# 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

## **Contents**

2.13	The Control_Select class
2.14	The Stewart class 94
2.15	The IO_matrix_file class
2.16	Graphics
2.17	Config class
2.18	Miscellaneous
2.19	Summary of functions

 $3 \quad \text{Reorting bugs(t)} \\ 0.145467(,) \\ -383.427(c) \\ -0.391643(on) \\ 32.5153(t) \\ 0.0649008(r) \\ 0.055949(i) \\ 0.165609(colors) \\ 0.0649008(r) \\ 0.064908(r) \\ 0.0649008(r) \\ 0.064008(r) \\ 0.0649008(r) \\ 0.064008(r) \\ 0.0640$ 

make

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

•		

- Removed class ml i nk. DH and modified DH pagametegs age now included in l i nk.
- Added kine\_pd().
- Cgeated a new torque member function that allowed to have load on last link.
- · Fixed bug in modified DH dynamics.
- Added a class Quaterni on.
- Added the pgoggamrtest to compage gesults with Peter Cogke MATLAB toolbox.
- Added member function set\_plot2d to genegate plots using the Plot2d class.
- Added utility class IO\_matrix\_file dealing with data files (not documented yet).

e  $^{\frac{4}{3}}$  on , (2002/08/09) Moved the aggays of Col umnVector to the con-

 $\dagger_{\mathbf{o}}$  ce the ROBOOP proTram source directory ConfTuratcton file for CMake

CMakeLists txt

header file robot

header file for CLIK clik h

confi g h configuration files directory

con pd\_

#### irotk

 $\mathbf{yn}_{\mathbf{t}}$ 

ReturnMatrix irotk(const Matrix & R);

Given a homogeneous transform matrix  $\ensuremath{\mathsf{R}}\xspace,$  this function returns a column vector

k (2.5)

with k a unit vector such that the  $3 \times 3$  rotation bloc of the matrix

$$Rot(k, ) (2.6)$$

is equal to the  $3\times 3$  rotation bloc of the matrix R.

 $\mathbf{e}_{t}$  n  $\mathbf{e}$ 

ColumnVector.

$$yn_{\ell}$$
 (2.7)

Yn: (2.7)
ReturnMatrix irpy(const Matrix & R);

of the matrix

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

#### rotk

$$yn_{\xi}$$
 ReturnMatrix rotk(const Real theta, const ColumnVector & k); 
$$\mathbf{De^{\xi}c} \wedge_{\xi} \mathbf{\acute{o}n}$$

This function returns the matrix of a rotation of an angle  ${\tt theta}$  around the vector

```
rpy
  yn;
ReturnMatrix rpy(constColumnVector&a);
Dedcedcedcolumn
Giv a
```

## rotx, roty, rotz

 $\mathbf{yn}_{\mathbf{t}}$ 

ReturnMatrix rotx(const Real alpha);

## 2.2 The Quaterni on class

The Quaterni on

#### operators

```
\mathbf{yn}_{\mathbf{t}}
               operator+(const Quaternion & q)const;
  Quaterni on
               operator-(const Quaternion & q)const;
  Quaterni on
               operator*(const Quaternion & q)const;
  Quaterni on
               operator*(const ColumnVector & vec)const;
  Quaterni on
               operator*(constReal c)const;
  Quaterni on
               operator/(const Quaternion & q)const;
  Quaterni on
                operator/(constReal c)const;
  Quaterni on
 Defc fon
tRT
```

## conjugate and inverse

 $\mathbf{yn}_{\mathfrak{t}}$ 

### quaternion time derivative

$$\mathbf{yn}_{\mathbf{t}}$$

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

The quaternion time de&ivative is obtain f om the quate&nio

### unit and norm

#### **Rotation matrices**

### Omega,

 $\mathbf{yn}_{\mathbf{t}}$ 

#### Squad

 $\mathbf{yn}_{\mathbf{t}}$ 

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

Defc fon

Squad stands for Spherical Cubic Interpolation. Squad is not a member

### Squad\_prime

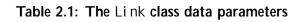
```
\mathbf{yn}_{\mathbf{t}}
```

Quaternion Squad\_prime(const Quaternion & p, const Quaternion & a, const Quaternion & b, const Quaternion & q, const Real t);

Detc ' ton

# 2.3 The Robot and mRobot classes

The Robot and mRobot



#### constructors

# get\_q, get\_qp, get\_qpp

 $\mathbf{yn}_{\mathfrak{t}}$ 

ReturnMatrix get\_q(void);

#### inv\_kin

$$\mathbf{yn}_{\mathbf{t}}$$

The inerse kinematic model is computed using a Newton-Raphs on technique. If mj = 0, it is ased on the ollowing [6]:

$${}^{0}T_{n}(q) = {}^{0}T_{n}(q + q) {}^{0}T_{n}(q) T(q) = T_{oj}$$
 (2.35)

$$T(q) = (^{0}T_{n}(q))^{-1}T_{0j} = I \triangle$$
 (2.36)



### inv\_kin\_

### jacobian\_dot

```
yn;
ReturnMatrix jacobian_dot(const int ref=0);
Defc / fon
```

The manipulator Jacobian time derivative can be used to compute the end

### jacobian\_DLS\_inv

 $\mathbf{yn}_{\mathfrak{t}}$ 

### 2.3.3 Dynamics

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

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

### inertia

$$\mathbf{yn}_{\mathbf{t}}$$

ReturnMatrix inertia(const ColumnVector & q);

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

$$= D(q \ddot{q}$$
 (2.69)

# torque\_n**dy**eloci

 $\mathbf{yn}_{\mathfrak{t}}$ 

### G and C

#### 2.3.4 Linearized dynamics

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

#### dqp\_torque

This function computes

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

et n

None (torque and dtorque

#### dtau\_dq

$$\mathbf{yn}_{\mathbf{t}}$$

ReturnMatrix dtau\_dq(const ColumnVector & q, const ColumnVector & qp, const ColumnVector & qpp);

 $\mathbf{De}^{\dagger}\mathbf{c}$  ' ton

#### This function computes

$$\frac{\phantom{a}}{\mathbf{q}} = \mathbf{S}_2(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}}) \tag{2.74}$$

et n

Matrix

### $dtau\_dqp$

```
\mathbf{yn}_{\{} ReturnMatrix dtau_dqp(const ColumnVector & q, const ColumnVector & qp);
```

### perturb\_robot

_		

2.5 The Spl_path class  Spl_path uses three instances of the class Spl_Cubic f986(28(21(o)0.0492351rc)-3120549(p)0.32898(a)0.
Spi_path uses three instances of the class Spi_Cubic 1700(20(21(0)0.04723511C)-3120347(p)0.32676(a)0.

2.7 The Traj ectory\_Sel ect class

# set\_trajectory

```
\mathbf{yn}_{\text{f}} void set_trajectory(const string & filename);
```

## torque\_cmd

 $\mathbf{yn}_{\ell}$ 

ReturnMatrix torque\_cmd(Robot\_basic & robot, const ColumnVector & qd, const ColumnVector & qpd);

# **2.10 The** Computed\_torque\_method class

The Computed\_torque\_method class deals with the well known computed

## torque\_cmd

 $\mathbf{yn}_{\ell}$ 

ReturnMatrix torque\_cmd(Robot\_basic & robot, const ColumnVector & qd, const ColumnVector & qpd);

```
K_d,\,K_p \mathbf{y}\mathbf{n}_{\{\}} short set_Kp(const Diagonal Matrix & Kp); short set_Kd(const Diagonal Matrix & Kd); \mathbf{De}^{\{\}}\mathbf{c} \triangleq \{\mathbf{o}\mathbf{n}\}
```

These functions sets the joint position error gain matrix,  $\boldsymbol{K}_{\boldsymbol{p}}$ 

## torque

```
\begin{array}{c} \textbf{K}_{pp}, \ \textbf{K}_{vp}, \ \textbf{K}_{po}, \ \textbf{K}_{vo} \\ \textbf{yn}_{\xi} \\ \\ \text{void set\_Kpp(const double Kpp);} \\ \text{void set\_Kpp(const double Kvp);} \\ \text{void set\_Kpo(const double Kpo);} \\ \text{void set\_Kvo(const double Kvo);} \\ \\ \textbf{De}^{\sharp} \textbf{c} & \quad & \quad & \quad & \quad & \quad & \\ \end{array}
```

#### control

 $\mathbf{yn}_{\mathbf{t}}$ 

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

Defc fon

# $get\_dof$

```
yn;
int get_dof();
Defc / fon
```

This function return the degree of185(f)812(f)0.304362(r)-0.210368(e)-0.234986(e)-0.234986(d)0.331218

#### set\_control

$$\mathbf{y}\mathbf{n}_{\xi}$$
 void set\_control(const string & filename);

This function set the active controller.

e<sub>t</sub> n e

None

# 2.14 The Stewart class

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

#### read

```
yn_{\{} short read(const vector<Matrix> & data); short read(const vector<Matrix> & data, const vector<string> & data_title);
```

### addcommand

```
\mathbf{y}\mathbf{n}_{\xi} void addcommand(const char * gcom); \mathbf{De^{j}c} \wedge \mathbf{t^{'}on}
```

#### addcurve

```
\mathbf{yn}_{\{\}} void addcurve(const Matrix & data, const char * label = "",
```

## gnuplot

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

e<sub>t</sub> n e

None

#### settitle

```
yn;
void settitle(const char * t);
Defc / fon
```

This function sets the title of the graph to the strin(n)148210368(i)-0.248413(n)(r)57Tf Td2584.369(t)

## setxlabel

yn; void setxlabel (const char \* t); 
$$\mathbf{De^{j}c} \stackrel{\text{?}}{\leftarrow} \stackrel{\text{?}}{\leftarrow} \mathbf{on}$$
 
$$t.$$
 
$$\mathbf{e}_{\ell} \quad n = \mathbf{e}$$
 None

# setylabel

 $\mathbf{yn}_{\mathbf{t}}$ 

#### set\_plot2d

## Reading and writing

```
\label{eq:conf} yn_{\xi} short read_conf(); short write_conf(const string filename, const string file_title, const int space_between the configuration of t
```

The member function read

\_conf reads a configuration file (specified by con-structor). The membeonsu

#### select

 $\mathbf{yn}_{t}$ 

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

 $\mathbf{e}_{t}$   $\mathbf{n}$   $\mathbf{e}$ 

Status, as a short int.

0 successful

SECTION\_OR\_PARAMETER\_DOES\_NOT\_EXIST

#### add

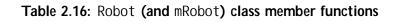
### x\_prod\_matrix

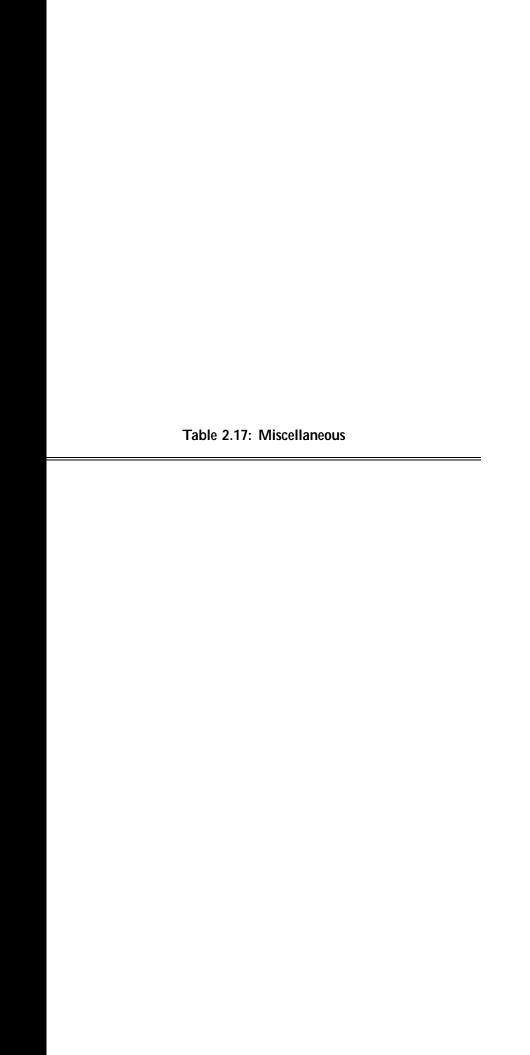
$$\begin{array}{l} \mathbf{y}\mathbf{n}_{\xi} \\ \text{ReturnMatrix x\_prod\_matrix(const ColumnVector \& x);} \\ \mathbf{De^{j}c} \wedge {_{\xi}}^{'}\mathbf{on} \end{array}$$

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

Table 2.3: Quaternion class member functions

<sub>{</sub> e n on o	
+, -, *, /, =	





SAMPLE CODE THAT MAKE THE BUG APPARENT:

Chapter 4

**Credits and** 

## Chapter 5

## **Bibliography**

[1] Jack C. K. Chou, "Quaternion kinematic and dynamic di erential eq0.21036742(a)0.0492351(-)]TJ

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

## 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<sup>th</sup> coordinate frame if applicable):

Initialize: 
$$f_{n+1} = n_{n+1} = 0$$
.

$$\dot{v}_{ci} = \bar{\bar{v}}$$

$$F_i = m_i v_{ci}$$
 (A.16)  
 $N_i = I_{ci} \dot{i} +$ 

# Appendix B Recursive Newton-Euler

• Bapkın/almi titæfiz pioms<sub>1</sub> for

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

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 lan-

though third parties are not compelled to copy the source along with the object code.

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

Also add information on how to contact you by electronic cd p .0492351(c)-per m.0492503541(i)-0.248413(l)-0.247294(.)-0.248413]TJ 16.9289-13.5539Td [(Y)83.4109(o)0.0492351(u)-248413]TJ 16.9289-13.5539Td