.8, 2.6.0, 2.7.2;
- Microsoft C++ (7.0, 8.0);

[1] Program source and documentation are available from the URL: `http://www.ind2.polymtl.ca/ROBOOP`

[2] available from the site `http://nz.com/webnz/robert/index.html`

- Microsoft Visual C++ 2.0;
- Sun C++ (version 4.0.1);
- Watcom C++ (version 10.0).

See the file `newmat.txt` in the `newmat` directory for more details.

In order to use the graphic features of this package, the software `gnuplot`[3] (version 3.5 on later) must be installed in the `PATH` of your computer (the binary name is `wgnupl32.exe` under Windows 95/NT and `gnuplot` under most of other platforms).

1.3 Copyright

1.4 Version history

version 1.09 (98/09/27) Makefile for MS Visual C++ 6.0. The program files in this version are the following revisions:

```
Id: bench.cpp,v 1.8 1998-05-10 21:08:36-04
Id: comp_dq.cpp,v 1.6 1998-05-10 21:08:37-04
Id: comp_dqp.cpp,v 1.6 1998-05-10 21:08:38-04
Id: delta_t.cpp,v 1.6 1998-05-10 21:08:38-04
Id: demo.cpp,v 1.11 1998/09/20 14:06:05
Id: dynamics.cpp,v 1.7 1998-05-10 21:08:39-04
```

[3] `gnuplot is freely available from the following location:`
`http://www.cs.dartmouth.edu/gnuplot_info.html`

```
Id: gnugraph.cpp,v 1.10 1998/09/20 14:44:54
Id: homogen.cpp,v 1.5 1998-05-31 23:20:55-04
Id: kinemat.cpp,v 1.7 1998-05-10 21:08:41-04
Id: robot.cpp,v 1.7 1998-05-12 22:29:23-04
Id: sensitiv.cpp,v 1.4 1998-05-10 21:08:42-04
Id: utils.cpp,v 1.6 1998-05-10 21:08:43-04
Id: robot.h,v 1.10 1998-05-12 22:29:24-04
```

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.

1.5 Files in the distribution

readme	txt	readme file
roboop	gnu	make file for GNU G++
roboop	gw32	make file for GNU-Win32 (Windows 95/NT)
roboop	mak	make file for Borland C++ 4.5 (Windows 95/NT)
roboop	mk	make file for Watcom C++ 10.0 (Windows 95/NT)
bench	mk1	used by the make file roboop.mk
demo	mk1	used by the make file roboop.mk
newmat	mk1	used by the make file roboop.mk
roboop	mk1	used by the make file roboop.mk
roboop	nt	make file for Visual C++ 6.0 (Windows 95/NT)
demo	txt	output of the demo program
rr_dat	txt	example of robot data file
newmat		directory of the matrix library NEWMAT09 see the file `newmat.txt`
docs		documentation directory
gnulgpl	txt	GNU Library General Public License
robot	ps	documentation in `postscript` format
source		the ROBOOP program source directory
robot	h	header file
bench	cpp	benchmark program file
comp_dq	cpp	simpified version of `delta_t` with no `dqp` and `dqpp`
comp_dqp	cpp	simpified version of `delta_t` with no `dq` and `dqpp`
delta_t	cpp	compute torque variation w/r to `dq`, `dqp` and `dqpp`
demo	cpp	demo program file
dynamics	cpp	dynamics functions
gnugraph	cpp	graphics functions
homogen	cpp	homogeneous transform functions
kinemat	cpp	kinematics functions
robot	cpp	constructors and other stuff
sensitiv	cpp	partial derivatives of robot dynamics
utils	cpp	miscellaneous

Chapter 2

Reference manual

This package uses data types defined by the `NEWMAT09` 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 `NEWMAT09` 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 lenght 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(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 lenght of at least 3. Only the first 3 elements are used.

Return Value

`Matrix`

rotk

Syntax

```
ReturnMatrix rotk(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 lenght 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 lenght of at least 3. Only the first 3 elements are used.

Return Value

`Matrix`

rotx, roty, rotz

Syntax

```
ReturnMatrix rotx(Real alpha);
ReturnMatrix roty(Real beta);
ReturnMatrix rotz(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 lenght of at least 3. Only the first 3 elements are used.

Return Value

`Matrix`

2.2 The Robot class

The Robot class is composed of the following data elements:

- the number of degree of freedom n (`int dof`);
- the gravity acceleration vector 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 2.2–2.5).

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) [1]. 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`). It also containts 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.

Table 2.1: The Link class data

Kinematic parameters		Inertial parameters	
`int`	`joint_type`	`Real`	`m`
`Real`	`theta, d, a, alpha`	`ColumnVector`	`r`
`Matrix`	`R,`	`Matrix`	`I`
`ColumnVector`	`p`		

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

$$\boldsymbol{R} = \begin{bmatrix} \cos\theta & -\cos\alpha\sin\theta & \sin\alpha\sin\theta \\ \sin\theta & \cos\alpha\cos\theta & -\sin\alpha\cos\theta \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix} \tag{2.16}$$

$$\boldsymbol{p} = \begin{bmatrix} a\cos\theta \\ a\sin\theta \\ d \end{bmatrix} \tag{2.17}$$

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:

$$\boldsymbol{T} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{p} \\ 0 & 1 \end{bmatrix} \tag{2.18}$$

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 [2]). The elements (3,2) and (3,3) of $\boldsymbol{R}$ provide storage for $\cos\alpha$ and $\sin\alpha$ which are computed only once. 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.

2.2.1 Robot object initialization

The Robot class provides a default constructor that creates a 1 dof robot. A $n_{dof} \times 5$ matrix containing the kinematic parameters can be supplied uppon initialization. Also, a $n_{dof} \times 15$ matrix containing the kinematic and inertial parameters can be supplied. A filename can also be used (BEWARE, this constructor is still under development). See the file `rr_dat.txt` for an example. 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	d	Denavit-Hartenberg parameter
5	α	Denavit-Hartenberg parameter
6	m	mass of the link
7	c_x	center of mass along axis x
8	c_y	center of mass along axis y
9	c_z	center of mass along axis z
10	I_{xx}	element xx of the inertia tensor matrix
11	I_{xy}	element xy of the inertia tensor matrix
12	I_{xz}	element xz of the inertia tensor matrix
13	I_{yy}	element yy of the inertia tensor matrix
14	I_{yz}	element yz of the inertia tensor matrix
15	I_{zz}	element zz of the inertia tensor matrix

constructors

Syntax

```
Robot(int ndof=1);
Robot(const Matrix & dhinit);
Robot(char * filename);

Robot(Robot & x);

Robot & operator=(Robot & x);
```

Description

Robot object constructors, copy constructor and equal operator.

Return Value

None

get_q

Syntax

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

Description

This function returns a column vector containing the joint variables when called with no argument. It returns the scalar value of the i^{th} joint variable when called with an integer argument.

Return Value

`ColumnVector` or `Real`

set_q

Syntax

```
void set_q(const ColumnVector & q);
void set_q(const Matrix & q);
void set_q(Real q, int i);
```

Description

This function sets the joint variables or the i^{th} joint variable to `q`.

Return Value

None

2.2.2 Kinematics

The forward kinematic model defines the relation:

$$ {}^0\boldsymbol{T}_n \;=\; \boldsymbol{G}(\boldsymbol{q}) \tag{2.19} $$

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

$$ \boldsymbol{q} \;=\; \boldsymbol{G}^{-1}({}^0\boldsymbol{T}_n) \tag{2.20} $$

In general, this equation allows multiple solutions.

inv_kin

Syntax

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

Description

The inverse kinematic model is computed using a Newton-Raphson technique. If `mj == 0`, it is based on the following [3]:

$$
\begin{aligned}
{}^0\boldsymbol{T}_n(\boldsymbol{q}^*) &= {}^0\boldsymbol{T}_n(\boldsymbol{q}+\delta\boldsymbol{q}) \approx {}^0\boldsymbol{T}_n(\boldsymbol{q})\delta\boldsymbol{T}(\delta\boldsymbol{q}) = \boldsymbol{T}_{obj} && (2.21)\\
\delta\boldsymbol{T}(\delta\boldsymbol{q}) &= ({}^0\boldsymbol{T}_n(\boldsymbol{q}))^{-1}\boldsymbol{T}_{obj} = \boldsymbol{I}+\boldsymbol{\Delta} && (2.22)\\
\boldsymbol{\Delta} &= \begin{bmatrix} 0 & -\delta_z & \delta_y & d_x \\ \delta_z & 0 & -\delta_x & d_y \\ -\delta_y & \delta_x & 0 & d_z \\ 0 & 0 & 0 & 0 \end{bmatrix} && (2.23)\\
{}^n\delta\boldsymbol{\chi} &= \begin{bmatrix} d_x & d_y & d_z & \delta_x & \delta_y & \delta_z \end{bmatrix}^T && (2.24)\\
{}^n\delta\boldsymbol{\chi} &\approx {}^n\boldsymbol{J}(\boldsymbol{q})\delta\boldsymbol{q} && (2.25)
\end{aligned}
$$

If `mj == 1`, it is based on the following Taylor expansion [3, 4]:

$$
{}^0\boldsymbol{T}_n(\boldsymbol{q}^*) = {}^0\boldsymbol{T}_n(\boldsymbol{q}+\delta\boldsymbol{q}) \approx {}^0\boldsymbol{T}_n(\boldsymbol{q}) + \sum_{i=1}^{n} \frac{\partial {}^0\boldsymbol{T}_n}{\partial q_i}\delta q_i \qquad (2.26)
$$

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.

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`

jacobian

Syntax

```
ReturnMatrix jacobian(int ref=0);
```

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:

$$^{i}\dot{\boldsymbol{\chi}} = {}^{i}\boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} \tag{2.27}$$

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

$$^{i}\delta\boldsymbol{\chi} \approx {}^{i}\boldsymbol{J}(\boldsymbol{q})\delta\boldsymbol{q} \tag{2.28}$$

The manipulation Jacobian expressed in the base frame is given by (see [5])

$$^{0}\boldsymbol{J}(\boldsymbol{q}) = \left[\begin{array}{cccc} {}^{0}\boldsymbol{J}_1(\boldsymbol{q}) & {}^{0}\boldsymbol{J}_2(\boldsymbol{q}) & \cdots & {}^{0}\boldsymbol{J}_n(\boldsymbol{q}) \end{array} \right] \tag{2.29}$$

with

$$^{0}\boldsymbol{J}_i(\boldsymbol{q}) = \left[\begin{array}{c} \boldsymbol{z}_i \times {}^{i-1}\boldsymbol{p}_n \\ \boldsymbol{z}_i \end{array} \right] \text{ for a revolute joint} \tag{2.30}$$

$$^{0}\boldsymbol{J}_i(\boldsymbol{q}) = \left[\begin{array}{c} \boldsymbol{z}_i \\ 0 \end{array} \right] \text{ for a prismatic joint} \tag{2.31}$$

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 is given by

$$^{i}\boldsymbol{J}(\boldsymbol{q}) = \left[\begin{array}{cc} ({}^{0}\boldsymbol{R}_i)^T & 0 \\ 0 & ({}^{0}\boldsymbol{R}_i)^T \end{array} \right] {}^{0}\boldsymbol{J}(\boldsymbol{q}) \tag{2.32}$$

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

Return Value

`Matrix`

kine

Syntax

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

Description

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

$$
\begin{aligned}
{}^0\boldsymbol{R}_i &= {}^0\boldsymbol{R}_{i-1}{}^{i-1}\boldsymbol{R}_i && (2.33)\\
{}^0\boldsymbol{p}_i &= {}^0\boldsymbol{p}_{i-1} + {}^0\boldsymbol{R}_{i-1}\boldsymbol{p}_i && (2.34)
\end{aligned}
$$

where

$$
{}^0\boldsymbol{T}_i = \begin{bmatrix} {}^0\boldsymbol{R}_i & {}^0\boldsymbol{p}_i \\ 0 & 1 \end{bmatrix} \qquad (2.35)
$$

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)

dTdqi

Syntax

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

Description

This function computes the partial derivatives:

$$\frac{\partial^0 \boldsymbol{T}_n}{\partial q_i} = {}^0\boldsymbol{T}_{i-1}\boldsymbol{Q}_i\,{}^{i-1}\boldsymbol{T}_n \tag{2.36}$$

with

$$\boldsymbol{Q}_i = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \text{ for a revolute joint} \tag{2.37}$$

$$\boldsymbol{Q}_i = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \text{ for a prismatic joint} \tag{2.38}$$

Return Value

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

2.2.3 Dynamics

The robotics manipulator dynamic model is given by (see appendix A)

$$\boldsymbol{\tau} = \boldsymbol{D}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}}) + \boldsymbol{G}(\boldsymbol{q}) \tag{2.39}$$

acceleration

Syntax

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

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 [6] presented methods to compute the inverse dynamics. A simplified RNE version computing

$$\boldsymbol{\tau} = \boldsymbol{D}(\boldsymbol{q})\ddot{\boldsymbol{q}} \tag{2.40}$$

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 :

$$\ddot{\boldsymbol{q}} = \boldsymbol{D}^{-1}(\boldsymbol{q})\left[\boldsymbol{\tau} - \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}}) - \boldsymbol{G}(\boldsymbol{q})\right] \tag{2.41}$$

Return Value

`ColumnVector`

inertia

Syntax

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

Description

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

$$\boldsymbol{\tau} = \boldsymbol{D}(\boldsymbol{q})\ddot{\boldsymbol{q}} \tag{2.42}$$

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

Return Value

`Matrix`

torque

Syntax

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

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 [7, 8] used to solve this problem (see appendix A).

Return Value

`ColumnVector`

torque_novelocity

Syntax

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

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`

2.2.4 Linearized dynamics

Murray and Neuman [8] have developed an efficient recursive linearized Newton-Euler formulation that can be used to compute (see appendix A)

$$\delta\boldsymbol{\tau} \;=\; \boldsymbol{D}(\boldsymbol{q})\delta\ddot{\boldsymbol{q}} + \boldsymbol{S_1}(\boldsymbol{q},\dot{\boldsymbol{q}})\delta\dot{\boldsymbol{q}} + \boldsymbol{S_2}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}})\delta\boldsymbol{q} \tag{2.43}$$

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

$$\delta\boldsymbol{\tau} \;=\; \boldsymbol{D}(\boldsymbol{q})\delta\ddot{\boldsymbol{q}} + \boldsymbol{S_1}(\boldsymbol{q},\dot{\boldsymbol{q}})\delta\dot{\boldsymbol{q}} + \boldsymbol{S_2}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}})\delta\boldsymbol{q} \tag{2.44}$$

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

$$\boldsymbol{S}_2(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}})\delta\boldsymbol{q} \tag{2.45}$$

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

$$\boldsymbol{S}_1(\boldsymbol{q},\dot{\boldsymbol{q}})\delta\dot{\boldsymbol{q}} \tag{2.46}$$

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

$$\frac{\partial \boldsymbol{\tau}}{\partial \boldsymbol{q}} = \boldsymbol{S_2}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}}) \tag{2.47}$$

Return Value

`Matrix`

dtau_dqp

Syntax

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

Description

This function computes

$$\frac{\partial \boldsymbol{\tau}}{\partial \dot{\boldsymbol{q}}} = \boldsymbol{S_1}(\boldsymbol{q},\dot{\boldsymbol{q}}) \tag{2.48}$$

Return Value

`Matrix`

2.3 Graphics

Graphics are provided through calls to the `gnuplot` [1] software. Instances of the class `Plot2d` are used to generate the data and command files required by the call to `gnuplot`.

Plot2d object initialization

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

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

[1] `gnuplot is freely available from the following location:`
`http://www.cs.dartmouth.edu/gnuplot_info.html`

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

2.4 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

$$\dot{\boldsymbol{x}} \;=\; \boldsymbol{f}(\boldsymbol{x}(t), t) \tag{2.49}$$

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

$$\left[\; t_0 \quad t_1 \quad \cdots \quad t_{nsteps} \;\right] \tag{2.50}$$

`xout` as

$$\left[\; \boldsymbol{x}_0 \quad \boldsymbol{x}_1 \quad \cdots \quad \boldsymbol{x}_{nsteps} \;\right] \tag{2.51}$$

`xo` as $\boldsymbol{x}_{nsteps}$, `nok` and `nbad` to the number of good and bad steps taken. The function `odeint` is adapted from [9].

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

$$\dot{\boldsymbol{x}} \;=\; \boldsymbol{f}(\boldsymbol{x}(t), t) \tag{2.52}$$

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

$$\left[\; t_0 \quad t_1 \quad \cdots \quad t_{nsteps} \;\right] \tag{2.53}$$

and `xout` as

$$\left[\; \boldsymbol{x}_0 \quad \boldsymbol{x}_1 \quad \cdots \quad \boldsymbol{x}_{nsteps} \;\right] \tag{2.54}$$

Return Value

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

vec_x_prod

Syntax

```
ReturnMatrix vec_x_prod(const ColumnVector & x,
                        const ColumnVector & y);
```

Description

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

Return Value

`ColumnVector`

2.5 Summary of functions

Table 2.2: Homogeneous transforms

Homogeneous Transforms	
eulzxz	transform of Euler angles
ieulzxz	Eulers 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 segement
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: Plot2d class member functions

Graphics	
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

Table 2.4: Robot class member functions

Joint Variables	
get_q	get the robot joint variables
set_q	set the robot joint variables
Robot Kinematics	
inv_kin	inverse kinematics
jacobian	robot jacobian
kine	forward kinematics
dTdqi	partial derivartive of forward kinematics
Robot Dynamics	
acceleration	forward dynamics
inertia	robot inertia matrix
torque	inverse dynamics
torque_novelocity	inverse dynamics without velocity and gravity
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 2.5: Miscellaneous

Miscellaneous	
odeint	adaptive step size Runge-Kutta integrator
Runge_Kutta4	fixed step size 4^{th} order Runge-Kutta integrator
vec_x_prod	vector cross product

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 assignements and homework. You can use your Web browser to send comments or bug report with the URL:

`http://www.ind2.polymtl.ca/ROBOOP`.

If you don't have acces 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.ind2.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 [10] 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 acknowledgements

I would like to thank Robert Davies for making is `NEWMAT09` library available.

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

Chapter 5

Future developments

In future releases, we hope to include the following:

- functions for basic control laws (computed torque method [7], sliding modes, etc);
- 3D graphics functions using `gnuplot`;
- functions dealing quaternions;
- a class using the modified D-H formalism [11];
- make files for other compilers.

Bibliography

[1] 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.

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

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

[4] J. J. Uicker, J. Denavit, and R. S. Hartenberg, "An iterative method for the displacement analysis of spatial mechanisms", *ASME Jour. of Applied Mechanics*, vol. 31, pp. 309–316, June 1964.

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

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

Appendix A

Recursive Newton-Euler algorithms

In order to apply the RNE as presented in [8], let us define the following variables (referenced in the i^{th} coordinate frame if applicable):

- σ_i is the joint type; $\sigma_i = 0$ for a revolute joint and $\sigma_i = 1$ 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 $\dot{\boldsymbol{v}}_0 = -\boldsymbol{g}$.

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

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

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

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

A.2 Recursive linearized Newton-Euler formulation

With

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

one can use the following

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

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

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

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

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

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

$$\delta \boldsymbol{F}_i = m_i \delta \dot{\boldsymbol{v}}_{ci} \tag{A.16}$$

$$\delta \boldsymbol{N}_i = \boldsymbol{I}_{ci} \delta \dot{\omega}_i + \delta \omega_i \times (\boldsymbol{I}_{ci} \omega_i) + \omega_i \times (\boldsymbol{I}_{ci} \delta \omega_i) \tag{A.17}$$

$$\delta \boldsymbol{f}_i = \boldsymbol{R}_{i+1} [\delta \boldsymbol{f}_{i+1}] + \delta \boldsymbol{F}_i + \sigma_{i+1} \boldsymbol{Q} \boldsymbol{R}_{i+1} [\boldsymbol{f}_{i+1}] \delta \theta_{i+1} \tag{A.18}$$

$$\begin{aligned} \delta \boldsymbol{n}_i = {} & \boldsymbol{R}_{i+1} [\delta \boldsymbol{n}_{i+1}] + \delta \boldsymbol{N}_i + \boldsymbol{p}_i \times \delta \boldsymbol{f}_i + \boldsymbol{r}_i \times \delta \boldsymbol{F}_i \\ & + (1 - \sigma_i)(\boldsymbol{p}_{di} \times \boldsymbol{f}_i) \delta d_i + \sigma_{i+1} \boldsymbol{Q} \boldsymbol{R}_{i+1} [\boldsymbol{n}_{i+1}] \delta \theta_{i+1} \end{aligned} \tag{A.19}$$

$$\begin{aligned} \delta \tau_i = {} & \sigma_i [\delta \boldsymbol{n}_i^T (\boldsymbol{R}_i^T \boldsymbol{z}_0) - \boldsymbol{n}_i^T (\boldsymbol{R}_i^T \boldsymbol{Q} \boldsymbol{z}_0) \delta \theta_i] \\ & + (1 - \sigma_i) [\delta \boldsymbol{f}_i^T (\boldsymbol{R}_i^T \boldsymbol{z}_0)] \end{aligned} \tag{A.20}$$

Appendix B

GNU Library General Public License

Content of the file `GNUlgpl.txt`.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Library 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 compile 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) 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.

   c) 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.

   d) 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 source code 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 to 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 Library 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