ROBOOP, A Robotics Object Oriented Package in C++
Reference Manual

1.31

Generated by Doxygen 1.5.1

Thu Dec 14 08:52:17 2006

Contents

Chapter 1

ROBOOP, A Robotics Object Oriented Package in C++ Hierarchical Index

1.1 ROBOOP, A Robotics Object Oriented Package in C++ Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

ROBOOP, A Robotics Object Oriented Package in C++ Class Index

2.1 ROBOOP, A Robotics Object Oriented Package in C++ Class List

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

Chapter 3

ROBOOP, A Robotics Object Oriented Package in C++ File Index

3.1 ROBOOP, A Robotics Object Oriented Package in C++ File List

Here is a list of all documented files with brief descriptions:

Chapter 4

ROBOOP, A Robotics Object Oriented Package in C++ Class Documentation

4.1 Clik Class Reference

`#include <clik.h>`

4.1.1 Detailed Description

Handle Closed Loop Inverse Kinematics scheme.

Definition at line 87 of file clik.h.

Public Member Functions

- Clik ()
- Clik (const Robot &robot_, const DiagonalMatrix &Kp_, const DiagonalMatrix &Ko_, const Real eps_=0.04, const Real lambda_max_=0.04, const Real dt=1.0)

  Constructor.

- Clik (const mRobot &mrobot_, const DiagonalMatrix &Kp_, const DiagonalMatrix &Ko_, const Real eps_=0.04, const Real lambda_max_=0.04, const Real dt=1.0)

  Constructor.

- Clik (const mRobot_min_para &mrobot_min_para_, const DiagonalMatrix &Kp_, const DiagonalMatrix &Ko_, const Real eps_=0.04, const Real lambda_max_=0.04, const Real dt=1.0)

  Constructor.

- Clik (const Clik &x)

  Copy constructor.

- ∼Clik ()
- Clik & operator= (const Clik &x)

  Overload = operator.

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

  Obtain joints position and velocity.

Private Member Functions

- int endeff_pos_ori_err (const ColumnVector &pd, const ColumnVector &pddot, const Quaternion &qd, const ColumnVector &wd)

  Obtain end effector position and orientation error.

Private Attributes

- Real dt

  Time frame.

- Real eps

  Range of singular region in Jacobian DLS inverse.

- Real lambda_max

  Damping factor in Jacobian DLS inverse.

- short robot_type

  Robot type used.

- Robot robot

  Robot instance.

- mRobot mrobot

  mRobot instance.

- mRobot_min_para mrobot_min_para

  mRobot_min_para instance.

- DiagonalMatrix Kp

  Position error gain.

- DiagonalMatrix Ko

  Orientation error gain.

- ColumnVector q

  Clik joint position.

- ColumnVector qp

  Clik joint velocity.

- ColumnVector qp_prev

  Clik previous joint velocity.

- ColumnVector Kpep

  Kp times position error.

- ColumnVector Koe0Quat

  Ko times orientation error (quaternion vector part).

- ColumnVector v

  Quaternion vector part.

4.1.2 Member Function Documentation

4.1.2.1 void Clik::q_qdot (const Quaternion & *qd*, const ColumnVector & *pd*, const ColumnVector & *pdd*, const ColumnVector & *wd*, ColumnVector & *q_*, ColumnVector & *qp_*)

Obtain joints position and velocity.

Parameters:

qd,: Desired eff orientatio in base frame.

pd,: Desired eff position in base frame.

pdd,: Desired eff velocity in base frame.

wd,: Desired eff angular velocity in base frame.

q_,: Output joint position.

qp_,: Output joint velocity.

Definition at line 269 of file clik.cpp.

References CLICK_DH, CLICK_mDH, CLICK_mDH_min_para, dt, endeff_pos_ori_err(), eps, Integ_Trap(), Robot_basic::jacobian_DLS_inv(), Koe0Quat, Kpep, lambda_max, mrobot, mrobot_min_para, q, qp, qp_prev, robot, robot_type, Robot_basic::set_q(), and v.

4.1.2.2 int Clik::endeff_pos_ori_err (const ColumnVector & *pd*, const ColumnVector & *pdd*, const Quaternion & *qqqd*, const ColumnVector & *wd*) `[private]`

Obtain end effector position and orientation error.

Parameters:

pd,: Desired eff position in base frame.

pdd,: Desired eff velocity in base frame.

qqqd,: Desired eff orientation in base frame.

wd,: Desired eff angular velocity in base frame.

Definition at line 224 of file clik.cpp.

References CLICK_DH, CLICK_mDH, CLICK_mDH_min_para, Robot_basic::kine(), Ko, Koe0Quat, Kp, Kpep, mrobot, mrobot_min_para, q, robot, robot_type, Quaternion::s(), Robot_basic::set_q(), Quaternion::v(), and x_prod_matrix().

Referenced by q_qdot().

4.2 Computed_torque_method Class Reference

`#include <controller.h>`

4.2.1 Detailed Description

Computer torque method controller class.

The dynamic model of a robot manipulator can be expressed in joint space as

$$B(q)\ddot{q} + C(q,\dot{q})\dot{q} + D\dot{q} + g(q) = \tau - J^T(q)f$$

The driving torques can be expressed as

$$\tau = B(q)\big(\ddot{q}_d + K_d(\dot{q}_d - \dot{q}) + K_p(q_d - q)\big) + C(q,\dot{q})\dot{q} + D\dot{q} + g(q) + J^T(q)f$$

where K_p, K_d are diagonal positive definie matrix.

Definition at line 220 of file controller.h.

Public Member Functions

- Computed_torque_method (const short dof=1)

  Constructor.

- Computed_torque_method (const Robot_basic &robot, const DiagonalMatrix &Kp, const DiagonalMatrix &Kd)

  Constructor.

- ReturnMatrix torque_cmd (Robot_basic &robot, const ColumnVector &qd, const ColumnVector &qpd, const ColumnVector &qppd)

  Output torque.

- short set_Kd (const DiagonalMatrix &Kd)

  Assign the velocity error gain matrix $K_d(i,i)$.

- short set_Kp (const DiagonalMatrix &Kp)

  Assign the position error gain matrix $K_p(i,i)$.

Private Attributes

- int dof

  Degree of freedom.

- ColumnVector q

  Robot joints positions.

- ColumnVector qp

  Robot joints velocity.

- ColumnVector qpp

  Robot joints acceleration.

- ColumnVector zero3

  3×1 *zero vector.*

- DiagonalMatrix Kp

  Position error gain.

- DiagonalMatrix Kd

  Velocity error gain.

4.2.2 Member Function Documentation

4.2.2.1 short Computed_torque_method::set_Kd (const DiagonalMatrix & *Kd_*)

Assign the velocity error gain matrix $K_d(i,i)$.

Returns:

short: 0 or WRONG_SIZE if the matrix is not $dof \times dof$.

Definition at line 571 of file controller.cpp.

References dof, Kd, and WRONG_SIZE.

Referenced by Computed_torque_method().

4.2.2.2 short Computed_torque_method::set_Kp (const DiagonalMatrix & *Kp_*)

Assign the position error gain matrix $K_p(i,i)$.

Returns:

short: 0 or WRONG_SIZE if the matrix is not $dof \times dof$.

Definition at line 588 of file controller.cpp.

References dof, Kp, and WRONG_SIZE.

Referenced by Computed_torque_method().

4.3 Config Class Reference

`#include <config.h>`

4.3.1 Detailed Description

Handle configuration files.

Definition at line 97 of file config.h.

Public Member Functions

- Config (const bool bPrintErrorMessages=true)

  Constructor.

- short **read_conf** (std::ifstream &inconffile)
- void clear ()

  Clear the data buffer.

- void print ()

  Print the configuration data.

- bool **section_exists** (const std::string &section) const
- bool **parameter_exists** (const std::string &section, const std::string &parameter) const
- template<typename T> bool select (const std::string &section, const std::string &parameter, T &value) const

  Get a parameter data, of a certain section, into the string value.

- short **write_conf** (std::ofstream &outconffile, const std::string &file_title, const int space_between_column)
- template<typename T> bool add (const std::string &section, const std::string &parameter, const T &value)

  Added the value(string) of the parameter in the section in the configuration data. The functions will added the parameter and the section if it does not already exist.

Private Attributes

- Conf_data conf

  Data *store from/to configuration file.*

- bool bPrintErrorMessages

  Print error messages on stderr.

4.3.2 Member Function Documentation

4.3.2.1 template<typename T> bool Config::select (const std::string & *section*, const std::string & *parameter*, T & *value*) const `[inline]`

Get a parameter data, of a certain section, into the string value.

Returns:

> false if the data can not be found and true otherwise.

Definition at line 109 of file config.h.

References conf.

4.4 Control_Select Class Reference

`#include <control_select.h>`

4.4.1 Detailed Description

Select controller class.

This class contains an instance of each controller class. The active controller will be selected when reading a controller file. "type" value correspond to the active controller, ex:

- `type = NONE` : no controller selected
- `type = PD` : Proportional Derivative
- `type = CTM` : Computer Torque Method
- `type = RRA` : Resolved Rate Acceleration
- `type = IMP` : Impedance

Bellow is an exemple of RRA configuration file (more info on configuration file in config.h/cpp):

```
 [CONTROLLER]

type:    RESOLVED_RATE_ACCELERATION
dof:     6

 [GAINS]

Kvp:          500.0
Kpp:         5000.0
Kvo:          500.0
Kpo:         5000.0
```

Definition at line 100 of file control_select.h.

Public Member Functions

- Control_Select ()

  Constructor.

- **Control_Select** (const std::string &filename)
- int get_dof ()

  Return the degree of freedom.

- void **set_control** (const std::string &filename)

Public Attributes

- Proportional_Derivative pd
- Computed_torque_method ctm
- Resolved_acc rra
- Impedance impedance
- short type

  Type of controller: PD, CTM,...

- short space_type

  JOINT_SPACE or CARTESIAN_SPACE.

- std::string ControllerName

  Controller name.

Private Attributes

- int dof

  Degree of freedom.

4.5 Data Struct Reference

`#include <config.h>`

4.5.1 Detailed Description

Basic data element used in Config class.

Definition at line 87 of file config.h.

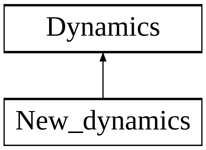
Public Attributes

- std::string section
- std::string parameter
- std::string value

4.6 Dynamics Class Reference

`#include <dynamics_sim.h>`

Inheritance diagram for Dynamics::



4.6.1 Detailed Description

Dynamics simulation handling class.

Definition at line 60 of file dynamics_sim.h.

Public Member Functions

- Dynamics (Robot_basic ∗robot_)

  Constructor.

- virtual ∼Dynamics ()
- void set_dof (Robot_basic ∗robot_)

  Set the degree of freedom.

- short set_controller (const Control_Select &x)

  Set the control variable from the Control_Select reference.

- short set_trajectory (const Trajectory_Select &x)

  Set the path_select variable from the Trajectory_Select reference.

- ReturnMatrix set_robot_on_first_point_of_splines ()

  Set the robot on first point of trajectory.

- void set_time_frame (const int nsteps)

  Set the number of iterations.

- void set_final_time (const double tf)

  Set the file time.

- void reset_time ()

  Set the simulation time to the inital time.

- void Runge_Kutta4_Real_time ()

  Runge Kutta solver for real time.

- void Runge_Kutta4 ()

  Runge Kutta solver.

- virtual void plot ()

  Virtual plot functions.

- ReturnMatrix xdot (const Matrix &xin)

  Obtain state derivative.

Static Public Member Functions

- static Dynamics ∗ Instance ()

  A pointer to Dynamics instance. Pointer is 0 if there is no instance (logic done in Constructor).

Public Attributes

- bool first_pass_Kutta

  First time in all Runge_Kutta4 functions.

- int ndof

  Degree of freedom.

- int dof_fix

  Degree of freedom + virtual link.

- int nsteps

  Numbers of iterations between.

- double h

  Runge Kutta temporary variable.

- double h2

  Runge Kutta temporary variable.

- double time

  Time during simulation.

- double to

  Initial simulation time.

- double tf

  Final time used in Runge_Kutta4_Real_time.

- double tf_cont

  Final time used in Runge_Kutta4.

- double dt

  Time frame.

- Matrix k1

  Runge Kutta temporary variable.

- Matrix k2

  Runge Kutta temporary variable.

- Matrix k3

  Runge Kutta temporary variable.

- Matrix k4

  Runge Kutta temporary variable.

- Matrix x

  Stated vector obtain in Runge Kutta functions.

- Matrix xd

  Statd vector derivative obtaint in xdot function.

- ColumnVector q

  Joints positions.

- ColumnVector qp

  Joints velocities.

- ColumnVector qpp

  Joints accelerations.

- ColumnVector qd

  Desired joints positions.

- ColumnVector qpd

  Desired joints velocities.

- ColumnVector qppd

  Desired joints accelerations.

- ColumnVector tau

  Controller output torque.

- ColumnVector pd

  Desired end effector cartesian position.

- ColumnVector ppd

  Desired end effector cartesian velocity.

- ColumnVector pppd

  Desired end effector cartesian acceleration.

- ColumnVector wd

  Desired end effector cartesian angular velocity.

- ColumnVector wpd

  Desired end effector cartesian angular acceleration.

- Quaternion quatd

  Desired orientation express by a quaternion.

- Control_Select control

  Instance of Control_Select class.

- Trajectory_Select path_select

  Instance of Trajectory_Select class.

- Robot_basic $*$ robot

  Pointer on Robot_basic class.

Static Public Attributes

- static Dynamics ∗ instance

  Static pointer on Dynamics class.

4.6.2 Member Function Documentation

4.6.2.1 void Dynamics::set_dof (Robot_basic ∗ *robot_*)

Set the degree of freedom.

Obtain the degree of freedom from Robot_basic pointer. Some vectors will be resize with new current dof value.

Definition at line 109 of file dynamics_sim.cpp.

References dof_fix, first_pass_Kutta, Robot_basic::get_dof(), Robot_basic::get_fix(), ndof, q, qd, qp, qpd, qpp, qppd, robot, and tau.

4.6.2.2 ReturnMatrix Dynamics::set_robot_on_first_point_of_splines ()

Set the robot on first point of trajectory.

Assigned the robot joints position to the first point of the trajectory if the latter is expressed in joint space, or assigned the robot joints position via inverse kinematics if the trajectory is expressed in cartesian space. The function return a message on the console if the format of the trajectory file is incorrect.

Definition at line 190 of file dynamics_sim.cpp.

References CARTESIAN_SPACE, Spl_path::get_final_time(), Robot_basic::get_q(), Robot_basic::inv_kin(), JOINT_SPACE, ndof, Spl_path::p_pdot(), Spl_path::p_pdot_pddot(), Trajectory_Select::path, Trajectory_Select::path_quat, path_select, pd, ppd, pppd, q, qd, qpd, Spl_Quaternion::quat_w(), quatd, Quaternion::R(), robot, Robot_basic::set_q(), tf_cont, Trajectory_Select::type, and wd.

Referenced by Runge_Kutta4(), and Runge_Kutta4_Real_time().

4.6.2.3 ReturnMatrix Dynamics::xdot (const Matrix & *x*)

Obtain state derivative.

Parameters:

x,: state vector (joint speed and joint velocity).

The controller torque is applied if any controller has been selected, then the joint acceleration is obtained.

Definition at line 252 of file dynamics_sim.cpp.

References Robot_basic::acceleration(), CARTESIAN_SPACE, control, Control_Select::ctm, CTM, dof_fix, dt, JOINT_SPACE, ndof, Spl_path::p_pdot(), Spl_path::p_pdot_pddot(), Trajectory_Select::path, Trajectory_Select::path_quat, path_select, pd, Control_Select::pd, PD, plot(), ppd, pppd, q, qd, qp, qpd, qpp, qppd, Spl_Quaternion::quat_w(), quatd, robot, Control_Select::rra, RRA, Robot_basic::set_q(), Robot_basic::set_qp(), tau, time, Resolved_acc::torque_cmd(), Computed_torque_method::torque_cmd(), Proportional_Derivative::torque_cmd(), Trajectory_Select::type, Control_Select::type, wd, wpd, and xd.

Referenced by Runge_Kutta4(), and Runge_Kutta4_Real_time().

4.7 GNUcurve Class Reference

`#include <gnugraph.h>`

4.7.1 Detailed Description

Object for one curve.

Definition at line 127 of file gnugraph.h.

Public Member Functions

- **GNUcurve** (const std::vector< double > &x, std::vector< double > &y, const std::string &label="", LineType_en enLineType=LINES)
- GNUcurve (void)

  Constructor.

- void dump (void)

  Method to dump the content of a curve to stdout.

Public Attributes

- std::vector< double > vdX
- std::vector< double > vdY
- std::string clabel

  string defining the curve label for the legend

- LineType_en enLineType

  Line type.

4.8 Impedance Class Reference

`#include <controller.h>`

4.8.1 Detailed Description

Impedance controller class.

The implemantation of the impedance controller is made of two section: the first one is the generation of a compliance trajectory and the second one used a position controller to ensure the end effector follow the compliance trajectory (We recommended to used the resolve acceleration controller scheme, implemented in the class Resolved_acc).

This class generate a compliance path given by the translational and the rotational impedance.

$$M_p\ddot{\tilde{p}} + D_p\dot{\tilde{p}} + K_p\tilde{p} = f$$

$$M_o\dot{\tilde{\omega}} + D_o\tilde{\omega} + K_o'\tilde{v} = n$$

where $\tilde{x} = x_c - x_d$ and v is the vector par of the quaternion representing the orientation error between the compliant and desired frame. The orientation error can also be express by rotation matrix, $\tilde{R} = R_d^T R_c$. The quaternion mathematics are implemented in the Quaternion class. The index $_c$ and $_d$ denote the compliance and the desired respectively.

The impedance parameters M_p, D_p, K_p, M_o, D_o and K_o are 3×3 diagonal positive definite matrix

Definition at line 91 of file controller.h.

Public Member Functions

- Impedance ()

  Constructor.

- Impedance (const Robot_basic &robot, const DiagonalMatrix &Mp_, const DiagonalMatrix &Dp_, const DiagonalMatrix &Kp_, const DiagonalMatrix &Mo_, const DiagonalMatrix &Do_, const DiagonalMatrix &Ko_)

  Constructor.

- short set_Mp (const DiagonalMatrix &Mp_)

  Assign the translational impedance inertia matrix M_p.

- short set_Mp (const Real MP_i, const short i)

  Assign the translational impedance inertia term $M_p(i, i)$.

- short set_Dp (const DiagonalMatrix &Dp_)

  Assign the translational impedance damping matrix D_p*.*

- short set_Dp (const Real Dp_i, const short i)

  Assign the translational impedance damping term $D_p(i,i)$*.*

- short set_Kp (const DiagonalMatrix &Kp_)

  Assign the translational impedance stifness matrix K_p*.*

- short set_Kp (const Real Kp_i, const short i)

  Assign the translational impedance stifness term $K_p(i,i)$*.*

- short set_Mo (const DiagonalMatrix &Mo_)

  Assign the rotational impedance inertia matrix M_o*.*

- short set_Mo (const Real Mo_i, const short i)

  Assign the rotational impedance inertia term $M_o(i,i)$*.*

- short set_Do (const DiagonalMatrix &Do_)

  Assign the rotational impedance damping matrix D_o*.*

- short set_Do (const Real Do_i, const short i)

  Assign the rotational impedance damping term $D_o(i,i)$*.*

- short set_Ko (const DiagonalMatrix &Ko_)

  Assign the rotational impedance stifness matrix K_o*.*

- short set_Ko (const Real Ko_i, const short i)

  Assign the rotational impedance stifness term $K_o(i,i)$*.*

- 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)

  Generation of a compliance trajectory.

Public Attributes

- Quaternion qc

  Compliant frame quaternion.

- Quaternion qcp

  Compliant frame quaternion derivative.

- Quaternion qcp_prev

  Previous value of qcp.

- Quaternion qcd

  Orientation error (betweem compliant and desired frame) quaternion.

- Quaternion quat

  Temporary quaternion.

- ColumnVector pc

  Compliant position.

- ColumnVector pcp

  Compliant velocity.

- ColumnVector pcpp

  Compliant acceleration.

- ColumnVector pcp_prev

  Previous value of pcp.

- ColumnVector pcpp_prev

  Previous value of pcpp.

- ColumnVector pcd

  Difference between pc and desired position.

- ColumnVector pcdp

  Difference between pcp and desired velocity.

- ColumnVector wc

  Compliant angular velocity.

- ColumnVector wcp

  Compliant angular acceleration.

- ColumnVector wcp_prev

  Previous value of wcp.

- ColumnVector wcd

  Difference between wc and desired angular velocity.

Private Attributes

- DiagonalMatrix Mp

  Translational impedance inertia matrix.

- DiagonalMatrix Dp

  Translational impedance damping matrix.

- DiagonalMatrix Kp

  Translational impedance stifness matrix.

- DiagonalMatrix Mo

  Rotational impedance inertia matrix.

- DiagonalMatrix Do

  Rotational impedance damping matrix.

- DiagonalMatrix Ko

  Rotational impedance stifness matrix.

- Matrix Ko_prime

  Modified rotational impedance stifness matrix.

4.8.2 Member Function Documentation

4.8.2.1 short Impedance::set_Mp (const DiagonalMatrix & *Mp_*)

Assign the translational impedance inertia matrix M_p.

Returns:

short: 0 or WRONG_SIZE if the matrix is not 3×3.

Definition at line 100 of file controller.cpp.

References Mp, and WRONG_SIZE.

Referenced by Impedance().

4.8.2.2 short Impedance::set_Mp (const Real *Mp_i*, const short *i*)

Assign the translational impedance inertia term $M_p(i, i)$.

Returns:

short: 0 or WRONG_SIZE if the matrix is not 3×3.

Definition at line 117 of file controller.cpp.

References Mp, and WRONG_SIZE.

4.8.2.3 short Impedance::set_Dp (const DiagonalMatrix & *Dp_*)

Assign the translational impedance damping matrix D_p.

Returns:

short: 0 or WRONG_SIZE if the matrix is not 3×3.

Definition at line 133 of file controller.cpp.

References Dp, and WRONG_SIZE.

Referenced by Impedance().

4.8.2.4 short Impedance::set_Dp (const Real *Dp_i*, const short *i*)

Assign the translational impedance damping term $D_p(i, i)$.

Returns:

short: 0 or WRONG_SIZE if the matrix is not 3×3.

Definition at line 150 of file controller.cpp.

References Dp, and WRONG_SIZE.

4.8.2.5 short Impedance::set_Kp (const DiagonalMatrix & *Kp_*)

Assign the translational impedance stifness matrix K_p.

Returns:

short: 0 or WRONG_SIZE if the matrix is not 3×3.

Definition at line 166 of file controller.cpp.

References Kp, and WRONG_SIZE.

Referenced by Impedance().

4.8.2.6 short Impedance::set_Kp (const Real *Kp_i*, const short *i*)

Assign the translational impedance stifness term $K_p(i, i)$.

Returns:

short: 0 or WRONG_SIZE if the matrix is not 3 × 3.

Definition at line 183 of file controller.cpp.

References Kp, and WRONG_SIZE.

4.8.2.7 short Impedance::set_Mo (const DiagonalMatrix & *Mo_*)

Assign the rotational impedance inertia matrix M_o.

Returns:

short: 0 or WRONG_SIZE if the matrix is not 3 × 3.

Definition at line 199 of file controller.cpp.

References Mo, and WRONG_SIZE.

Referenced by Impedance().

4.8.2.8 short Impedance::set_Mo (const Real *Mo_i*, const short *i*)

Assign the rotational impedance inertia term $M_o(i, i)$.

Returns:

short: 0 or WRONG_SIZE if the matrix is not 3 × 3.

Definition at line 216 of file controller.cpp.

References Mo, and WRONG_SIZE.

4.8.2.9 short Impedance::set_Do (const DiagonalMatrix & *Do_*)

Assign the rotational impedance damping matrix D_o.

Returns:

short: 0 or WRONG_SIZE if the matrix is not 3×3.

Definition at line 232 of file controller.cpp.

References Do, and WRONG_SIZE.

Referenced by Impedance().

4.8.2.10 short Impedance::set_Do (const Real *Do_i*, const short *i*)

Assign the rotational impedance damping term $D_o(i,i)$.

Returns:

short: 0 or WRONG_SIZE if the matrix is not 3×3.

Definition at line 249 of file controller.cpp.

References Do, and WRONG_SIZE.

4.8.2.11 short Impedance::set_Ko (const DiagonalMatrix & *Ko_*)

Assign the rotational impedance stifness matrix K_o.

Returns:

short: 0 or WRONG_SIZE if the matrix is not 3×3.

Definition at line 265 of file controller.cpp.

References Ko, and WRONG_SIZE.

Referenced by Impedance().

4.8.2.12 short Impedance::set_Ko (const Real *Ko_i*, const short *i*)

Assign the rotational impedance stifness term $K_o(i,i)$.

Returns:

short: 0 or WRONG_SIZE if the matrix is not 3×3.

Definition at line 282 of file controller.cpp.

References Ko, and WRONG_SIZE.

4.8.2.13 short Impedance::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*)

Generation of a compliance trajectory.

Parameters:

pdpp,: desired end effector acceleration.

pdp,: desired end effector velocity.

pd,: desired end effector position.

wdp,: desired end effector angular acceleration.

wd,: desired end effector angular velocity.

qd,: desired quaternion.

f,: end effector contact force.

n,: end effector contact moment.

dt,: time frame.

Returns:

short: 0 or WRONG_SIZE if one of the vector input is not 3×1.

The translational and rotational impedance equations are integrated, with input f and n to computed $\ddot{p}_c$ and $\dot{\omega}_c$, $\dot{p}_c$ and ω_c, and then p_c and q_c. The compliant quaternion q_c is obtained with the quaternion propagation equations (see Quaternion class).

The quaternion -q represents exactly the same rotation as the quaternion q. The temporay quaternion, quat, is quatd plus a sign correction. It is customary to choose the sign G on q1 so that q0.Gq1 $>$=0 (the angle between q0 ang Gq1 is acute). This choice avoids extra spinning caused by the interpolated rotations.

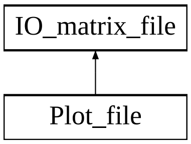
Definition at line 298 of file controller.cpp.

References BASE_FRAME, Do, Quaternion::dot(), Quaternion::dot_prod(), Dp, Quaternion::E(), Quaternion::i(), Integ_quat(), Integ_Trap(), Ko, Ko_prime, Kp, Mo, Mp, pc, pcd, pcdp, pcp, pcp_prev, pcpp, pcpp_prev, qc, qcd, qcp, qcp_prev, quat, Quaternion::v(), wc, wcd, wcp, wcp_prev, and WRONG_SIZE.

4.9 IO_matrix_file Class Reference

`#include <gnugraph.h>`

Inheritance diagram for IO_matrix_file::



4.9.1 Detailed Description

Read and write data at every iterations in a file.

Definition at line 201 of file gnugraph.h.

Public Member Functions

- **IO_matrix_file** (const std::string &filename)
- short **write** (const std::vector< Matrix > &data)
- short **write** (const std::vector< Matrix > &data, const std::vector< std::string > &title)
- short **read** (std::vector< Matrix > &data)
- short **read** (std::vector< Matrix > &data, std::vector< std::string > &title)
- short **read_all** (std::vector< Matrix > &data, std::vector< std::string > &data_title)

Private Attributes

- int position_read

  Position to read the file.

- int nb_iterations_write

  Number of iterations in writing mode.

- int nb_iterations_read

  Number of iterations in reading mode.

- int nb_element

  Number of elements to read or write.

- std::string filename

  File name.

4.10 Link Class Reference

`#include <robot.h>`

4.10.1 Detailed Description

Link definitions.

A n degree of freedom (dof) serial manipulator is composed of n links. This class describe the property of a link. A n dof robot has n instance of the class Link.

Definition at line 137 of file robot.h.

Public Member Functions

- Link (const int jt=0, const Real it=0.0, const Real id=0.0, const Real ia=0.0, const Real ial=0.0, const Real theta_min=-M_PI/2, const Real theta_max=M_PI/2, const Real it_off=0.0, const Real mass=1.0, const Real cmx=0.0, const Real cmy=0.0, const Real cmz=0.0, const Real ixx=0.0, const Real ixy=0.0, const Real ixz=0.0, const Real iyy=0.0, const Real iyz=0.0, const Real izz=0.0, const Real iIm=0.0, const Real iGr=0.0, const Real iB=0.0, const Real iCf=0.0, const bool dh=true, const bool min_inertial_para=false, const bool immobile=false)

    Constructor.

- ∼Link ()

    Destructor.

- void transform (const Real q)

    Set the rotation matrix R and the vector p.

- bool get_DH (void) const

    Return DH value.

- int get_joint_type (void) const

    Return the joint type.

- Real get_theta (void) const

    Return theta.

- Real get_d (void) const

    Return d.

- Real get_a (void) const

Return a.

- Real get_alpha (void) const

  Return alpha.

- Real get_q (void) const

  Return joint position (theta if joint type is rotoide, d otherwise).

- Real get_theta_min (void) const

  Return theta_min.

- Real get_theta_max (void) const

  Return theta_max.

- Real get_joint_offset (void) const

  Return joint_offset.

- ReturnMatrix get_mc (void)

  Return mc.

- ReturnMatrix get_r (void)

  Return r.

- ReturnMatrix get_p (void) const

  Return p.

- Real get_m (void) const

  Return m.

- Real get_Im (void) const

  Return Im.

- Real get_Gr (void) const

  Return Gr.

- Real get_B (void) const

  Return B.

- Real get_Cf (void) const

  Return Cf.

- ReturnMatrix get_I (void) const

Return I.

- bool get_immobile (void) const

  Return immobile.

- void set_m (const Real m_)

  Set m.

- void set_mc (const ColumnVector &mc_)

  Set mc.

- void set_r (const ColumnVector &r_)

  Set r.

- void set_Im (const Real Im_)

  Set Im.

- void set_B (const Real B_)

  Set B.

- void set_Cf (const Real Cf_)

  Set Cf.

- void set_I (const Matrix &I)

  Set I.

- void set_immobile (bool im)

  Set immobile.

Public Attributes

- Matrix R

  Orientation matrix of actual link w.r.t to previous link.

- Real qp

  Joint velocity.

- Real qpp

  Joint acceleration.

Private Attributes

- int joint_type

  Joint type.

- Real theta

  theta DH parameter.

- Real d

  d DH parameter.

- Real a

  a DH parameter.

- Real alpha

  alpha DH parameter.

- Real theta_min

  Min joint angle.

- Real theta_max

  Max joint angle.

- Real joint_offset

  Offset in joint angle (rotoide and prismatic).

- bool DH

  DH notation(true) or DH modified notation.

- bool min_para

  Minimum inertial parameter.

- ColumnVector r

  Position of center of mass w.r.t. link coordinate system (min_para=F).

- ColumnVector p

  Position vector of actual link w.r.t to previous link.

- Real m

  Mass of the link.

- Real Im

  Motor Inertia.

- Real Gr

  Gear Ratio.

- Real B

  Viscous coefficient.

- Real Cf

  Coulomb fiction coefficient.

- ColumnVector mc

  Mass × center of gravity (used if min_para = true).

- Matrix I

  Inertia matrix w.r.t. center of mass and link coordinate system orientation.

- bool immobile

  true if the joint is to be considered locked - ignored for inverse kinematics, but can still be reassigned through transform

Friends

- class Robot_basic
- class Robot
- class mRobot
- class mRobot_min_para

4.10.2 Member Function Documentation

4.10.2.1 Real Link::get_q (void) const

Return joint position (theta if joint type is rotoide, d otherwise).

The joint offset is removed from the value.

Definition at line 299 of file robot.cpp.

References d, joint_offset, joint_type, and theta.

4.11 LinkStewart Class Reference

`#include <stewart.h>`

4.11.1 Detailed Description

LinkStewart definitions.

A Stewart platform is composed 6 links. This class describe the proprities of each of the platform's link.

Definition at line 53 of file stewart.h.

Public Member Functions

- LinkStewart (const ColumnVector &InitLink, const Matrix wRp, const ColumnVector q)

  Constructor.

- LinkStewart (const LinkStewart &x)

  Copy constructor.

- LinkStewart ()

  Default Constructor.

- ~LinkStewart ()

  Destructor.

- const LinkStewart & operator= (const LinkStewart &x)
- void set_ap (const ColumnVector NewAp)

  Set the position vector of platform attachment point.

- void set_b (const ColumnVector Newb)

  Set the position vector of the base attachment point.

- void set_I1aa (const Real NewI1aa)

  Set the value of inertia along the coaxial axis of part 1.

- void set_I1nn (const Real NewI1nn)

  Set the value of inertia along the tangent axis of part 1.

- void set_I2aa (const Real NewI2aa)

  Set the value of inertia along the coaxial axis of part 2.

- void set_I2nn (const Real NewI2nn)

  Set the value of inertia along the tangent axis of part 2.

- void set_m1 (const Real Newm1)

  Set the mass of part 1.

- void set_m2 (const Real Newm2)

  Set the mass of part 2.

- void set_Lenght1 (const Real NewLenght1)

  Set the lenght between platform attachment point and center of mass of part 1.

- void set_Lenght2 (const Real NewLenght2)

  Set the lenght between base attachment point and center of mass of part 2.

- ReturnMatrix get_ap () const

  Return the position vector of platform attachement point.

- ReturnMatrix get_b () const

  Return the position vector of base attachement point.

- Real get_I1aa () const

  Return the value of inertia along the coaxial axis of part 1.

- Real get_I1nn () const

  Return the value of inertia along the tangent axis of part 1.

- Real get_I2aa () const

  Return the value of inertia along the coaxial axis of part 2.

- Real get_I2nn () const

  Return the value of inertia along the tangent axis of part 2.

- Real get_m1 () const

  Return the mass of part 1.

- Real get_m2 () const

  Return the mass of part 2.

- Real get_Lenght1 () const

  Return the lenght between platform attachment point and center of mass of part 1.

- Real get_Lenght2 () const

  Return the lenght between base attachment point and center of mass of part 2.

- void LTransform (const Matrix wRp, const ColumnVector q)

  Recalculate the link's parameters related to the platform position.

- void d_LTransform (const ColumnVector dq, const ColumnVector Omega, const Real dl, const Real ddl)

  Recalculate the link's parameters related to the platform speed.

- void dd_LTransform (const ColumnVector ddq, const ColumnVector Omega, const ColumnVector Alpha, const Real dl, const Real ddl)

  Recalculate the link's parameters related to the platform acceleration.

- void tau_LTransform (const Real dl, const Real ddl, const Real Gravity)

  Recalculate the link's parameters related to the platform dynamics.

- ReturnMatrix Find_UnitV ()

  Return the unit vector of the link direction.

- ReturnMatrix Find_a (const Matrix _wRp, const ColumnVector _q)

  Return the position of the attachment point on the platform.

- ReturnMatrix Find_da (const ColumnVector dq, const ColumnVector Omega)

  Return the speed of the attachment point of the link on the platform.

- ReturnMatrix Find_dda (const ColumnVector ddq, const ColumnVector Omega, const ColumnVector Alpha)

  Return the acceleration of the attachment point of the link on the platform.

- Real Find_Lenght ()

  Return the lenght of the link.

- ReturnMatrix Find_VctU ()

  Return the unit vector of the universal joint along the first axis of the fixed revolute joint.

- ReturnMatrix Find_VctV ()

  Return the unit vector of the universal joint along the second axis of the fixed revolute joint.

- ReturnMatrix Find_VctC ()

Return the unit vector of the universal joint along the third axis of the fixed revolute joint.

- ReturnMatrix Find_AngularKin (const Real dl, const Real ddl)

  Return the angular speed (Column 1) and angular acceleration (Column 2) of the link.

- ReturnMatrix NormalForce ()

  Return the normal component of the reaction force of the platform acting on the link.

- ReturnMatrix AxialForce (const Matrix J1, const ColumnVector C, const int Index)

  Return the axial component of the reaction force of the platform acting on the link.

- ReturnMatrix Find_N (const Real Gravity=GRAVITY)

  Return the intermediate matrix N for force calculation.

- ReturnMatrix Moment ()

  Return the moment component transmitted to the link from the base or the platform (depending where is the universal joint).

- Real ActuationForce (const Matrix J1, const ColumnVector C, const int Index, const Real Gravity=GRAVITY)

  Return the actuation force that power the prismatic joint.

- ReturnMatrix Find_ACM1 (const Real dl, const Real ddl)

  Return the acceleration of the center of mass of the first part of the link.

Public Attributes

- ColumnVector UnitV

  Unit Vector of the link.

- ColumnVector aPos

  Position of the platform attachment point.

- ColumnVector Vu

  Unit Vector of the universal joint (Rotational).

- ColumnVector Vc

  Unit Vector of the universal joint (Rotational).

- ColumnVector Vv

  Unit Vector of the universal joint (Rotational).

- ColumnVector da

  Speed of the platform attachment point .

- ColumnVector dda

  Acceleration of the platform attachment point.

- ColumnVector LOmega

  Angular speed of the link.

- ColumnVector LAlpha

  Angular acceleration of the link.

- ColumnVector ACM1

  Acceleration of the first center of mass.

- ColumnVector M

  Moment vector of the link.

- ColumnVector N

  Intermediate vector for dynamics calculations .

- ColumnVector gravity

  Gravity vector.

- Real L

  Lenght of the link.

Private Attributes

- ColumnVector ap

  Platform coordinates of the link in the local frame.

- ColumnVector b

  Base coordinates of the link int the global frame.

- Real I1aa

  Inertia along the coaxial axis for part 1.

- Real I1nn

  Inertia along the tangent axis for part 1.

- Real I2aa

  Inertia along the coaxial axis for part 2.

- Real I2nn

  Inertia along the tangent axis for part 2.

- Real m1

  Mass of part 1.

- Real m2

  Mass of part 2.

- Real Lenght1

  Lenght between the mass center (part 1) and the platform attachment.

- Real Lenght2

  Lenght between the mass center (part 2) and the base attachment.

Friends

- class Stewart

4.11.2 Constructor & Destructor Documentation

4.11.2.1 LinkStewart::LinkStewart (const ColumnVector & *InitLink*, const Matrix *wRp*, const ColumnVector *q*)

Constructor.

Parameters:

InitLink,: LinkStewart initialization matrix.
wRp,: Rotation matrix
q,: Position of the platform

Definition at line 87 of file stewart.cpp.

References ACM1, ap, aPos, b, da, dda, Find_a(), Find_Lenght(), Find_UnitV(), Find_VctU(), gravity, I1aa, I1nn, I2aa, I2nn, L, LAlpha, Lenght1, Lenght2, LOmega, m1, m2, N, UnitV, Vc, Vu, and Vv.

4.11.3 Member Function Documentation

4.11.3.1 const LinkStewart & LinkStewart::operator= (const LinkStewart & *x*)

Definition at line 159 of file stewart.cpp.

References ACM1, ap, aPos, b, da, dda, gravity, I1aa, I1nn, I2aa, I2nn, L, LAlpha, Lenght1, Lenght2, LOmega, m1, m2, N, UnitV, Vc, Vu, and Vv.

4.11.3.2 void LinkStewart::LTransform (const Matrix *wRp*, const ColumnVector *q*)

Recalculate the link's parameters related to the platform position.

Parameters:

wRp,: rotation matrix.

q,: Position of the platform.

Definition at line 315 of file stewart.cpp.

References aPos, Find_a(), Find_Lenght(), Find_UnitV(), Find_VctC(), Find_VctV(), L, UnitV, Vc, and Vv.

4.11.3.3 void LinkStewart::d_LTransform (const ColumnVector *dq*, const ColumnVector *Omega*, const Real *dl*, const Real *ddl*)

Recalculate the link's parameters related to the platform speed.

Parameters:

dq,: Speed of the platform.

Omega,: Agular speed of the platform.

dl,: Extension rate of the link.

ddl,: Extension acceleration of the link.

Definition at line 329 of file stewart.cpp.

References da, Find_AngularKin(), Find_da(), LAlpha, and LOmega.

4.11.3.4 void LinkStewart::dd_LTransform (const ColumnVector *ddq*, const ColumnVector *Omega*, const ColumnVector *Alpha*, const Real *dl*, const Real *ddl*)

Recalculate the link's parameters related to the platform acceleration.

Parameters:

- ***ddq,:*** Acceleration of the platform.
- ***Omega,:*** Angular speed of the platform.
- ***Alpha,:*** Angular acceleration of the platform.
- ***dl,:*** Extension rate of the link.
- ***ddl,:*** Extension acceleration of the link.

Definition at line 347 of file stewart.cpp.

References dda, Find_AngularKin(), Find_dda(), LAlpha, and LOmega.

4.11.3.5 void LinkStewart::tau_LTransform (const Real *dl*, const Real *ddl*, const Real *Gravity*)

Recalculate the link's parameters related to the platform dynamics.

Parameters:

- ***dl,:*** Extension rate of the link.
- ***ddl,:*** Extension acceleration of the link.
- ***Gravity,:*** Gravity (9.81).

Definition at line 366 of file stewart.cpp.

References ACM1, Find_ACM1(), Find_N(), and N.

4.11.3.6 ReturnMatrix LinkStewart::Find_UnitV ()

Return the unit vector of the link direction.

The unit vector representing the orientation of the link is equal to:

$n = \frac{a_w - b}{Lenght}$

where:

- A is the position of the attachment point on the platform (world referential).
- B is the position of the attachment point on the base (world referential).
- Lenght is the lenght of the link.

Definition at line 416 of file stewart.cpp.

References aPos, b, and L.

Referenced by LinkStewart(), and LTransform().

4.11.3.7 ReturnMatrix LinkStewart::Find_a (const Matrix *wRp*, const ColumnVector *q*)

Return the position of the attachment point on the platform.

Parameters:

wRp,: Rotation matrix.

q,: Position of the platform.

The position of the attachment point on the platform is equal to the position of the center of the platform plus the position of the attach (in the local referencial) multiplicated by the rotation matrix:

$a = (x, y, z)_q + wRp \cdot a_l$

where:

- a_l is the position of the attach in the local referencial
- $(x, y, z)_q$ is the position of the platform center (first 3 elements of the q vector)

Definition at line 378 of file stewart.cpp.

References ap.

Referenced by LinkStewart(), and LTransform().

4.11.3.8 ReturnMatrix LinkStewart::Find_da (const ColumnVector *dq*, const ColumnVector *Omega*)

Return the speed of the attachment point of the link on the platform.

Parameters:

dq,: Speed of the platform

Omega,: Angular speed of the platform

This function represent the equation: $\dot{a} = (\dot{x}, \dot{y}, \dot{z})_p + \omega \times a_w$

Where:

- $(\dot{x}, \dot{y}, \dot{z})_p$ is the speed of the platform center (first 3 elements of dq vector)
- ω is the angular speed of the platform
- a_w is the position of the attachment point of the link to the platform in the world referential

Definition at line 438 of file stewart.cpp.

References aPos, and da.

Referenced by d_LTransform().

4.11.3.9 ReturnMatrix LinkStewart::Find_dda (const ColumnVector *ddq*, const ColumnVector *Omega*, const ColumnVector *Alpha*)

Return the acceleration of the attachment point of the link on the platform.

Parameters:

ddq,: Acceleration of the platform.

Omega,: Angular speed of the platform.

Alpha,: Angular acceleration of the platform

This function represent the equation: $\ddot{a} = (\ddot{x}, \ddot{y}, \ddot{z})_p + \alpha \times a_w + \omega \times (\omega \times a_w)$

Where:

- $(\ddot{x}, \ddot{y}, \ddot{z})_p$ is the acceleration of the platform center (first 3 elements of ddq vector)
- α is the angular acceleration of the platform
- ω is the angular speed of the platform
- a_w is the position of the attachment point of the link to the platform in the world referential

Definition at line 464 of file stewart.cpp.

References aPos, and dda.

Referenced by dd_LTransform().

4.11.3.10 Real LinkStewart::Find_Lenght ()

Return the lenght of the link.

$l = \sqrt{(a_w - b) \cdot (a_w - b)}$

where:

- a_w is the position of the attachment point of the link to the platform in the world referential
- b is the attachment point of the link to the base

Definition at line 486 of file stewart.cpp.

References aPos, and b.

Referenced by LinkStewart(), and LTransform().

4.11.3.11 ReturnMatrix LinkStewart::Find_VctU ()

Return the unit vector of the universal joint along the first axis of the fixed revolute joint.

This vector is equal to the unitary projection of the link unit vector on the X-Z plane:

$u_x = \frac{n_x}{\sqrt{n_x^2+n_z^2}}$;$u_y = 0$;$u_z = \frac{n_z}{\sqrt{n_x^2+n_z^2}}$

where:

- u_x, u_y and u_z are the elements of the vector
- n_x and n_z are the x component and the z component of the link unit vector

Definition at line 503 of file stewart.cpp.

References UnitV.

Referenced by LinkStewart().

4.11.3.12 ReturnMatrix LinkStewart::Find_VctV ()

Return the unit vector of the universal joint along the second axis of the fixed revolute joint.

Eq:

$v = \frac{u \times n}{\|u \times n\|}$

Where:

- u is the unit vector of the universal joint along the first axis of the fixed revolute joint
- n is the unit vector of the link

Definition at line 527 of file stewart.cpp.

References UnitV, and Vu.

Referenced by LTransform().

4.11.3.13 ReturnMatrix LinkStewart::Find_VctC ()

Return the unit vector of the universal joint along the third axis of the fixed revolute joint.

Eq:

$c = u \times v$

Where:

- u is the unit vector of the universal joint along the first axis of the fixed revolute joint
- v is the unit vector of the universal joint along the second axis of the fixed revolute joint

Definition at line 548 of file stewart.cpp.

References Vu, and Vv.

Referenced by LTransform().

4.11.3.14 ReturnMatrix LinkStewart::Find_AngularKin (const Real *dl*, const Real *ddl*)

Return the angular speed (Column 1) and angular acceleration (Column 2) of the link.

Parameters:

dl,: Extention rate of the link

ddl,: Extention acceleration of the link

Eqs for angular speed:

$\omega_u = -(\dot{a} - \dot{l}n) \cdot v/(ln \cdot c)$

$\omega_v = (\dot{a} - \dot{l}n) \cdot u/(ln \cdot c)$

$\omega = \omega_u u + \omega_v v$

Eqs for angular acceleration:

$\ddot{a}' = \ddot{a} - \omega_u \omega_v lc \times n - \ddot{l}n - 2\dot{l}\omega \times n - l\omega \times (\omega \times n)$

$\alpha_u = -\ddot{a}' \cdot v/(ln \cdot c)$

$\alpha_v = \ddot{a}' \cdot u/(ln \cdot c)$

$\alpha = \alpha_u u + \alpha_v v + \omega_u \omega_v c$

where:

- $\dot{a}$ is the speed of the attachment point of the link to the platform
- $\dot{l}$ is the extension rate of the link
- n is the unit vector of the link
- l is the lenght of the link
- u, v, c are the rot. vectors of the universal joint

Definition at line 585 of file stewart.cpp.

References da, dda, L, UnitV, Vc, Vu, and Vv.

Referenced by d_LTransform(), and dd_LTransform().

4.11.3.15 ReturnMatrix LinkStewart::NormalForce ()

Return the normal component of the reaction force of the platform acting on the link.

Eq:

$F^n = (N \times n - M \times n)/l$

Where:

- N is an intermediate matrix (Find_N())
- n is the unit vector of the link
- M is the reaction moment on the link (Moment())
- l is the lenght of the link

Definition at line 687 of file stewart.cpp.

References L, Moment(), N, and UnitV.

4.11.3.16 ReturnMatrix LinkStewart::AxialForce (const Matrix *J1*, const ColumnVector *C*, const int *Index*)

Return the axial component of the reaction force of the platform acting on the link.

Parameters:

J1,: First intermidiate jacobian matrix (find with Stewart::Find_InvJacob1())

C,: Intermidiate matrix in the dynamics calculation (find with Stewart::Find_C())

Index,: Number of the link (1 to 6)

Eq:

$$\begin{pmatrix} f_1^a \\ \vdots \\ f_6^a \end{pmatrix} = J_1^T C$$

Where:

- J_q^T is the jacobian matrix
- C is an intermediate matrix (Stewart::Find_C())

Definition at line 727 of file stewart.cpp.

References UnitV.

Referenced by ActuationForce().

4.11.3.17 ReturnMatrix LinkStewart::Find_N (const Real *Gravity* = `GRAVITY`)

Return the intermediate matrix N for force calculation.

Parameters:

Gravity,: Gravity (9.81)

Eqs:

$$I_1 = I_{1aa} nn^T + I_{1nn}(I_{3\times3} - nn^T)$$

$$I_2 = I_{2aa} nn^T + I_{2nn}(I_{3\times3} - nn^T)$$

$$N = -m_1(l-l_1)n \times G - m_2 l_2(n \times G) + (I_1+I_2)\alpha - (I_1+I_2)\omega \times \omega + m_1(l-l_1)n \times a_1 + m_2 l_2 n \times a_2$$

Eq for a_2 (a_1 is found with the Find_ACM1 function):

$$a_2 = l_2 \omega \times (\omega \times n) + l_2 \alpha \times n$$

Where:

- I_{1aa} and I_{2aa} are the mass moment of inertia component about the main axis of the two parts of the link
- I_{1nn} and I_{2nn} are the mass moment of inertia component about the normal axis of the two parts of the link
- n is the unit vector of the link
- $I_{3\times3}$ is a identity matrix

- m_1 is the mass of the first part of the link
- l is the lenght of the link
- l_1 is the distance between the center of mass of the first part of the link and the base
- G is the gravity
- m_2 is the mass of the second part of the link
- l_2 is the distance between the center of mass of the second part of the link and the platform
- α is the angular acceleration of the link
- ω is the angular speed of the link
- a_1 and a_2 are the acceleration of the center of mass of the two parts of the links

Definition at line 641 of file stewart.cpp.

References ACM1, gravity, I1aa, I1nn, I2aa, I2nn, L, LAlpha, Lenght1, Lenght2, LOmega, m1, m2, and UnitV.

Referenced by tau_LTransform().

4.11.3.18 ReturnMatrix LinkStewart::Moment ()

Return the moment component transmitted to the link from the base or the platform (depending where is the universal joint).

Eq:

$M = N \cdot n/c \cdot n$

Where:

- N is an intermediate matrix (Find_N)
- n is the unit vector of the link
- c is the rot. vector along the normal axis of the universal joint

Definition at line 666 of file stewart.cpp.

References M, N, UnitV, and Vc.

Referenced by Stewart::Find_C(), and NormalForce().

4.11.3.19 Real LinkStewart::ActuationForce (const Matrix *J1*, const ColumnVector *C*, const int *Index*, const Real *Gravity* = `GRAVITY`)

Return the actuation force that power the prismatic joint.

Parameters:

J1,: First intermidiate jacobian matrix (find with Stewart::Find_InvJacob1())

C,: Intermidiate matrix in the dynamics calculation (find with Stewart::Find_C())

Index,: Number of the link (1 to 6)

Gravity,: Gravity (9.81)

Eq:

$f = m_1 a_1 \cdot n - f^a - m_1 G \cdot n$

Where:

- m_1 is the mass of the first part of the link
- a_1 is the acceleration of the center of mass of the first part of the link
- n is the unit vector of the link
- f^a is from LinkStewart::AxialForce
- G is gravity

Definition at line 759 of file stewart.cpp.

References ACM1, AxialForce(), gravity, m1, and UnitV.

4.11.3.20 ReturnMatrix LinkStewart::Find_ACM1 (const Real *dl*, const Real *ddl*)

Return the acceleration of the center of mass of the first part of the link.

Parameters:

dl,: Extention rate of the link

ddl,: Extention acceleration of the link

Eq:

$a_1 = (l - l_1)\omega \times (\omega \times n) + (l - l_1)\alpha \times n + 2\omega \times \dot{l}n + \ddot{l}n$

Where:

- l is the lenght of the link
- l_1 is the distance between the center of mass of the first part of the link to the base
- ω is the angular speed of the link
- α is the angular acceleration of the link
- n is the unit vector of the link
- $\dot{l}$ is the extension rate of the link
- $\ddot{l}$ is the extension acceleration of the link

Definition at line 802 of file stewart.cpp.

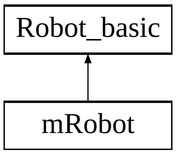
References L, LAlpha, Lenght1, LOmega, and UnitV.

Referenced by tau_LTransform().

4.12 mRobot Class Reference

`#include <robot.h>`

Inheritance diagram for mRobot::



4.12.1 Detailed Description

Modified DH notation robot class.

Definition at line 389 of file robot.h.

Public Member Functions

- mRobot (const int ndof=1)

  Constructor.

- mRobot (const Matrix &initrobot_motor)

  Constructor.

- mRobot (const Matrix &initrobot, const Matrix &initmotor)

  Constructor.

- mRobot (const mRobot &x)

  Copy constructor.

- **mRobot** (const std::string &filename, const std::string &robotName)
- virtual ∼mRobot ()

  Destructor.

- virtual void robotType_inv_kin ()

  Identify inverse kinematics familly.

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

  Overload inv_kin function.

- virtual ReturnMatrix inv_kin (const Matrix &Tobj, const int mj, const int endlink, bool &converge)

    Inverse kinematics solutions.

- virtual ReturnMatrix inv_kin_rhino (const Matrix &Tobj, bool &converge)

    Analytic Rhino inverse kinematics.

- virtual ReturnMatrix inv_kin_puma (const Matrix &Tobj, bool &converge)

    Analytic Puma inverse kinematics.

- virtual ReturnMatrix inv_kin_schilling (const Matrix &Tobj, bool &converge)

    Analytic Schilling inverse kinematics.

- virtual void kine_pd (Matrix &Rot, ColumnVector &pos, ColumnVector &pos_dot, const int ref) const

    Direct kinematics with velocity.

- virtual ReturnMatrix jacobian (const int ref=0) const

    Jacobian of mobile links expressed at frame ref.

- virtual ReturnMatrix jacobian (const int endlink, const int ref) const

    Jacobian of mobile joints up to endlink expressed at frame ref.

- virtual ReturnMatrix jacobian_dot (const int ref=0) const

    Jacobian derivative of mobile joints expressed at frame ref.

- virtual void dTdqi (Matrix &dRot, ColumnVector &dp, const int i)

    Partial derivative of the robot position (homogeneous transf.).

- virtual ReturnMatrix dTdqi (const int i)

    Partial derivative of the robot position (homogeneous transf.).

- virtual ReturnMatrix torque (const ColumnVector &q, const ColumnVector &qp, const ColumnVector &qpp)

    Joint torque, without contact force, based on Recursive Newton-Euler formulation.

- virtual ReturnMatrix torque (const ColumnVector &q, const ColumnVector &qp, const ColumnVector &qpp, const ColumnVector &Fext_, const ColumnVector &Next_)

    Joint torque based on Recursive Newton-Euler formulation.

- virtual ReturnMatrix torque_novelocity (const ColumnVector &qpp)

    Joint torque. when joint velocity is 0, based on Recursive Newton-Euler formulation.

- virtual 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)

  Delta torque dynamics.

- virtual void dq_torque (const ColumnVector &q, const ColumnVector &qp, const ColumnVector &qpp, const ColumnVector &dq, ColumnVector &torque, ColumnVector &dtorque)

  Delta torque due to delta joint position.

- virtual void dqp_torque (const ColumnVector &q, const ColumnVector &qp, const ColumnVector &dqp, ColumnVector &torque, ColumnVector &dtorque)

  Delta torque due to delta joint velocity.

- virtual ReturnMatrix G ()

  Joint torque due to gravity based on Recursive Newton-Euler formulation.

- virtual ReturnMatrix C (const ColumnVector &qp)

  Joint torque due to centrifugal and Corriolis based on Recursive Newton-Euler formulation.

4.12.2 Member Function Documentation

4.12.2.1 void mRobot::robotType_inv_kin () `[virtual]`

Identify inverse kinematics familly.

Identify the inverse kinematics analytic solution based on the similarity of the robot DH parameters and the DH parameters of know robots (ex: Puma, Rhino, ...). The inverse kinematics will be based on a numerical alogorithm if there is no match .

Implements Robot_basic.

Definition at line 1354 of file robot.cpp.

References Robot_basic::DEFAULT, Robot_basic::PUMA, Puma_mDH(), Robot_basic::RHINO, Rhino_mDH(), Robot_basic::robotType, Robot_basic::SCHILLING, and Schilling_mDH().

Referenced by mRobot().

4.12.2.2 ReturnMatrix mRobot::inv_kin (const Matrix & *Tobj*, const int *mj*, const int *endlink*, bool & *converge*) `[virtual]`

Inverse kinematics solutions.

The solution is based on the analytic inverse kinematics if robot type (familly) is Rhino or Puma, otherwise used the numerical algoritm defined in Robot_basic class.

Reimplemented from Robot_basic.

Definition at line 603 of file invkine.cpp.

References Robot_basic::inv_kin(), inv_kin_puma(), inv_kin_rhino(), inv_kin_schilling(), Robot_basic::PUMA, Robot_basic::RHINO, Robot_basic::robotType, and Robot_basic::SCHILLING.

4.12.2.3 ReturnMatrix mRobot::inv_kin_rhino (const Matrix & *Tobj*, bool & *converge*) `[virtual]`

Analytic Rhino inverse kinematics.

converge will be false if the desired end effector pose is outside robot range.

Implements Robot_basic.

Definition at line 628 of file invkine.cpp.

References Robot_basic::a, Link::a, Link::d, G(), Robot_basic::get_q(), K, Robot_basic::links, and M_PI.

Referenced by inv_kin().

4.12.2.4 ReturnMatrix mRobot::inv_kin_puma (const Matrix & *Tobj*, bool & *converge*) `[virtual]`

Analytic Puma inverse kinematics.

converge will be false if the desired end effector pose is outside robot range.

Implements Robot_basic.

Definition at line 733 of file invkine.cpp.

References Robot_basic::a, Link::a, C(), Link::d, Robot_basic::get_q(), Robot_basic::links, and M_PI.

Referenced by inv_kin().

4.12.2.5 ReturnMatrix mRobot::inv_kin_schilling (const Matrix & *Tobj*, bool & *converge*) `[virtual]`

Analytic Schilling inverse kinematics.

converge will be false if the desired end effector pose is outside robot range.

Implements Robot_basic.

Definition at line 893 of file invkine.cpp.

References Robot_basic::a, Link::a, C(), Link::d, Robot_basic::get_q(), K, Robot_basic::links, and M_PI.

Referenced by inv_kin().

4.12.2.6 void mRobot::kine_pd (Matrix & *Rot*, ColumnVector & *pos*, ColumnVector & *pos_dot*, const int *j*) const `[virtual]`

Direct kinematics with velocity.

Parameters:

Rot,: Frame j rotation matrix w.r.t to the base frame.

pos,: Frame j position vector wr.r.t to the base frame.

pos_dot,: Frame j velocity vector w.r.t to the base frame.

j,: Frame j. Print an error on the console if j is out of range.

Implements Robot_basic.

Definition at line 552 of file kinemat.cpp.

4.12.2.7 ReturnMatrix mRobot::jacobian (const int *endlink*, const int *ref*) const `[virtual]`

Jacobian of mobile joints up to endlink expressed at frame ref.

See Robot::jacobian for equations.

Implements Robot_basic.

Definition at line 682 of file kinemat.cpp.

4.12.2.8 ReturnMatrix mRobot::jacobian_dot (const int *ref* = `0`) const `[virtual]`

Jacobian derivative of mobile joints expressed at frame ref.

See Robot::jacobian_dot for equations.

Implements Robot_basic.

Definition at line 736 of file kinemat.cpp.

4.12.2.9 void mRobot::dTdqi (Matrix & *dRot*, ColumnVector & *dp*, const int *i*) `[virtual]`

Partial derivative of the robot position (homogeneous transf.).

This function computes the partial derivatives:

$$\frac{\partial {}^0T_n}{\partial q_i} = {}^0T_i Q_i \; {}^iT_n$$

with

$$Q_i = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

for a revolute joint and

$$Q_i = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

for a prismatic joint.

$dRot$ and dp are modified on output.

Implements Robot_basic.

Definition at line 582 of file kinemat.cpp.

References Robot_basic::dof, Robot_basic::dp, Robot_basic::links, Robot_basic::p, Link::p, Link::R, Robot_basic::R, and threebythreeident.

Referenced by dTdqi().

4.12.2.10 ReturnMatrix mRobot::dTdqi (const int *i*) `[virtual]`

Partial derivative of the robot position (homogeneous transf.).

See mRobot::dTdqi(Matrix & dRot, ColumnVector & dp, const int i) for equations.

Implements Robot_basic.

Definition at line 664 of file kinemat.cpp.

References dTdqi().

4.12.2.11 ReturnMatrix mRobot::torque (const ColumnVector & *q*, const ColumnVector & *qp*, const ColumnVector & *qpp*, const ColumnVector & *Fext_*, const ColumnVector & *Next_*) `[virtual]`

Joint torque based on Recursive Newton-Euler formulation.

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.

$z_0 = \left[\begin{array}{ccc} 0 & 0 & 1 \end{array} \right]^T$

$p_i = \left[\begin{array}{ccc} a_{i-1} & -d_i sin\alpha_{i-1} & d_i cos\alpha_{i-1} \end{array} \right]^T$ is the position of the i^{th} with respect to the $i-1^{th}$ frame.

Forward Iterations for $i = 1, 2, \ldots, n$. Initialize: $\omega_0 = \dot{\omega}_0 = 0$ and $\dot{v}_0 = -g$.

$$\omega_i = R_i^T \omega_{i-1} + \sigma_i z_0 \dot{\theta}_i$$

$$\dot{\omega}_i = R_i^T \dot{\omega}_{i-1} + \sigma_i R_i^T \omega_{i-1} \times z_0 \dot{\theta}_i + \sigma_i z_0 \ddot{\theta}_i$$

$$\dot{v}_i = R_i^T (\dot{\omega}_{i-1} \times p_i + \omega_{i-1} \times (\omega_{i-1} \times p_i) + \dot{v}_{i-1}) + (1 - \sigma_i)(2\omega_i \times z_0 dotd_i + z_0 \ddot{d}_i)$$

Backward Iterations for $i = n, n-1, \ldots, 1$. Initialize: $f_{n+1} = n_{n+1} = 0$.

$$\dot{v}_{ci} = \dot{\omega}_i \times r_i + \omega_i \times (\omega_i \times r_i) + v_i$$

$$F_i = m_i \dot{v}_{ci}$$

$$N_i = I_{ci} \ddot{\omega}_i \ + \omega_i \times I_{ci} \omega_i$$

$$f_i = R_{i+1} f_{i+1} + F_i$$

$$n_i = N_i + R_{i+1} n_{i+1} + r_i \times F_i + p_{i+1} \times R_{i+1} f_{i+1}$$

$$\tau_i = \sigma_i n_i^T z_0 + (1 - \sigma_i) f_i^T z_0$$

Implements Robot_basic.

Definition at line 422 of file dynamics.cpp.

References Robot_basic::a, Link::B, Link::Cf, Robot_basic::dof, Robot_basic::f, Robot_basic::F, Robot_basic::fix, Link::Gr, Robot_basic::gravity, Link::I, Link::Im, Robot_basic::links, Robot_basic::n, Robot_basic::N, Robot_basic::p, Link::R, Robot_basic::R, Robot_basic::set_q(), Robot_basic::set_qp(), sign(), Robot_basic::vp, Robot_basic::w, Robot_basic::wp, and Robot_basic::z0.

4.12.2.12 void mRobot::delta_torque (const ColumnVector & *q*, const ColumnVector & *qp*, const ColumnVector & *qpp*, const ColumnVector & *dq*, const ColumnVector & *dqp*, const ColumnVector & *dqpp*, ColumnVector & *ltorque*, ColumnVector & *dtorque*) `[virtual]`

Delta torque dynamics.

This function computes

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

Murray and Neuman Cite_: Murray86 have developed an efficient recursive linearized Newton-Euler formulation. In order to apply the RNE as presented in let us define the following variables

$$p_{di} = \frac{\partial p_i}{\partial d_i} = \begin{bmatrix} 0 & \sin\alpha_i & \cos\alpha_i \end{bmatrix}^T$$

$$Q = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Forward Iterations for $i = 1, 2, \dots, n$. Initialize: $\delta\omega_0 = \delta\dot{\omega}_0 = \delta\dot{v}_0 = 0$.

$$\delta\omega_i = R_i^T\delta\omega_{i-1} + \sigma_i(z_0\delta\dot{\theta}_i - QR_i^T\omega_i\delta\theta_i)$$

$$\delta\dot{\omega}_i = R_i^T\delta\dot{w}_{i-1} + \sigma_i[R_i^T\delta\omega_{i-1}\times z_0\dot{\theta}_i + R_i^T\omega_{i-1}\times z_0\delta\dot{\theta}_i + z_0\ddot{\theta}_i - (QR_i^T\dot{\omega}_{i-1} + QR_i^T\omega_{i-1}\times\omega z_0\dot{\theta}_i)\delta\theta_i]$$

$$\delta\dot{v}_i = R_i^T\Big(\delta\dot{\omega}_{i-1}\times p_i + \delta\omega_{i-1}\times(\omega_{i-1}\times p_i) + \omega_{i-1}\times(\delta\omega_{i-1}\times p_i) + \delta\dot{v}_i\Big) + (1-\sigma_i)\Big(2\delta\omega_i\times z_0\dot{d}_i + 2\omega_i\times z_0\delta\dot{d}_i + z_0\delta\ddot{d}_i\Big) - \sigma_i QR_i^T\Big($$

Backward Iterations for $i = n, n-1, \dots, 1$. Initialize: $\delta f_{n+1} = \delta n_{n+1} = 0$.

$$\delta\dot{v}_{ci} = \delta\dot{v}_i + \delta\dot{\omega}_i\times r_i + \delta\omega_i\times(\omega_i\times r_i) + \omega_i\times(\delta\omega_i\times r_i)$$

$$\delta F_i = m_i\delta\dot{v}_{ci}$$

$$\delta N_i = I_{ci}\delta\dot{\omega}_i + \delta\omega_i\times(I_{ci}\omega_i) + \omega_i\times(I_{ci}\delta\omega_i)$$

$$\delta f_i = R_{i+1}\delta f_{i+1} + \delta F_i + \sigma_{i+1}R_{i+1}Qf_{i+1}\delta\theta_{i+1}$$

$$\delta n_i = \delta N_i + R_{i+1}\delta n_{i+1} + r_i\times\delta F_i + p_{i+1}\times R_{i+1}\delta f_{i+1} + \sigma_{i+1}\Big(R_{i+1}Qn_{i+1} + p_{i+1}\times R_{i+1}Qf_{i+1}\Big)\delta\theta_{i+1} + (1-\sigma_{i+1})p_{di+1}p_{di+1}\times$$

$$\delta\tau_i = \sigma\delta n_i^T z_0 + (1-\sigma_i)\delta f_i^T z_0$$

Implements Robot_basic.

Definition at line 291 of file delta_t.cpp.

References Robot_basic::a, Robot_basic::da, Robot_basic::df, Robot_basic::dF, Robot_basic::dn, Robot_basic::dN, Robot_basic::dof, Robot_basic::dp, Robot_basic::dvp, Robot_basic::dw, Robot_basic::dwp, Robot_basic::f, Robot_basic::F, Robot_basic::gravity, Link::I, Robot_basic::links, Link::m, Robot_basic::n, Robot_basic::N, Link::p, Robot_basic::p, Link::R, Robot_basic::R, Robot_basic::set_q(), Robot_basic::vp, Robot_basic::w, Robot_basic::wp, and Robot_basic::z0.

4.12.2.13 void mRobot::dq_torque (const ColumnVector & *q*, const ColumnVector & *qp*, const ColumnVector & *qpp*, const ColumnVector & *dq*, ColumnVector & *ltorque*, ColumnVector & *dtorque*) `[virtual]`

Delta torque due to delta joint position.

This function computes $S_2(q, \dot{q}, \ddot{q})\delta q$. See mRobot::delta_torque for equations.

Implements Robot_basic.

Definition at line 205 of file comp_dq.cpp.

References Robot_basic::a, Robot_basic::da, Robot_basic::df, Robot_basic::dF, Robot_basic::dn, Robot_basic::dN, Robot_basic::dof, Robot_basic::dp, Robot_basic::dvp, Robot_basic::dw, Robot_basic::dwp, Robot_basic::f, Robot_basic::F, Robot_basic::gravity, Link::I, Robot_basic::links, Link::m, Robot_basic::n, Robot_basic::N, Link::p, Robot_basic::p, Link::R, Robot_basic::R, Robot_basic::set_q(), Robot_basic::vp, Robot_basic::w, Robot_basic::wp, and Robot_basic::z0.

4.12.2.14 void mRobot::dqp_torque (const ColumnVector & *q*, const ColumnVector & *qp*, const ColumnVector & *dqp*, ColumnVector & *ltorque*, ColumnVector & *dtorque*) `[virtual]`

Delta torque due to delta joint velocity.

This function computes $S_1(q, \dot{q}, \ddot{q})\delta \dot{q}$. See mRobot::delta_torque for equations.

Implements Robot_basic.

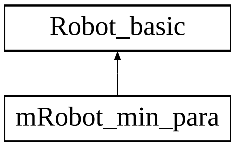
Definition at line 188 of file comp_dqp.cpp.

References Robot_basic::a, Robot_basic::da, Robot_basic::df, Robot_basic::dF, Robot_basic::dn, Robot_basic::dN, Robot_basic::dof, Robot_basic::dp, Robot_basic::dvp, Robot_basic::dw, Robot_basic::dwp, Robot_basic::f, Robot_basic::F, Robot_basic::gravity, Link::I, Robot_basic::links, Link::m, Robot_basic::n, Robot_basic::N, Link::p, Robot_basic::p, Link::R, Robot_basic::R, Robot_basic::set_q(), Robot_basic::vp, Robot_basic::w, Robot_basic::wp, and Robot_basic::z0.

4.13 mRobot_min_para Class Reference

`#include <robot.h>`

Inheritance diagram for mRobot_min_para::



4.13.1 Detailed Description

Modified DH notation and minimal inertial parameters robot class.

Definition at line 437 of file robot.h.

Public Member Functions

- mRobot_min_para (const int ndof=1)

  Constructor.

- mRobot_min_para (const Matrix &dhinit)

  Constructor.

- mRobot_min_para (const Matrix &initrobot, const Matrix &initmotor)

  Constructor.

- mRobot_min_para (const mRobot_min_para &x)

  Copy constructor.

- **mRobot_min_para** (const std::string &filename, const std::string &robotName)

- virtual ∼mRobot_min_para ()

  Destructor.

- virtual void robotType_inv_kin ()

  Identify inverse kinematics familly.

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

  Overload inv_kin function.

- virtual ReturnMatrix inv_kin (const Matrix &Tobj, const int mj, const int endlink, bool &converge)

  Inverse kinematics solutions.

- virtual ReturnMatrix inv_kin_rhino (const Matrix &Tobj, bool &converge)

  Analytic Rhino inverse kinematics.

- virtual ReturnMatrix inv_kin_puma (const Matrix &Tobj, bool &converge)

  Analytic Puma inverse kinematics.

- virtual ReturnMatrix inv_kin_schilling (const Matrix &Tobj, bool &converge)

  Analytic Schilling inverse kinematics.

- virtual void kine_pd (Matrix &Rot, ColumnVector &pos, ColumnVector &pos_dot, const int ref=0) const

  Direct kinematics with velocity.

- virtual ReturnMatrix jacobian (const int ref=0) const

  Jacobian of mobile links expressed at frame ref.

- virtual ReturnMatrix jacobian (const int endlink, const int ref) const

  Jacobian of mobile joints up to endlink expressed at frame ref.

- virtual ReturnMatrix jacobian_dot (const int ref=0) const

  Jacobian derivative of mobile joints expressed at frame ref.

- virtual void dTdqi (Matrix &dRot, ColumnVector &dp, const int i)

  Partial derivative of the robot position (homogeneous transf.).

- virtual ReturnMatrix dTdqi (const int i)

  Partial derivative of the robot position (homogeneous transf.).

- virtual ReturnMatrix torque (const ColumnVector &q, const ColumnVector &qp, const ColumnVector &qpp)

  Joint torque without contact force based on Recursive Newton-Euler formulation.

- virtual ReturnMatrix torque (const ColumnVector &q, const ColumnVector &qp, const ColumnVector &qpp, const ColumnVector &Fext_, const ColumnVector &Next_)

  Joint torque based on Recursive Newton-Euler formulation.

- virtual ReturnMatrix torque_novelocity (const ColumnVector &qpp)

Joint torque. when joint velocity is 0, based on Recursive Newton-Euler formulation.

- virtual 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)

  Delta torque dynamics.

- virtual void dq_torque (const ColumnVector &q, const ColumnVector &qp, const ColumnVector &qpp, const ColumnVector &dq, ColumnVector &torque, ColumnVector &dtorque)

  Delta torque due to delta joint position.

- virtual void dqp_torque (const ColumnVector &q, const ColumnVector &qp, const ColumnVector &dqp, ColumnVector &torque, ColumnVector &dtorque)

  Delta torque due to delta joint velocity.

- virtual ReturnMatrix G ()

  Joint torque due to gravity based on Recursive Newton-Euler formulation.

- virtual ReturnMatrix C (const ColumnVector &qp)

  Joint torque due to centrifugal and Corriolis based on Recursive Newton-Euler formulation.

4.13.2 Member Function Documentation

4.13.2.1 void mRobot_min_para::robotType_inv_kin () `[virtual]`

Identify inverse kinematics familly.

Identify the inverse kinematics analytic solution based on the similarity of the robot DH parameters and the DH parameters of know robots (ex: Puma, Rhino, ...). The inverse kinematics will be based on a numerical alogorithm if there is no match .

Implements Robot_basic.

Definition at line 1423 of file robot.cpp.

References Robot_basic::DEFAULT, Robot_basic::PUMA, Puma_mDH(), Robot_basic::RHINO, Rhino_mDH(), Robot_basic::robotType, Robot_basic::SCHILLING, and Schilling_mDH().

Referenced by mRobot_min_para().

4.13.2.2 ReturnMatrix mRobot_min_para::inv_kin (const Matrix & *Tobj*, const int *mj*, const int *endlink*, bool & *converge*) `[virtual]`

Inverse kinematics solutions.

The solution is based on the analytic inverse kinematics if robot type (familly) is Rhino or Puma, otherwise used the numerical algoritm defined in Robot_basic class.

Reimplemented from Robot_basic.

Definition at line 1009 of file invkine.cpp.

References Robot_basic::inv_kin(), inv_kin_puma(), inv_kin_rhino(), Robot_basic::PUMA, Robot_basic::RHINO, and Robot_basic::robotType.

4.13.2.3 ReturnMatrix mRobot_min_para::inv_kin_rhino (const Matrix & *Tobj*, bool & *converge*) `[virtual]`

Analytic Rhino inverse kinematics.

converge will be false if the desired end effector pose is outside robot range.

Implements Robot_basic.

Definition at line 1031 of file invkine.cpp.

References Robot_basic::a, Link::a, Link::d, G(), Robot_basic::get_q(), K, Robot_basic::links, and M_PI.

Referenced by inv_kin().

4.13.2.4 ReturnMatrix mRobot_min_para::inv_kin_puma (const Matrix & *Tobj*, bool & *converge*) `[virtual]`

Analytic Puma inverse kinematics.

converge will be false if the desired end effector pose is outside robot range.

Implements Robot_basic.

Definition at line 1128 of file invkine.cpp.

References Robot_basic::a, Link::a, C(), Link::d, Robot_basic::get_q(), Robot_basic::links, and M_PI.

Referenced by inv_kin().

4.13.2.5 ReturnMatrix mRobot_min_para::inv_kin_schilling (const Matrix & *Tobj*, bool & *converge*) `[virtual]`

Analytic Schilling inverse kinematics.

converge will be false if the desired end effector pose is outside robot range.

Implements Robot_basic.

Definition at line 1289 of file invkine.cpp.

References Robot_basic::a, Link::a, C(), Link::d, Robot_basic::get_q(), K, Robot_basic::links, and M_PI.

4.13.2.6 void mRobot_min_para::kine_pd (Matrix & *Rot*, ColumnVector & *pos*, ColumnVector & *pos_dot*, const int *j* = `0`) const `[virtual]`

Direct kinematics with velocity.

Parameters:

Rot,: Frame j rotation matrix w.r.t to the base frame.

pos,: Frame j position vector wr.r.t to the base frame.

pos_dot,: Frame j velocity vector w.r.t to the base frame.

j,: Frame j. Print an error on the console if j is out of range.

Implements Robot_basic.

Definition at line 799 of file kinemat.cpp.

4.13.2.7 ReturnMatrix mRobot_min_para::jacobian (const int *endlink*, const int *ref*) const `[virtual]`

Jacobian of mobile joints up to endlink expressed at frame ref.

See Robot::jacobian for equations.

Implements Robot_basic.

Definition at line 905 of file kinemat.cpp.

4.13.2.8 ReturnMatrix mRobot_min_para::jacobian_dot (const int *ref* = `0`) const `[virtual]`

Jacobian derivative of mobile joints expressed at frame ref.

See Robot::jacobian_dot for equations.

Implements Robot_basic.

Definition at line 961 of file kinemat.cpp.

4.13.2.9 void mRobot_min_para::dTdqi (Matrix & *dRot*, ColumnVector & *dp*, const int *i*) `[virtual]`

Partial derivative of the robot position (homogeneous transf.).

This function computes the partial derivatives:

$$\frac{\partial {}^0T_n}{\partial q_i} = {}^0T_i Q_i \, {}^iT_n$$

See mRobot::dTdqi(Matrix & dRot, ColumnVector & dp, const int i) for equations.

Implements Robot_basic.

Definition at line 829 of file kinemat.cpp.

References Robot_basic::dof, Robot_basic::dp, Robot_basic::links, Robot_basic::p, Link::p, Link::R, Robot_basic::R, and threebythreeident.

Referenced by dTdqi().

4.13.2.10 ReturnMatrix mRobot_min_para::dTdqi (const int *i*) `[virtual]`

Partial derivative of the robot position (homogeneous transf.).

See mRobot::dTdqi(Matrix & dRot, ColumnVector & dp, const int i) for equations.

Implements Robot_basic.

Definition at line 887 of file kinemat.cpp.

References dTdqi().

4.13.2.11 ReturnMatrix mRobot_min_para::torque (const ColumnVector & *q*, const ColumnVector & *qp*, const ColumnVector & *qpp*, const ColumnVector & *Fext_*, const ColumnVector & *Next_*) `[virtual]`

Joint torque based on Recursive Newton-Euler formulation.

See ReturnMatrix mRobot::torque(const ColumnVector & q, const ColumnVector & qp, const ColumnVector & qpp, const ColumnVector & Fext, const ColumnVector & Next) for the Recursive Newton-Euler formulation.

Implements Robot_basic.

Definition at line 697 of file dynamics.cpp.

References Link::B, Link::Cf, Robot_basic::dof, Robot_basic::f, Robot_basic::F, Robot_basic::fix, Link::Gr, Robot_basic::gravity, Link::I, Link::Im, Robot_basic::links, Robot_basic::n, Robot_basic::N, Robot_basic::p, Link::R, Robot_basic::R, Robot_basic::set_q(), Robot_basic::set_qp(), sign(), Robot_basic::vp, Robot_basic::w, Robot_basic::wp, and Robot_basic::z0.

4.13.2.12 void mRobot_min_para::delta_torque (const ColumnVector & *q*, const ColumnVector & *qp*, const ColumnVector & *qpp*, const ColumnVector & *dq*, const ColumnVector & *dqp*, const ColumnVector & *dqpp*, ColumnVector & *ltorque*, ColumnVector & *dtorque*) `[virtual]`

Delta torque dynamics.

See mRobot::delta_torque for equations.

Implements Robot_basic.

Definition at line 511 of file delta_t.cpp.

References Robot_basic::df, Robot_basic::dF, Robot_basic::dn, Robot_basic::dN, Robot_basic::dof, Robot_basic::dp, Robot_basic::dvp, Robot_basic::dw, Robot_basic::dwp, Robot_basic::f, Robot_basic::F, Robot_basic::gravity, Link::I, Robot_basic::links, Link::m, Robot_basic::n, Robot_basic::N, Link::p, Robot_basic::p, Link::R, Robot_basic::R, Robot_basic::set_q(), Robot_basic::vp, Robot_basic::w, Robot_basic::wp, and Robot_basic::z0.

4.13.2.13 void mRobot_min_para::dq_torque (const ColumnVector & *q*, const ColumnVector & *qp*, const ColumnVector & *qpp*, const ColumnVector & *dq*, ColumnVector & *ltorque*, ColumnVector & *dtorque*) `[virtual]`

Delta torque due to delta joint position.

This function computes $S_2(q, \dot{q}, \ddot{q})\delta q$. See mRobot::delta_torque for equations.

Implements Robot_basic.

Definition at line 335 of file comp_dq.cpp.

References Robot_basic::df, Robot_basic::dF, Robot_basic::dn, Robot_basic::dN, Robot_basic::dof, Robot_basic::dp, Robot_basic::dvp, Robot_basic::dw, Robot_basic::dwp, Robot_basic::f, Robot_basic::F, Robot_basic::gravity, Link::I, Robot_basic::links, Link::m, Robot_basic::n, Robot_basic::N, Link::p, Robot_basic::p, Link::R, Robot_basic::R, Robot_basic::set_q(), Robot_basic::vp, Robot_basic::w, Robot_basic::wp, and Robot_basic::z0.

4.13.2.14 void mRobot_min_para::dqp_torque (const ColumnVector & *q*, const ColumnVector & *qp*, const ColumnVector & *dqp*, ColumnVector & *ltorque*, ColumnVector & *dtorque*) `[virtual]`

Delta torque due to delta joint velocity.

This function computes $S_1(q, \dot{q}, \ddot{q})\delta \dot{q}$. See mRobot::delta_torque for equations.

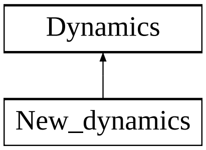
Implements Robot_basic.

Definition at line 303 of file comp_dqp.cpp.

References Robot_basic::df, Robot_basic::dF, Robot_basic::dn, Robot_basic::dN, Robot_basic::dof, Robot_basic::dp, Robot_basic::dvp, Robot_basic::dw, Robot_basic::dwp, Robot_basic::f, Robot_basic::F, Robot_basic::gravity, Link::I, Robot_basic::links, Link::m, Robot_basic::n, Robot_basic::N, Link::p, Robot_basic::p, Link::R, Robot_basic::R, Robot_basic::set_q(), Robot_basic::vp, Robot_basic::w, Robot_basic::wp, and Robot_basic::z0.

4.14 New_dynamics Class Reference

Inheritance diagram for New_dynamics::



4.14.1 Detailed Description

This is an example of customize Dynamics class.

This class enherite from Dynamics class. At every time frame the new virtual plot functions record the current time and the robot joints positions. The data can then be used to create a plot.

Definition at line 63 of file demo_2dof_pd.cpp.

Public Member Functions

- New_dynamics (Robot_basic $*$robot_)

  Constructor.

- virtual void plot ()

  Customize plot function.

Public Attributes

- Robot_basic $*$ robot
- bool first_pass_plot
- RowVector tout
- Matrix xout
- int i

4.14.2 Member Function Documentation

4.14.2.1 void New_dynamics::plot () `[virtual]`

Customize plot function.

Record the time (tout) and the joints positions (xout). This member function is call by the member function xdot.

Reimplemented from Dynamics.

Definition at line 94 of file demo_2dof_pd.cpp.

References first_pass_plot, i, Dynamics::nsteps, robot, Dynamics::tf_cont, Dynamics::time, Dynamics::to, tout, Dynamics::x, and xout.

4.14.3 Member Data Documentation

4.14.3.1 Robot_basic∗ New_dynamics::robot

Robot_basic pointer.

Reimplemented from Dynamics.

Definition at line 69 of file demo_2dof_pd.cpp.

Referenced by New_dynamics(), and plot().

4.14.3.2 bool New_dynamics::first_pass_plot

First time in plot function.

Definition at line 70 of file demo_2dof_pd.cpp.

Referenced by New_dynamics(), and plot().

4.14.3.3 RowVector New_dynamics::tout

Output time vector.

Definition at line 71 of file demo_2dof_pd.cpp.

Referenced by main(), and plot().

4.14.3.4 Matrix New_dynamics::xout

Output state vector.

Definition at line 72 of file demo_2dof_pd.cpp.

Referenced by main(), and plot().

4.14.3.5 int New_dynamics::i

Temporary index.

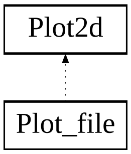
Definition at line 73 of file demo_2dof_pd.cpp.

Referenced by New_dynamics(), and plot().

4.15 Plot2d Class Reference

`#include <gnugraph.h>`

Inheritance diagram for Plot2d::



4.15.1 Detailed Description

2d plot object.

Definition at line 149 of file gnugraph.h.

Public Member Functions

- Plot2d (void)

  Constructor.

- void dump (void)

  Method to dump the content of Plot2d to stdout.

- void **settitle** (const std::string &t)
- void **setxlabel** (const std::string &t)
- void **setylabel** (const std::string &t)
- void **addcurve** (const Matrix &data, const std::string &label="", LineType_en enLineType=DATAPOINTS)
- void gnuplot (void)

  Creates a GNUplot graphic.

- void **addcommand** (const std::string &gcom)

Private Attributes

- std::string title

  Graph title.

- std::string xlabel

Graph x axis.

- std::string ylabel

  Graph y axis.

- std::string gnucommand

  GNU plot command.

- VectorCurves vCurves

4.16 Plot3d Class Reference

`#include <gnugraph.h>`

4.16.1 Detailed Description

3d plot object.

Definition at line 174 of file gnugraph.h.

Public Member Functions

- Plot3d ()

  Default constructor.

- void **settitle** (const std::string &t)
- void **setxlabel** (const std::string &t)
- void **setylabel** (const std::string &t)
- void **setzlabel** (const std::string &t)
- void gnuplot (const Matrix &xyz)

  Creates a GNUplot graphic.

Private Attributes

- std::string title

  Graph title.

- std::string xlabel

  Graph x axis.

- std::string ylabel

  Graph y axis.

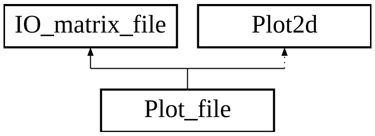
- std::string zlabel

  Graph z axis.

4.17 Plot_file Class Reference

`#include <gnugraph.h>`

Inheritance diagram for Plot_file::



4.17.1 Detailed Description

Creates a graphic from a data file.

Definition at line 223 of file gnugraph.h.

Public Member Functions

- **Plot_file** (const std::string &filename)
- short **graph** (const std::string &title_graph, const std::string &label, const short x, const short y, const short x_start, const short y_start, const short y_end)

Private Attributes

- std::vector< Matrix > data_from_file

  Data file.

- std::vector< std::string > data_title

  Data file title.

4.18 Proportional_Derivative Class Reference

`#include <controller.h>`

4.18.1 Detailed Description

Proportional derivative controller class.

The driving torques can be expressed as

$$\tau = K_p(q_d - q) + K_d(\dot{q}_d - q)$$

where K_p, K_d are diagonal positive definie matrix.

Definition at line 252 of file controller.h.

Public Member Functions

- Proportional_Derivative (const short dof=1)

  Constructor.

- Proportional_Derivative (const Robot_basic &robot, const DiagonalMatrix &Kp, const DiagonalMatrix &Kd)

  Constructor.

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

  Output torque.

- short set_Kd (const DiagonalMatrix &Kd)

  Assign the velocity error gain matrix $K_p(i,i)$.

- short set_Kp (const DiagonalMatrix &Kp)

  Assign the position error gain matrix $K_p(i,i)$.

Private Attributes

- int dof

  Degree of freedom.

- ColumnVector q

  Robot joints positions.

- ColumnVector qp

  Robot joints velocity.

- ColumnVector qpp

  Robot joints acceleration.

- ColumnVector tau

  Output torque.

- ColumnVector zero3

  3×1 *zero vector.*

- DiagonalMatrix Kp

  Position error gain.

- DiagonalMatrix Kd

  Velocity error gain.

4.18.2 Member Function Documentation

4.18.2.1 short Proportional_Derivative::set_Kd (const DiagonalMatrix & *Kd_*)

Assign the velocity error gain matrix $K_p(i,i)$.

Returns:

short: 0 or WRONG_SIZE if the matrix is not $dof \times dof$.

Definition at line 657 of file controller.cpp.

References dof, Kd, and WRONG_SIZE.

Referenced by Proportional_Derivative().

4.18.2.2 short Proportional_Derivative::set_Kp (const DiagonalMatrix & *Kp_*)

Assign the position error gain matrix $K_p(i,i)$.

Returns:

short: 0 or WRONG_SIZE if the matrix is not $dof \times dof$.

Definition at line 674 of file controller.cpp.

References dof, Kp, and WRONG_SIZE.

Referenced by Proportional_Derivative().

4.19 Quaternion Class Reference

`#include <quaternion.h>`

4.19.1 Detailed Description

Quaternion class definition.

Definition at line 92 of file quaternion.h.

Public Member Functions

- Quaternion ()

  Constructor.

- Quaternion (const Real angle_in_rad, const ColumnVector &axis)

  Constructor.

- Quaternion (const Real s, const Real v1, const Real v2, const Real v3)

  Constructor.

- Quaternion (const Matrix &R)

  Constructor.

- Quaternion operator+ (const Quaternion &q) const

  Overload + operator.

- Quaternion operator- (const Quaternion &q) const

  Overload - operator.

- Quaternion operator ∗ (const Quaternion &q) const

  Overload ∗ operator.

- Quaternion operator/ (const Quaternion &q) const

  Overload / operator.

- Quaternion conjugate () const

  Conjugate.

- Quaternion i () const

Quaternion inverse.

$$q^{-1} = \frac{q^*}{N(q)}$$

where q^* *and* $N(q)$ *are the quaternion conjugate and the quaternion norm respectively.*

- Quaternion & unit ()

  Normalize a quaternion.

- Quaternion exp () const

  Exponential of a quaternion.

- Quaternion power (const Real t) const
- Quaternion Log () const

  Logarithm of a unit quaternion.

- Quaternion dot (const ColumnVector &w, const short sign) const

  Quaternion time derivative.

- ReturnMatrix E (const short sign) const

  Matrix E.

- Real norm () const

  Return the quaternion norm.

- Real dot_prod (const Quaternion &q) const

  Quaternion dot product.

- Real s () const

  Return scalar part.

- void set_s (const Real s)

  Set scalar part.

- ReturnMatrix v () const

  Return vector part.

- void set_v (const ColumnVector &v)

  Set vector part.

- ReturnMatrix R () const

  Rotation matrix from a unit quaternion.

- ReturnMatrix T () const

  Transformation matrix from a quaternion.

Private Attributes

- Real s_

  Quaternion scalar part.

- ColumnVector v_

  Quaternion vector part.

4.19.2 Constructor & Destructor Documentation

4.19.2.1 Quaternion::Quaternion (const Matrix & *R*)

Constructor.

Cite_: Dam. The unit quaternion obtained from a matrix (see Quaternion::R())

$$R(s,v) = \begin{bmatrix} s^2+v_1^2-v_2^2-v_3^2 & 2v_1v_2+2sv_3 & 2v_1v_3-2sv_2 \\ 2v_1v_2-2sv_3 & s^2-v_1^2+v_2^2-v_3^2 & 2v_2v_3+2sv_1 \\ 2v_1v_3+2sv_2 & 2v_2v_3-2sv_1 & s^2-v_1^2-v_2^2+v_3^2 \end{bmatrix}$$

First we find s:

$$R_{11}+R_{22}+R_{33}+R_{44}=4s^2$$

Now the other values are:

$$s = \pm\frac{1}{2}\sqrt{R_{11}+R_{22}+R_{33}+R_{44}}$$

$$v_1 = \frac{R_{32}-R_{23}}{4s}$$

$$v_2 = \frac{R_{13}-R_{31}}{4s}$$

$$v_3 = \frac{R_{21}-R_{12}}{4s}$$

The sign of s cannot be determined. Depending on the choice of the sign for s the sign of v change as well. Thus the quaternions q and $-q$ represent the same rotation, but the interpolation curve changed with the choice of the sign. A positive sign has been chosen.

Definition at line 120 of file quaternion.cpp.

References EPSILON, i(), R(), s_, and v_.

4.19.3 Member Function Documentation

4.19.3.1 Quaternion Quaternion::operator+ (const Quaternion & *rhs*) const

Overload + operator.

The quaternion addition is

$$q_1 + q_2 = [s_1, v_1] + [s_2, v_2] = [s_1 + s_2, v_1 + v_2]$$

The result is not necessarily a unit quaternion even if q_1 and q_2 are unit quaternions.

Definition at line 203 of file quaternion.cpp.

References s_, and v_.

4.19.3.2 Quaternion Quaternion::operator- (const Quaternion & *rhs*) const

Overload - operator.

The quaternion soustraction is

$$q_1 - q_2 = [s_1, v_1] - [s_2, v_2] = [s_1 - s_2, v_1 - v_2]$$

The result is not necessarily a unit quaternion even if q_1 and q_2 are unit quaternions.

Definition at line 223 of file quaternion.cpp.

References s_, and v_.

4.19.3.3 Quaternion Quaternion::operator $*$ (const Quaternion & *rhs*) const

Overload $*$ operator.

The multiplication of two quaternions is

$$q = q_1 q_2 = [s_1 s_2 - v_1 \cdot v_2, v_1 \times v_2 + s_1 v_2 + s_2 v_1]$$

where $\cdot$ and $\times$ denote the scalar and vector product in R^3 respectively.

If q_1 and q_2 are unit quaternions, then q will also be a unit quaternion.

Definition at line 243 of file quaternion.cpp.

References s_, and v_.

4.19.3.4 Quaternion Quaternion::conjugate () const

Conjugate.

The conjugate of a quaternion $q = [s, v]$ is $q^* = [s, -v]$

Definition at line 283 of file quaternion.cpp.

References s_, and v_.

Referenced by i().

4.19.3.5 Quaternion Quaternion::exp () const

Exponential of a quaternion.

Let a quaternion of the form $q = [0, \theta v]$, q is not necessarily a unit quaternion. Then the exponential function is defined by $q = [\cos(\theta), v \sin(\theta)]$.

Definition at line 336 of file quaternion.cpp.

References EPSILON, s_, and v_.

Referenced by power().

4.19.3.6 Quaternion Quaternion::Log () const

Logarithm of a unit quaternion.

The logarithm function of a unit quaternion $q = [\cos(\theta), v \sin(\theta)]$ is defined as $log(q) = [0, v\theta]$. The result is not necessary a unit quaternion.

Definition at line 365 of file quaternion.cpp.

References EPSILON, s_, and v_.

Referenced by power().

4.19.3.7 Quaternion Quaternion::dot (const ColumnVector & *w*, const short *sign*) const

Quaternion time derivative.

The quaternion time derivative, quaternion propagation equation, is

$$\dot{s} = -\frac{1}{2}v^T w_{_0}$$

$$\dot{v} = \frac{1}{2}E(s, v)w_{_0}$$

$$E = sI - S(v)$$

where $w_{_0}$ is the angular velocity vector expressed in the base frame. If the vector is expressed in the object frame, $w_{_b}$, the time derivative becomes

$$\dot{s} = -\frac{1}{2}v^T w_{_b}$$

$$\dot{v} = \frac{1}{2}E(s,v)w_{_b}$$

$$E = sI + S(v)$$

Definition at line 388 of file quaternion.cpp.

References s_, sign(), and v_.

Referenced by Impedance::control().

4.19.3.8 ReturnMatrix Quaternion::E (const short *sign*) const

Matrix E.

See Quaternion::dot for explanation.

Definition at line 426 of file quaternion.cpp.

References BODY_FRAME, sign(), threebythreeident, and x_prod_matrix().

Referenced by Impedance::control(), and Omega().

4.19.3.9 Real Quaternion::norm () const

Return the quaternion norm.

The norm of quaternion is defined by

$$N(q) = s^2 + v \cdot v$$

Definition at line 298 of file quaternion.cpp.

References s_, and v_.

Referenced by i(), and unit().

4.19.3.10 Real Quaternion::dot_prod (const Quaternion & *q*) const

Quaternion dot product.

The dot product of quaternion is defined by

$$q_1 \cdot q_2 = s_1 s_2 + v_1 \cdot v_2$$

Definition at line 445 of file quaternion.cpp.

Referenced by Impedance::control(), and Resolved_acc::torque_cmd().

4.19.3.11 ReturnMatrix Quaternion::R () const

Rotation matrix from a unit quaternion.

$p' = qpq^{-1} = Rp$ where p is a vector, R a rotation matrix and q q quaternion. The rotation matrix obtained from a quaternion is then

$$R(s,v) = (s^2 - v^T v)I + 2vv^T - 2sS(v)$$

$$R(s,v) = \left[\begin{array}{ccc} s^2 + v_1^2 - v_2^2 - v_3^2 & 2v_1v_2 + 2sv_3 & 2v_1v_3 - 2sv_2 \\ 2v_1v_2 - 2sv_3 & s^2 - v_1^2 + v_2^2 - v_3^2 & 2v_2v_3 + 2sv_1 \\ 2v_1v_3 + 2sv_2 & 2v_2v_3 - 2sv_1 & s^2 - v_1^2 - v_2^2 + v_3^2 \end{array} \right]$$

where $S(\cdot)$ is the cross product matrix defined by

$$S(u) = \left[\begin{array}{ccc} 0 & -u_3 & u_2 \\ u_3 & 0 & -u_1 \\ -u_2 & u_1 & 0 \end{array} \right]$$

Definition at line 458 of file quaternion.cpp.

References s_, threebythreeident, v_, and x_prod_matrix().

Referenced by homogen_demo(), main(), Quaternion(), and Dynamics::set_robot_on_first_point_of_splines().

4.19.3.12 ReturnMatrix Quaternion::T () const

Transformation matrix from a quaternion.

See Quaternion::R() for equations.

Definition at line 499 of file quaternion.cpp.

References fourbyfourident, s_, v_, and x_prod_matrix().

Referenced by homogen_demo(), and main().

4.20 Resolved_acc Class Reference

`#include <controller.h>`

4.20.1 Detailed Description

Resolved rate acceleration controller class.

The dynamic model of a robot manipulator can be expressed in joint space as

$$B(q)\ddot{q} + C(q,\dot{q})\dot{q} + D\dot{q} + g(q) = \tau - J^T(q)f$$

According to the concept of inverse dynamics, the driving torques can be chosen as

$$\tau = B(q)J^{-1}(q)\big(a - \dot{J}(q,\dot{q})\dot{q}\big) + C(q,\dot{q})\dot{q} + D\dot{q} + g(q) - J^T(q)f$$

where a is the a new control input defined by

$$a_p = \ddot{p}_d + k_{vp}\dot{\tilde{p}} + k_{pp}\tilde{p}$$

$$a_o = \dot{\omega}_d + k_{vo}\dot{\tilde{\omega}} + k_{po}\tilde{v}$$

where $\tilde{x} = x_c - x_d$ and v is the vector par of the quaternion representing the orientation error between the desired and end effector frame. k_{vp}, k_{pp}, k_{vo} and k_{po} are positive gains.

Up to now this class has been tested only with a 6 dof robot.

Definition at line 171 of file controller.h.

Public Member Functions

- Resolved_acc (const short dof=1)

  Constructor.

- Resolved_acc (const Robot_basic &robot, const Real Kvp, const Real Kpp, const Real Kvo, const Real Kpo)

  Constructor.

- void set_Kvp (const Real Kvp)

  Assign the gain k_{vp}.

- void set_Kpp (const Real Kpp)

  Assign the gain k_{pp}.

- void set_Kvo (const Real Kvo)

Assign the gain k_{vo}.

- void set_Kpo (const Real Kpo)

  Assign the gain k_{po}.

- 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)

  Output torque.

Private Attributes

- double Kvp

  Controller gains.

- double Kpp
- double Kvo
- double Kpo
- Matrix Rot

  Temporay rotation matrix.

- ColumnVector zero3

  3×1 *zero vector.*

- ColumnVector qp

  Robot joints velocity.

- ColumnVector qpp

  Robot joints acceleration.

- ColumnVector a

  Control input.

- ColumnVector p

  End effector position.

- ColumnVector pp

  End effector velocity.

- ColumnVector quat_v_error

Vector part of error quaternion.

- Quaternion quat

  Temporary quaternion.

4.20.2 Member Function Documentation

4.20.2.1 ReturnMatrix Resolved_acc::torque_cmd (Robot_basic & *robot*, const ColumnVector & *pdpp*, const ColumnVector & *pdp*, const ColumnVector & *pd*, const ColumnVector & *wdp*, const ColumnVector & *wd*, const Quaternion & *quatd*, const short *link_pc*, const Real *dt*)

Output torque.

For more robustess the damped least squares inverse Jacobian is used instead of the inverse Jacobian.

The quaternion -q represents exactly the same rotation as the quaternion q. The temporay quaternion, quat, is quatd plus a sign correction. It is customary to choose the sign G on q1 so that q0.Gq1 >=0 (the angle between q0 ang Gq1 is acute). This choice avoids extra spinning caused by the interpolated rotations.

Definition at line 471 of file controller.cpp.

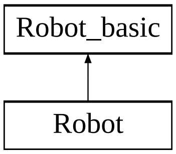
References a, Quaternion::dot_prod(), Robot_basic::get_dof(), Robot_basic::get_-q(), Robot_basic::get_qp(), Robot_basic::jacobian_DLS_inv(), Robot::jacobian_dot(), Robot::kine_pd(), Kpo, Kpp, Kvo, Kvp, p, pp, qp, qpp, quat, quat_v_error, robot, Rot, Quaternion::s(), Robot::torque(), Quaternion::v(), Robot_basic::w, x_prod_matrix(), and zero3.

Referenced by Dynamics::xdot().

4.21 Robot Class Reference

`#include <robot.h>`

Inheritance diagram for Robot::



4.21.1 Detailed Description

DH notation robot class.

Definition at line 340 of file robot.h.

Public Member Functions

- Robot (const int ndof=1)

  Constructor.

- Robot (const Matrix &initrobot)

  Constructor.

- Robot (const Matrix &initrobot, const Matrix &initmotor)

  Constructor.

- Robot (const Robot &x)

  Copy constructor.

- **Robot** (const std::string &filename, const std::string &robotName)
- virtual ~Robot ()

  Destructor.

- virtual void robotType_inv_kin ()

  Identify inverse kinematics familly.

- virtual void kine_pd (Matrix &Rot, ColumnVector &pos, ColumnVector &pos_dot, const int ref) const

  Direct kinematics with velocity.

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

  Overload inv_kin function.

- virtual ReturnMatrix inv_kin (const Matrix &Tobj, const int mj, const int endlink, bool &converge)

  Inverse kinematics solutions.

- virtual ReturnMatrix inv_kin_rhino (const Matrix &Tobj, bool &converge)

  Analytic Rhino inverse kinematics.

- virtual ReturnMatrix inv_kin_puma (const Matrix &Tobj, bool &converge)

  Analytic Puma inverse kinematics.

- virtual ReturnMatrix inv_kin_schilling (const Matrix &Tobj, bool &converge)

  Analytic Schilling inverse kinematics.

- virtual ReturnMatrix jacobian (const int ref=0) const

  Jacobian of mobile links expressed at frame ref.

- virtual ReturnMatrix jacobian (const int endlink, const int ref) const

  Jacobian of mobile links up to endlink expressed at frame ref.

- virtual ReturnMatrix jacobian_dot (const int ref=0) const

  Jacobian derivative of mobile joints expressed at frame ref.

- virtual void dTdqi (Matrix &dRot, ColumnVector &dp, const int i)

  Partial derivative of the robot position (homogeneous transf.).

- virtual ReturnMatrix dTdqi (const int i)

  Partial derivative of the robot position (homogeneous transf.).

- virtual ReturnMatrix torque (const ColumnVector &q, const ColumnVector &qp, const ColumnVector &qpp)

  Joint torque, without contact force, based on Recursive Newton-Euler formulation.

- virtual ReturnMatrix torque (const ColumnVector &q, const ColumnVector &qp, const ColumnVector &qpp, const ColumnVector &Fext_, const ColumnVector &Next_)

  Joint torque based on Recursive Newton-Euler formulation.

- virtual ReturnMatrix torque_novelocity (const ColumnVector &qpp)

  Joint torque. when joint velocity is 0, based on Recursive Newton-Euler formulation.

- virtual void delta_torque (const ColumnVector &q, const ColumnVector &qp, const ColumnVector &qpp, const ColumnVector &dq, const ColumnVector &dqp, const ColumnVector &dqpp, ColumnVector &ltorque, ColumnVector &dtorque)

  Delta torque dynamics.

- virtual void dq_torque (const ColumnVector &q, const ColumnVector &qp, const ColumnVector &qpp, const ColumnVector &dq, ColumnVector &torque, ColumnVector &dtorque)

  Delta torque due to delta joint position.

- virtual void dqp_torque (const ColumnVector &q, const ColumnVector &qp, const ColumnVector &dqp, ColumnVector &torque, ColumnVector &dtorque)

  Delta torque due to delta joint velocity.

- virtual ReturnMatrix G ()

  Joint torque due to gravity based on Recursive Newton-Euler formulation.

- virtual ReturnMatrix C (const ColumnVector &qp)

  Joint torque due to centrifugal and Corriolis based on Recursive Newton-Euler formulation.

4.21.2 Member Function Documentation

4.21.2.1 void Robot::robotType_inv_kin () `[virtual]`

Identify inverse kinematics familly.

Identify the inverse kinematics analytic solution based on the similarity of the robot DH parameters and the DH parameters of know robots (ex: Puma, Rhino, ...). The inverse kinematics will be based on a numerical alogorithm if there is no match .

Implements Robot_basic.

Definition at line 1284 of file robot.cpp.

References Robot_basic::DEFAULT, Robot_basic::PUMA, Puma_DH(), Robot_basic::RHINO, Rhino_DH(), Robot_basic::robotType, Robot_basic::SCHILLING, and Schilling_DH().

Referenced by Robot().

4.21.2.2 void Robot::kine_pd (Matrix & *Rot*, ColumnVector & *pos*, ColumnVector & *pos_dot*, const int *j*) const `[virtual]`

Direct kinematics with velocity.

Parameters:

Rot,: Frame j rotation matrix w.r.t to the base frame.

pos,: Frame j position vector wr.r.t to the base frame.

pos_dot,: Frame j velocity vector w.r.t to the base frame.

j,: Frame j. Print an error on the console if j is out of range.

Implements Robot_basic.

Definition at line 219 of file kinemat.cpp.

Referenced by Resolved_acc::torque_cmd().

4.21.2.3 ReturnMatrix Robot::inv_kin (const Matrix & *Tobj*, const int *mj*, const int *endlink*, bool & *converge*) `[virtual]`

Inverse kinematics solutions.

The solution is based on the analytic inverse kinematics if robot type (familly) is Rhino or Puma, otherwise used the numerical algoritm defined in Robot_basic class.

Reimplemented from Robot_basic.

Definition at line 204 of file invkine.cpp.

References Robot_basic::inv_kin(), inv_kin_puma(), inv_kin_rhino(), inv_kin_schilling(), Robot_basic::PUMA, Robot_basic::RHINO, Robot_basic::robotType, and Robot_basic::SCHILLING.

4.21.2.4 ReturnMatrix Robot::inv_kin_rhino (const Matrix & *Tobj*, bool & *converge*) `[virtual]`

Analytic Rhino inverse kinematics.

converge will be false if the desired end effector pose is outside robot range.

Implements Robot_basic.

Definition at line 229 of file invkine.cpp.

References Robot_basic::a, Link::a, Link::d, G(), Robot_basic::get_q(), K, Robot_basic::links, and M_PI.

Referenced by inv_kin().

4.21.2.5 ReturnMatrix Robot::inv_kin_puma (const Matrix & *Tobj*, bool & *converge*) `[virtual]`

Analytic Puma inverse kinematics.

converge will be false if the desired end effector pose is outside robot range.

Implements Robot_basic.

Definition at line 326 of file invkine.cpp.

References Robot_basic::a, Link::a, C(), Link::d, Robot_basic::get_q(), Robot_basic::links, and M_PI.

Referenced by inv_kin().

4.21.2.6 ReturnMatrix Robot::inv_kin_schilling (const Matrix & *Tobj*, bool & *converge*) `[virtual]`

Analytic Schilling inverse kinematics.

converge will be false if the desired end effector pose is outside robot range.

Implements Robot_basic.

Definition at line 486 of file invkine.cpp.

References Robot_basic::a, Link::a, C(), Link::d, Robot_basic::get_q(), K, Robot_basic::links, and M_PI.

Referenced by inv_kin().

4.21.2.7 ReturnMatrix Robot::jacobian (const int *endlink*, const int *ref*) const `[virtual]`

Jacobian of mobile links up to endlink expressed at frame ref.

The Jacobian expressed in based frame is

$$^0J(q) = \left[\begin{array}{cccc} ^0J_1(q) & ^0J_2(q) & \cdots & ^0J_n(q) \end{array} \right]$$

where $^0J_i(q)$ is defined by

$$^0J_i(q) = \left[\begin{array}{c} z_i \times ^i p_n \\ z_i \end{array} \right] \quad \text{rotoid joint}$$

$$^0J_i(q) = \left[\begin{array}{c} z_i \\ 0 \end{array} \right] \quad \text{prismatic joint}$$

Expressed in a different frame the Jacobian is obtained by

$$^iJ(q) = \left[\begin{array}{cc} ^0_iR^T & 0 \\ 0 & ^0_iR^T \end{array} \right] {}^0J(q)$$

Implements Robot_basic.

Definition at line 352 of file kinemat.cpp.

4.21.2.8 ReturnMatrix Robot::jacobian_dot (const int *ref* = `0`) const `[virtual]`

Jacobian derivative of mobile joints expressed at frame ref.

The Jacobian derivative expressed in based frame is

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

where $^0\dot{J}_i(q,\dot{q})$ is defined by

$$^0\dot{J}_i(q,\dot{q}) = \left[\begin{array}{c} \omega_{i-1} \times z_i \\ \omega_{i-1} \times ^{i-1}p_n + z_i \times ^{i-1}\dot{p}_n \end{array}\right] \quad \text{rotoid joint}$$

$$^0\dot{J}_i(q,\dot{q}) = \left[\begin{array}{c} 0 \\ 0 \end{array}\right] \quad \text{prismatic joint}$$

Expressed in a different frame the Jacobian derivative is obtained by

$$^iJ(q) = \left[\begin{array}{cc} ^0_iR^T & 0 \\ 0 & ^0_iR^T \end{array}\right] {^0J(q)}$$

Implements Robot_basic.

Definition at line 449 of file kinemat.cpp.

Referenced by Resolved_acc::torque_cmd().

4.21.2.9 void Robot::dTdqi (Matrix & *dRot*, ColumnVector & *dp*, const int *i*) `[virtual]`

Partial derivative of the robot position (homogeneous transf.).

This function computes the partial derivatives:

$$\frac{\partial{}^0T_n}{\partial q_i} = {}^0T_{i-1}Q_i\; {}^{i-1}T_n$$

in standard notation and

$$\frac{\partial{}^0T_n}{\partial q_i} = {}^0T_iQ_i\; {}^iT_n$$

in modified notation,

with

$$Q_i = \left[\begin{array}{cccc} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right]$$

for a revolute joint and

$$Q_i = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

for a prismatic joint.

$dRot$ and dp are modified on output.

Implements Robot_basic.

Definition at line 249 of file kinemat.cpp.

References Robot_basic::dof, Robot_basic::dp, Robot_basic::links, Robot_basic::p, Link::p, Robot_basic::R, Link::R, and threebythreeident.

Referenced by dTdqi(), and kinematics_demo().

4.21.2.10 ReturnMatrix Robot::dTdqi (const int *i*) `[virtual]`

Partial derivative of the robot position (homogeneous transf.).

See Robot::dTdqi(Matrix & dRot, ColumnVector & dp, const int i) for equations.

Implements Robot_basic.

Definition at line 334 of file kinemat.cpp.

References dTdqi().

4.21.2.11 ReturnMatrix Robot::torque (const ColumnVector & *q*, const ColumnVector & *qp*, const ColumnVector & *qpp*, const ColumnVector & *Fext*, const ColumnVector & *Next*) `[virtual]`

Joint torque based on Recursive Newton-Euler formulation.

In order to apply the RNE as presented in Murray 86, 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.

$z_0 = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$

$p_i = \begin{bmatrix} a_i & d_i \sin\alpha_i & d_i \cos\alpha_i \end{bmatrix}^T$ is the position of the i^{th} with respect to the $i-1^{th}$ frame.

Forward Iterations for $i = 1, 2, \ldots, n$. Initialize: $\omega_0 = \dot{\omega}_0 = 0$ and $\dot{v}_0 = -g$.

$$\omega_i = R_i^T[\omega_{i-1} + \sigma_i z_0 \dot{\theta}_i]$$

$$\dot{\omega}_i = R_i^T\{\dot{\omega}_{i-1} + \sigma_i[z_0\ddot{\theta}_i + \omega_{i-1} \times (z_0\dot{\theta}_i)]\}$$

$$\dot{v}_i = R_i^T\{\dot{v}_{i-1} + (1-\sigma_i)[z_0\ddot{d}_i + 2\omega_{i-1}\times(z_0\dot{d}_i)]\} + \dot{\omega}_i\times p_i + \omega_i\times(\omega_i\times p_i)$$

Backward Iterations for $i = n, n-1, \ldots, 1$. Initialize: $f_{n+1} = n_{n+1} = 0$.

$$\dot{v}_{ci} = v_i + \omega_i\times r_i + \omega_i\times(\omega_i\times r_i)$$

$$F_i = m_i\dot{v}_{ci}$$

$$N_i = I_{ci}\dot{\omega}_i + \omega_i\times(I_{ci}\omega_i)$$

$$f_i = R_{i+1}[f_{i+1}] + F_i$$

$$n_i = R_{i+1}[n_{i+1}] + p_i\times f_i + N_i + r_i\times F_i$$

$$\tau_i = \sigma_i n_i^T(R_i^T z_0) + (1-\sigma_i)f_i^T(R_i^T z_0)$$

Implements Robot_basic.

Definition at line 148 of file dynamics.cpp.

References Robot_basic::a, Link::B, Link::Cf, Robot_basic::dof, Robot_basic::f, Robot_basic::F, Link::Gr, Robot_basic::gravity, Link::I, Link::Im, Robot_basic::links, Robot_basic::n, Robot_basic::N, Robot_basic::p, Link::R, Robot_basic::R, Robot_basic::set_q(), Robot_basic::set_qp(), sign(), Robot_basic::vp, Robot_basic::w, Robot_basic::wp, and Robot_basic::z0.

4.21.2.12 void Robot::delta_torque (const ColumnVector & *q*, const ColumnVector & *qp*, const ColumnVector & *qpp*, const ColumnVector & *dq*, const ColumnVector & *dqp*, const ColumnVector & *dqpp*, ColumnVector & *ltorque*, ColumnVector & *dtorque*) `[virtual]`

Delta torque dynamics.

This function computes

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

Murray and Neuman Cite_: Murray86 have developed an efficient recursive linearized Newton-Euler formulation. In order to apply the RNE as presented in let us define the following variables

$$p_{di} = \frac{\partial p_i}{\partial d_i} = \begin{bmatrix} 0 & \sin\alpha_i & \cos\alpha_i \end{bmatrix}^T$$

$$Q = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Forward Iterations for $i = 1, 2, \ldots, n$. Initialize: $\delta\omega_0 = \delta\dot{\omega}_0 = \delta\dot{v}_0 = 0$.

$$\delta\omega_i = R_i^T\{\delta\omega_{i-1} + \sigma_i[z_0\delta\dot{\theta}_i - Q(\omega_{i-1} + \dot{\theta}_i)\delta\theta_i]\}$$

$$\delta\dot{\omega}_i = R_i^T\{\delta\dot{\omega}_{i-1}+\sigma_i[z_0\delta\ddot{\theta}_i+\delta\omega_{i-1}\times(z_0\dot{\theta}_i)+\omega_{i-1}\times(z_0\delta\dot{\theta}_i)]-\sigma_iQ[\omega_{i-1}+z_0\ddot{\theta}_i+\omega_{i-1}\times(z_0\dot{\theta}_i)]\delta\theta_i\}$$

$$\delta\dot{v}_i = R_i^T\{\delta\dot{v}_{i-1}-\sigma_iQ\dot{v}_{i-1}\delta\theta_i+(1-\sigma_i)[z_0\delta\ddot{d}_i+2\delta\omega_{i-1}\times(z_0\dot{d}_i)+2\omega_{i-1}\times(z_0\delta\dot{d}_i)]\}+\delta\dot{\omega}_i\times p_i+\delta\omega_i\times(\omega_i\times p_i)+\omega_i\times(\delta\omega_i\times p_i)+$$

Backward Iterations for $i = n, n-1, \ldots, 1$. Initialize: $\delta f_{n+1} = \delta n_{n+1} = 0$.

$$\delta\dot{v}_{ci} = \delta v_i + \delta\omega_i \times r_i + \delta\omega_i \times (\omega_i \times r_i) + \omega_i \times (\delta\omega_i \times r_i)$$

$$\delta F_i = m_i\delta\dot{v}_{ci}$$

$$\delta N_i = I_{ci}\delta\dot{\omega}_i + \delta\omega_i \times (I_{ci}\omega_i) + \omega_i \times (I_{ci}\delta\omega_i)$$

$$\delta f_i = R_{i+1}[\delta f_{i+1}] + \delta F_i + \sigma_{i+1}QR_{i+1}[f_{i+1}]\delta\theta_{i+1}$$

$$\delta n_i = R_{i+1}[\delta n_{i+1}]+\delta N_i+p_i\times\delta f_i+r_i\times\delta F_i+(1-\sigma_i)(p_{di}\times f_i)\delta d_i+\sigma_{i+1}QR_{i+1}[n_{i+1}]\delta\theta_{i+1}$$

$$\delta\tau_i = \sigma_i[\delta n_i^T(R_i^Tz_0) - n_i^T(R_i^TQz_0)\delta\theta_i] + (1-\sigma_i)[\delta f_i^T(R_i^Tz_0)]$$

Implements Robot_basic.

Definition at line 65 of file delta_t.cpp.

References Robot_basic::a, Robot_basic::da, Robot_basic::df, Robot_basic::dF, Robot_basic::dn, Robot_basic::dN, Robot_basic::dof, Robot_basic::dp, Robot_basic::dvp, Robot_basic::dw, Robot_basic::dwp, Robot_basic::f, Robot_basic::F, Robot_basic::gravity, Link::I, Robot_basic::links, Link::m, Robot_basic::n, Robot_basic::N, Robot_basic::p, Link::R, Robot_basic::R, Robot_basic::set_q(), Robot_basic::vp, Robot_basic::w, Robot_basic::wp, and Robot_basic::z0.

4.21.2.13 void Robot::dq_torque (const ColumnVector & *q*, const ColumnVector & *qp*, const ColumnVector & *qpp*, const ColumnVector & *dq*, ColumnVector & *ltorque*, ColumnVector & *dtorque*) `[virtual]`

Delta torque due to delta joint position.

This function computes $S_2(q, \dot{q}, \ddot{q})\delta q$. See Robot::delta_torque for equations.

Implements Robot_basic.

Definition at line 65 of file comp_dq.cpp.

References Robot_basic::a, Robot_basic::da, Robot_basic::df, Robot_basic::dF, Robot_basic::dn, Robot_basic::dN, Robot_basic::dof, Robot_basic::dp, Robot_basic::dvp, Robot_basic::dw, Robot_basic::dwp, Robot_basic::f, Robot_basic::F, Robot_basic::gravity, Link::I, Robot_basic::links, Link::m, Robot_basic::n, Robot_basic::N, Robot_basic::p, Link::R, Robot_basic::R, Robot_basic::set_q(), Robot_basic::vp, Robot_basic::w, Robot_basic::wp, and Robot_basic::z0.

4.21.2.14 void Robot::dqp_torque (const ColumnVector & *q*, const ColumnVector & *qp*, const ColumnVector & *dqp*, ColumnVector & *ltorque*, ColumnVector & *dtorque*) `[virtual]`

Delta torque due to delta joint velocity.

This function computes $S_1(q, \dot{q}, \ddot{q})\delta\dot{q}$. See Robot::delta_torque for equations.

Implements Robot_basic.

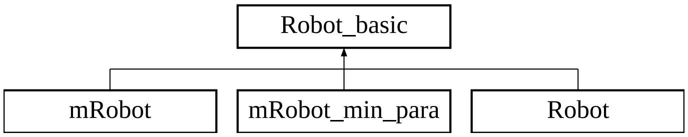
Definition at line 63 of file comp_dqp.cpp.

References Robot_basic::a, Robot_basic::da, Robot_basic::df, Robot_basic::dF, Robot_basic::dn, Robot_basic::dN, Robot_basic::dof, Robot_basic::dp, Robot_basic::dvp, Robot_basic::dw, Robot_basic::dwp, Robot_basic::f, Robot_basic::F, Robot_basic::gravity, Link::I, Robot_basic::links, Link::m, Robot_basic::n, Robot_basic::N, Robot_basic::p, Link::R, Robot_basic::R, Robot_basic::set_q(), Robot_basic::vp, Robot_basic::w, Robot_basic::wp, and Robot_basic::z0.

4.22 Robot_basic Class Reference

`#include <robot.h>`

Inheritance diagram for Robot_basic::



4.22.1 Detailed Description

Virtual base robot class.

Definition at line 216 of file robot.h.

Public Member Functions

- Robot_basic (const int ndof=1, const bool dh_parameter=false, const bool min_inertial_para=false)

  Constructor.

- Robot_basic (const Matrix &initrobot_motor, const bool dh_parameter=false, const bool min_inertial_para=false)

  Constructor.

- Robot_basic (const Matrix &initrobot, const Matrix &initmotor, const bool dh_parameter=false, const bool min_inertial_para=false)

  Constructor.

- **Robot_basic** (const std::string &filename, const std::string &robotName, const bool dh_parameter=false, const bool min_inertial_para=false)
- Robot_basic (const Robot_basic &x)

  Copy constructor.

- virtual ∼Robot_basic ()

  Destructor.

- Robot_basic & operator= (const Robot_basic &x)

  Overload = operator.

- Real get_q (const int i) const
- bool get_DH () const

  Return true if in DH notation, false otherwise.

- int get_dof () const

  Return dof.

- int get_available_dof () const

  Counts number of currently non-immobile links.

- int get_available_dof (const int endlink) const

  Counts number of currently non-immobile links up to and including endlink.

- int get_fix () const

  Return fix.

- ReturnMatrix get_q (void) const

  Return the joint position vector.

- ReturnMatrix get_qp (void) const

  Return the joint velocity vector.

- ReturnMatrix get_qpp (void) const

  Return the joint acceleration vector.

- ReturnMatrix get_available_q (void) const

  Return the joint position vector of available (non-immobile) joints.

- ReturnMatrix get_available_qp (void) const

  Return the joint velocity vector of available (non-immobile) joints.

- ReturnMatrix get_available_qpp (void) const

  Return the joint acceleration vector of available (non-immobile) joints.

- ReturnMatrix get_available_q (const int endlink) const

  Return the joint position vector of available (non-immobile) joints up to and including endlink.

- ReturnMatrix get_available_qp (const int endlink) const

  Return the joint velocity vector of available (non-immobile) joints up to and including endlink.

- ReturnMatrix get_available_qpp (const int endlink) const

Return the joint acceleration vector of available (non-immobile) joints up to and including endlink.

- void set_q (const ColumnVector &q)

  Set the joint position vector.

- void set_q (const Matrix &q)

  Set the joint position vector.

- void set_q (const Real q, const int i)
- void set_qp (const ColumnVector &qp)

  Set the joint velocity vector.

- void set_qpp (const ColumnVector &qpp)

  Set the joint acceleration vector.

- void kine (Matrix &Rot, ColumnVector &pos) const

  Direct kinematics at end effector.

- void kine (Matrix &Rot, ColumnVector &pos, const int j) const

  Direct kinematics at end effector.

- ReturnMatrix kine (void) const

  Return the end effector direct kinematics transform matrix.

- ReturnMatrix kine (const int j) const

  Return the frame j direct kinematics transform matrix.

- ReturnMatrix kine_pd (const int ref=0) const

  Direct kinematics with velocity.

- virtual void **kine_pd** (Matrix &Rot, ColumnVector &pos, ColumnVector &pos_dot, const int ref) const=0
- virtual void **robotType_inv_kin** ()=0
- virtual ReturnMatrix inv_kin (const Matrix &Tobj, const int mj=0)

  Numerical inverse kinematics. See inv_kin(const Matrix & Tobj, const int mj, const int endlink, bool & converge).

- ReturnMatrix inv_kin (const Matrix &Tobj, const int mj, bool &converge)
- virtual ReturnMatrix inv_kin (const Matrix &Tobj, const int mj, const int endlink, bool &converge)

  Numerical inverse kinematics.

- virtual ReturnMatrix **inv_kin_rhino** (const Matrix &Tobj, bool &converge)=0
- virtual ReturnMatrix **inv_kin_puma** (const Matrix &Tobj, bool &converge)=0
- virtual ReturnMatrix **inv_kin_schilling** (const Matrix &Tobj, bool &converge)=0
- virtual ReturnMatrix jacobian (const int ref=0) const

  Jacobian of mobile links expressed at frame ref.

- virtual ReturnMatrix **jacobian** (const int endlink, const int ref) const=0
- virtual ReturnMatrix **jacobian_dot** (const int ref=0) const=0
- ReturnMatrix jacobian_DLS_inv (const double eps, const double lambda_max, const int ref=0) const

  Inverse Jacobian based on damped least squares inverse.

- virtual void **dTdqi** (Matrix &dRot, ColumnVector &dp, const int i)=0
- virtual ReturnMatrix **dTdqi** (const int i)=0
- ReturnMatrix acceleration (const ColumnVector &q, const ColumnVector &qp, const ColumnVector &tau)

  Joints acceleration without contact force.

- ReturnMatrix acceleration (const ColumnVector &q, const ColumnVector &qp, const ColumnVector &tau, const ColumnVector &Fext, const ColumnVector &Next)

  Joints acceleration.

- ReturnMatrix inertia (const ColumnVector &q)

  Inertia of the manipulator.

- virtual ReturnMatrix **torque_novelocity** (const ColumnVector &qpp)=0
- virtual ReturnMatrix **torque** (const ColumnVector &q, const ColumnVector &qp, const ColumnVector &qpp)=0
- virtual ReturnMatrix **torque** (const ColumnVector &q, const ColumnVector &qp, const ColumnVector &qpp, const ColumnVector &Fext_, const ColumnVector &Next_)=0
- virtual 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)=0
- virtual void **dq_torque** (const ColumnVector &q, const ColumnVector &qp, const ColumnVector &qpp, const ColumnVector &dq, ColumnVector &torque, ColumnVector &dtorque)=0
- virtual void **dqp_torque** (const ColumnVector &q, const ColumnVector &qp, const ColumnVector &dqp, ColumnVector &torque, ColumnVector &dtorque)=0

- ReturnMatrix dtau_dq (const ColumnVector &q, const ColumnVector &qp, const ColumnVector &qpp)

  Sensitivity of the dynamics with respect to q.

- ReturnMatrix dtau_dqp (const ColumnVector &q, const ColumnVector &qp)

  Sensitivity of the dynamics with respect to $\dot{q}$.

- virtual ReturnMatrix **G** ()=0
- virtual ReturnMatrix **C** (const ColumnVector &qp)=0
- void **error** (const std::string &msg1) const

Public Attributes

- ColumnVector ∗ w
- ColumnVector ∗ wp
- ColumnVector ∗ vp
- ColumnVector ∗ a
- ColumnVector ∗ f
- ColumnVector ∗ f_nv
- ColumnVector ∗ n
- ColumnVector ∗ n_nv
- ColumnVector ∗ F
- ColumnVector ∗ N
- ColumnVector ∗ p
- ColumnVector ∗ pp
- ColumnVector ∗ dw
- ColumnVector ∗ dwp
- ColumnVector ∗ dvp
- ColumnVector ∗ da
- ColumnVector ∗ df
- ColumnVector ∗ dn
- ColumnVector ∗ dF
- ColumnVector ∗ dN
- ColumnVector ∗ dp
- ColumnVector z0

  Axis vector at each joint.

- ColumnVector gravity

  Gravity vector.

- Matrix ∗ R

  Temprary rotation matrix.

- Link ∗ links

  Pointer on Link cclass.

Private Types

- enum EnumRobotType { DEFAULT = 0, RHINO = 1, PUMA = 2, SCHILLING = 3 }

  enum EnumRobotType

Private Member Functions

- void cleanUpPointers ()

  Free all vectors and matrix memory.

Private Attributes

- EnumRobotType robotType

  Robot type.

- int dof

  Degree of freedom.

- int fix

  Virtual link, used with modified DH notation.

Friends

- class Robot
- class mRobot
- class mRobot_min_para
- class Robotgl
- class mRobotgl

4.22.2 Member Enumeration Documentation

4.22.2.1 enum Robot_basic::EnumRobotType `[private]`

enum EnumRobotType

Enumerator:

DEFAULT Default robot familly.

RHINO Rhino familly.

PUMA Puma familly.

SCHILLING Schilling familly.

Definition at line 324 of file robot.h.

4.22.3 Constructor & Destructor Documentation

4.22.3.1 Robot_basic::Robot_basic (const int *ndof* = `1`, const bool *dh_parameter* = `false`, const bool *min_inertial_para* = `false`)

Constructor.

Parameters:

ndof,: Robot degree of freedom.

dh_parameter,: true if DH notation, false if modified DH notation.

min_inertial_para,: true inertial parameter are in minimal form.

Allocate memory for vectors and matrix pointers. Initialize all the Links instance.

Definition at line 573 of file robot.cpp.

References a, cleanUpPointers(), da, dF, df, dN, dn, dof, dp, dvp, dw, dwp, F, f, f_nv, fix, GRAVITY, gravity, links, N, n, n_nv, p, pp, R, threebythreeident, vp, w, wp, and z0.

4.22.3.2 Robot_basic::Robot_basic (const Matrix & *dhinit*, const bool *dh_parameter* = `false`, const bool *min_inertial_para* = `false`)

Constructor.

Parameters:

dhinit,: Robot initialization matrix.

dh_parameter,: true if DH notation, false if modified DH notation.

min_inertial_para,: true inertial parameter are in minimal form.

Allocate memory for vectors and matrix pointers. Initialize all the Links instance.

Definition at line 343 of file robot.cpp.

References a, cleanUpPointers(), da, dF, df, dN, dn, dof, dp, dvp, dw, dwp, F, f, f_nv, fix, GRAVITY, gravity, links, N, n, n_nv, p, pp, R, threebythreeident, vp, w, wp, and z0.

4.22.3.3 Robot_basic::Robot_basic (const Matrix & *initrobot*, const Matrix & *initmotor*, const bool *dh_parameter* = `false`, const bool *min_inertial_para* = `false`)

Constructor.

Parameters:

initrobot,: Robot initialization matrix.

initmotor,: Motor initialization matrix.

dh_parameter,: true if DH notation, false if modified DH notation.

min_inertial_para,: true inertial parameter are in minimal form.

Allocate memory for vectors and matrix pointers. Initialize all the Links instance.

Definition at line 451 of file robot.cpp.

References a, cleanUpPointers(), da, dF, df, dN, dn, dof, dp, dvp, dw, dwp, F, f, f_nv, fix, GRAVITY, gravity, links, N, n, n_nv, p, pp, R, threebythreeident, vp, w, wp, and z0.

4.22.3.4 Robot_basic::∼Robot_basic () `[virtual]`

Destructor.

Free all vectors and matrix memory.

Definition at line 879 of file robot.cpp.

References cleanUpPointers().

4.22.4 Member Function Documentation

4.22.4.1 Real Robot_basic::get_q (const int *i*) const `[inline]`

Definition at line 235 of file robot.h.

Referenced by Clik::Clik(), dynamics_demo(), kinematics_demo(), Dynamics::set_robot_on_first_point_of_splines(), Proportional_Derivative::torque_cmd(), Computed_torque_method::torque_cmd(), and Resolved_acc::torque_cmd().

4.22.4.2 void Robot_basic::set_q (const ColumnVector & *q*)

Set the joint position vector.

Set the joint position vector and recalculate the orientation matrix R and the position vector p (see Link class). Print an error if the size of q is incorrect.

Definition at line 1137 of file robot.cpp.

References dof, get_available_dof(), links, Link::p, p, and R.

Referenced by mRobot_min_para::delta_torque(), mRobot::delta_torque(), Robot::delta_torque(), mRobot_min_para::dq_torque(), mRobot::dq_torque(), Robot::dq_torque(), mRobot_min_para::dqp_torque(), mRobot::dqp_torque(), Robot::dqp_torque(), dynamics_demo(), Clik::endeff_pos_ori_err(), inertia(), inv_kin(), kinematics_demo(), main(), Clik::q_qdot(), Dynamics::set_robot_on_first_point_of_splines(), mRobot_min_para::torque(), mRobot::torque(), Robot::torque(), and Dynamics::xdot().

4.22.4.3 void Robot_basic::set_q (const Matrix & *q*)

Set the joint position vector.

Set the joint position vector and recalculate the orientation matrix R and the position vector p (see Link class). Print an error if the size of q is incorrect.

Definition at line 1070 of file robot.cpp.

References dof, get_available_dof(), links, Link::p, p, and R.

4.22.4.4 void Robot_basic::set_q (const Real *q*, const int *i*) `[inline]`

Definition at line 255 of file robot.h.

4.22.4.5 void Robot_basic::kine (Matrix & *Rot*, ColumnVector & *pos*) const

Direct kinematics at end effector.

Parameters:

Rot,: End effector orientation.

pos,: Enf effector position.

Definition at line 92 of file kinemat.cpp.

Referenced by Clik::endeff_pos_ori_err(), Impedance::Impedance(), kinematics_demo(), and main().

4.22.4.6 void Robot_basic::kine (Matrix & *Rot*, ColumnVector & *pos*, const int *j*) const

Direct kinematics at end effector.

Parameters:

Rot,: Frame j orientation.

pos,: Frame j position.

j,: Selected frame.

Definition at line 102 of file kinemat.cpp.

4.22.4.7 ReturnMatrix Robot_basic::kine_pd (const int *j* = `0`) const

Direct kinematics with velocity.

Return a 3×5 matrix. The first three columns are the frame j to the base rotation, the fourth column is the frame j w.r.t to the base postion vector and the last column is the frame j w.r.t to the base translational velocity vector. Print an error on the console if j is out of range.

Definition at line 142 of file kinemat.cpp.

4.22.4.8 ReturnMatrix Robot_basic::inv_kin (const Matrix & *Tobj*, const int *mj*, const int *endlink*, bool & *converge*) `[virtual]`

Numerical inverse kinematics.

Parameters:

Tobj,: Transformation matrix expressing the desired end effector pose.

mj,: Select algorithm type, 0: based on Jacobian, 1: based on derivative of T.

converge,: Indicate if the algorithm converge.

endlink,: the link to pretend is the end effector

The joint position vector before the inverse kinematics is returned if the algorithm does not converge.

Reimplemented in Robot, mRobot, and mRobot_min_para.

Definition at line 91 of file invkine.cpp.

References dof, get_available_dof(), get_available_q(), ITOL, jacobian(), kine(), links, M_PI, NITMAX, and set_q().

4.22.4.9 ReturnMatrix Robot_basic::jacobian_DLS_inv (const double *eps*, const double *lambda_max*, const int *ref* = 0) const

Inverse Jacobian based on damped least squares inverse.

Parameters:

- ***eps,:*** Range of singular region.
- ***lambda_max,:*** Value to obtain a good solution in singular region.
- ***ref,:*** Selected frame (ex: joint 4).

The Jacobian inverse, based on damped least squares, is

$$J^{-1}(q) = \left(J^T(q)J(q) + \lambda^2 I\right)^{-1} J^T(q)$$

where λ and I is a damping factor and the identity matrix respectively. Based on SVD (Singular Value Decomposition) the Jacobian is $J = \sum_{i=1}^{m} \sigma_i u_i v_i^T$, where u_i, v_i and σ_i are respectively the input vector, the ouput vector and the singular values ($\sigma_1 \geq \sigma_2 \cdots \geq \sigma_r \geq 0$, r is the rank of J). Using the previous equations we obtain

$$J^{-1}(q) = \sum_{i=1}^{m} \frac{\sigma_i}{\sigma_i^2 + \lambda_i^2} v_i u_i$$

A singular region, based on the smallest singular value, can be defined by

$$\lambda^2 = \begin{cases} 0 & \text{si } \sigma_6 \geq \epsilon \\ \Big(1 - (\frac{\sigma_6}{\epsilon})^2\Big)\lambda_{max}^2 & \text{sinon} \end{cases}$$

Definition at line 169 of file kinemat.cpp.

Referenced by Clik::q_qdot(), and Resolved_acc::torque_cmd().

4.22.4.10 ReturnMatrix Robot_basic::acceleration (const ColumnVector & *q*, const ColumnVector & *qp*, const ColumnVector & *tau_cmd*, const ColumnVector & *Fext*, const ColumnVector & *Next*)

Joints acceleration.

The robot dynamics is

$$B(q)\ddot{q} + C(q,\dot{q})\dot{q} + D\dot{q} + g(q) = \tau - J^T(q)f$$

then the joint acceleration is

$$\ddot{q} = B^{-1}(q)\big(\tau - J^T(q)f - C(q,\dot{q})\dot{q} - D\dot{q} - g(q)\big)$$

Definition at line 112 of file dynamics.cpp.

References dof, and inertia().

4.22.4.11 ReturnMatrix Robot_basic::dtau_dq (const ColumnVector & *q*, const ColumnVector & *qp*, const ColumnVector & *qpp*)

Sensitivity of the dynamics with respect to q.

This function computes $S_2(q, \dot{q}, \ddot{q})$.

Definition at line 58 of file sensitiv.cpp.

References dof.

4.22.4.12 ReturnMatrix Robot_basic::dtau_dqp (const ColumnVector & *q*, const ColumnVector & *qp*)

Sensitivity of the dynamics with respect to $\dot{q}$.

This function computes $S_1(q, \dot{q})$.

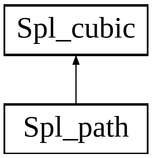
Definition at line 84 of file sensitiv.cpp.

References dof.

4.23 Spl_cubic Class Reference

`#include <trajectory.h>`

Inheritance diagram for Spl_cubic::



4.23.1 Detailed Description

Natural cubic splines class.

Definition at line 91 of file trajectory.h.

Public Member Functions

- Spl_cubic ()
- Spl_cubic (const Matrix &pts)

  Constructor.

- short interpolating (const Real t, ColumnVector &s)

  Interpolating the spline at time t. Extrapolating is not allowed.

- short first_derivative (const Real t, ColumnVector &ds)

  Spline first derivative at time t.

- short second_derivative (const Real t, ColumnVector &dds)

  Spline second derivative at time t.

Private Attributes

- int nb_path

  Number of path, i.e: path in x,y,z nb_path=3.

- Matrix Ak
- Matrix Bk
- Matrix Ck

- Matrix Dk
- RowVector tk

  Time at control points.

- bool bad_data

  Status flag.

4.23.2 Constructor & Destructor Documentation

4.23.2.1 Spl_cubic::Spl_cubic (const Matrix & *pts*)

Constructor.

Parameters:

pts,: Matrix containing the spline data.

The first line of the Matrix contain the sampling time Second line contain data (sk) to create spline i. Third " " i+1. on Nth line i+N.

The spline has the following form:

$$s = A_k(t - t_k)^3 + B_k(t - t_k)^2 + C_k(t - t_k) + D_k$$

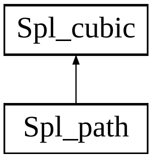
Definition at line 58 of file trajectory.cpp.

References Ak, bad_data, Bk, Ck, Dk, nb_path, and tk.

4.24 Spl_path Class Reference

`#include <trajectory.h>`

Inheritance diagram for Spl_path::



4.24.1 Detailed Description

Cartesian or joint space trajectory.

Definition at line 120 of file trajectory.h.

Public Member Functions

- Spl_path ()
- **Spl_path** (const std::string &filename)
- Spl_path (const Matrix &x)

  Constructor.

- short p (const Real time, ColumnVector &p)

  Position vector at time t.

- short p_pdot (const Real time, ColumnVector &p, ColumnVector &pdot)

  Position and velocity vector at time t.

- short p_pdot_pddot (const Real time, ColumnVector &p, ColumnVector &pdot, ColumnVector &pdotdot)

  Position, velocity and acceleration vector at time t.

- short get_type ()
- double get_final_time ()

Private Attributes

- short type

  Cartesian space or joint space.

- double final_time

  Spline final time.

4.25 Spl_Quaternion Class Reference

`#include <trajectory.h>`

4.25.1 Detailed Description

Cubic quaternions spline.

Definition at line 147 of file trajectory.h.

Public Member Functions

- Spl_Quaternion ()
- **Spl_Quaternion** (const std::string &filename)
- Spl_Quaternion (const quat_map &quat)

  Constructor.

- short quat (const Real t, Quaternion &s)

  Quaternion interpollation.

- short quat_w (const Real t, Quaternion &s, ColumnVector &w)

  Quaternion interpollation and angular velocity.

Private Attributes

- quat_map quat_data

  Data at control points.

4.25.2 Member Function Documentation

4.25.2.1 short Spl_Quaternion::quat (const Real *t*, Quaternion & *s*)

Quaternion interpollation.

$$S_n(t) = Squad(q_n, a_n, a(n+1), q(n+1), t)$$

Definition at line 503 of file trajectory.cpp.

References NOT_IN_RANGE, quat_data, Slerp(), and Squad().

4.26 Stewart Class Reference

`#include <stewart.h>`

4.26.1 Detailed Description

Stewart definitions.

Definition at line 143 of file stewart.h.

Public Member Functions

- Stewart ()

  Default Constructor.

- Stewart (const Matrix InitPlat, bool Joint=true)

  Constructor.

- Stewart (const Stewart &x)

  Copy Constructor.

- **Stewart** (const std::string &filename, const std::string &PlatformName)
- ~Stewart ()

  Destructor.

- const Stewart & operator= (const Stewart &x)
- void set_Joint (const bool _Joint)

  Set the position of the universal joint on the links.

- void set_q (const ColumnVector _q)

  Set the position of the platform.

- void set_dq (const ColumnVector _dq)

  Set the platform's speed.

- void set_ddq (const ColumnVector _ddq)

  Set the platform's acceleration.

- void set_pR (const ColumnVector _pR)

  Set the position of the center of mass of the platform.

- void set_pIp (const Matrix _pIp)

Set the inertia matrix of the platform.

- void set_mp (const Real _mp)

  Set the mass of the platform.

- bool get_Joint () const

  Return the position of the universal joint (true if at base, false if at platform).

- ReturnMatrix get_q () const

  Return the position of the platform.

- ReturnMatrix get_dq () const

  Return the speed of the platform.

- ReturnMatrix get_ddq () const

  Return the acceleration of the platform.

- ReturnMatrix get_pR () const

  Return the postion of the center of mass of the platfom.

- ReturnMatrix get_pIp () const

  Return the inertia matrix of the platform.

- Real get_mp () const

  Return the mass of the platform.

- void Transform ()

  Call the functions corresponding to the basic parameters when q changes.

- ReturnMatrix Find_wRp ()

  Return the rotation matrix wRp.

- ReturnMatrix Find_Omega ()

  Return the angular speed of the platform.

- ReturnMatrix Find_Alpha ()

  Return the angular acceleration of the platform.

- ReturnMatrix jacobian ()

  Return the jacobian matrix of the platform.

- ReturnMatrix Find_InvJacob1 ()

Return the first intermediate jacobian matrix (reverse) of the platform.

- ReturnMatrix Find_InvJacob2 ()

  Return the second intermediate jacobian matrix (reverse) of the platform.

- ReturnMatrix jacobian_dot ()

  Return time deriative of the inverse jacobian matrix of the platform.

- ReturnMatrix Find_dl ()

  Return the extension rate of the links in a vector.

- ReturnMatrix Find_ddl ()

  Return the extension acceleration of the links in a vector.

- ReturnMatrix Find_C (const Real Gravity=GRAVITY)

  Return intermediate matrix C for the dynamics calculations.

- ReturnMatrix Torque (const Real Gravity=GRAVITY)

  Return the torque vector of the platform.

- ReturnMatrix JointSpaceForceVct (const Real Gravity=GRAVITY)

  Return a vector containing the six actuation force components.

- ReturnMatrix InvPosKine ()

  Return the lenght of the links in a vector.

- ReturnMatrix ForwardKine (const ColumnVector guess_q, const ColumnVector l_given, const Real tolerance=0.001)

  Return the position vector of the platform (vector q).

- ReturnMatrix Find_h (const Real Gravity=GRAVITY)

  Return the intermediate matrix corresponding to the Coriolis and centrifugal + gravity force/torque components.

- ReturnMatrix Find_M ()

  Return the intermediate matrix corresponding to the inertia matrix of the machine.

- ReturnMatrix ForwardDyn (const ColumnVector Torque, const Real Gravity=GRAVITY)

  Return the acceleration vector of the platform (ddq).

- void Find_Mc_Nc_Gc (Matrix &Mc, Matrix &Nc, Matrix &Gc)

  Return(!) the intermediates matrix for forward dynamics with actuator dynamics.

- ReturnMatrix ForwardDyn_AD (const ColumnVector Command, const Real t)

  Return the acceleration of the platform (Stewart platform mechanism dynamics including actuator dynamics).

Public Attributes

- Matrix wRp

  Rotation matrix describing the orientation of the platform.

- Matrix Jacobian

  Jacobian matrix.

- Matrix IJ1

  Inverse of the first intermediate Jacobian matrix.

- Matrix IJ2

  Inverse of the second intermediate Jacobian matrix.

- ColumnVector dl

  Rate of expension vector.

- ColumnVector ddl

  Acceleration of expension vector.

- ColumnVector Alpha

  Angular speed of the platform.

- ColumnVector Omega

  Angular acceleration of the platform.

Private Attributes

- bool UJointAtBase

  Gives the position of the universal joint (true if at base, false if at platform).

- ColumnVector q

  Platform position (xyz + euler angles).

- ColumnVector dq

Platform speed.

- ColumnVector ddq

  Platform acceleration.

- ColumnVector pR

  Platform center of mass (in its own referential).

- ColumnVector gravity

  Gravity vector.

- Matrix pIp

  Platform Inertia (local ref.).

- Real mp

  Platform mass.

- Real p

  Pitch of the ballscrew (links).

- Real n

  Gear ratio (links motor).

- Real Js

  Moment of inertia (ballscrew).

- Real Jm

  Moment of inertia (motor).

- Real bs

  Viscous damping coefficient of the ballscrew.

- Real bm

  Viscous damping coefficient of the motor.

- Real Kb

  Motor back EMF.

- Real L

  Motor Inductance.

- Real R

Motor armature resistance.

- Real Kt

  Motor torque.

- LinkStewart Links [6]

  Platform links.

4.26.2 Constructor & Destructor Documentation

4.26.2.1 Stewart::Stewart (const Matrix *InitPlatt*, bool *Joint* = `true`)

Constructor.

Parameters:

InitPlatt,: Platform initialization matrix.

Joint,: bool indicating where is the universal joint

Definition at line 853 of file stewart.cpp.

References bm, bs, ddq, dq, Find_wRp(), gravity, Jm, Js, Kb, Kt, L, Links, M_PI, mp, n, p, pIp, pR, q, R, UJointAtBase, and wRp.

4.26.3 Member Function Documentation

4.26.3.1 const Stewart & Stewart::operator= (const Stewart & *x*)

Definition at line 1004 of file stewart.cpp.

References bm, bs, ddq, dq, gravity, Jm, Js, Kb, Kt, L, Links, mp, n, p, pIp, pR, q, R, and UJointAtBase.

4.26.3.2 void Stewart::Transform ()

Call the functions corresponding to the basic parameters when q changes.

These functions are called by Transform:

- Find_wRp()
- LinkStewart::LTransform() for each link
- Find_InvJacob1()

- Find_InvJacob2()
- jacobian()

Definition at line 1160 of file stewart.cpp.

References Find_InvJacob1(), Find_InvJacob2(), Find_wRp(), IJ1, IJ2, jacobian(), Jacobian, Links, q, and wRp.

Referenced by set_q().

4.26.3.3 ReturnMatrix Stewart::Find_wRp ()

Return the rotation matrix wRp.

Eq of the matrix:

$$wRp = \left(\begin{array}{ccc} \cos(\psi)\cos(\phi) - \cos(\theta)\sin(\phi)\sin(\psi) & -\sin(\psi)\cos(\phi) - \cos(\theta)\sin(\phi)\cos(\psi) & \sin(\theta)\sin(\phi) \\ \cos(\psi)\sin(\phi) + \cos(\theta)\cos(\phi)\sin(\psi) & -\sin(\psi)\sin(\phi) + \cos(\theta)\cos(\phi)\cos(\psi) & -\sin(\theta)\cos(\phi) \\ \sin(\psi)\sin(\theta) & \cos(\psi)\sin(\theta) & \cos(\theta) \end{array} \right)$$

Where:

- ψ, θ, ϕ, are the three Euler angles of the platform.

Definition at line 1186 of file stewart.cpp.

References q.

Referenced by Stewart(), and Transform().

4.26.3.4 ReturnMatrix Stewart::Find_Omega ()

Return the angular speed of the platform.

Eq:

$$\omega = \left(\begin{array}{ccc} 0 & \cos(\phi) & \cos(\theta)\sin(\phi) \\ 0 & \sin(\phi) & -\sin(\theta)\cos(\phi) \\ 1 & 0 & \cos(\theta) \end{array} \right) \left(\begin{array}{c} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{array} \right)$$

Where:

- ψ, θ, ϕ are the three Euler angles of the platform.
- $\dot{\psi}, \dot{\theta}, \dot{\phi}$ are the three Euler angle speed of the platform.

Definition at line 1223 of file stewart.cpp.

References dq, and q.

Referenced by set_ddq(), and set_dq().

4.26.3.5 ReturnMatrix Stewart::Find_Alpha ()

Return the angular acceleration of the platform.

Eq:

$$\alpha = \left(\begin{array}{ccc} 0 & \cos(\phi) & \cos(\theta)\sin(\phi) \\ 0 & \sin(\phi) & -\sin(\theta)\cos(\phi) \\ 1 & 0 & \cos(\theta) \end{array}\right)\left(\begin{array}{c} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{array}\right) + \left(\begin{array}{ccc} 0 & -\phi\sin(\phi) & \phi\cos(\phi)\sin(\theta) + \dot{\theta}\sin(\phi)\cos(\theta) \\ 0 & \phi\cos(\phi) & \phi\sin(\phi)\sin(\theta) - \dot{\theta}\cos(\phi)\cos(\theta) \\ 0 & 0 & -th\dot{e}ta\sin(\theta) \end{array}\right)\left(\begin{array}{c} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{array}\right)$$

Where:

- ψ, θ, ϕ are the three Euler angles of the platform.
- $\dot{\psi}, \dot{\theta}, \dot{\phi}$ are the three Euler angle speed of the platform.
- $\ddot{\psi}, \ddot{\theta}, \ddot{\phi}$ are the three Euler angle acceleration of the platform.

Definition at line 1261 of file stewart.cpp.

References ddq, dq, and q.

Referenced by set_ddq().

4.26.3.6 ReturnMatrix Stewart::jacobian ()

Return the jacobian matrix of the platform.

Eq:

$J = J_1^{-1} J_2^{-1}$

Where:

- J_1 and J_2 are intermediate matrix(Find_InvJacob1(), Find_InvJacob2())

Definition at line 1290 of file stewart.cpp.

References IJ1, and IJ2.

Referenced by Transform().

4.26.3.7 ReturnMatrix Stewart::Find_InvJacob1 ()

Return the first intermediate jacobian matrix (reverse) of the platform.

Eq:

$$J_1^{-1} = \begin{pmatrix} n_1^T & (a_{w1} \times n_1)^T \\ \vdots & \vdots \\ n_6^T & (a_{w6} \times n_6)^T \end{pmatrix}$$

Where:

- n_1 to n_6 are the unit vector of the links
- a_{w1} to a_{w6} are the attachment point of the links to the platform in the world referential

Definition at line 1316 of file stewart.cpp.

References Links, LinkStewart::UnitV, and wRp.

Referenced by JointSpaceForceVct(), and Transform().

4.26.3.8 ReturnMatrix Stewart::Find_InvJacob2 ()

Return the second intermediate jacobian matrix (reverse) of the platform.

Eq:

$$J_2^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cos\phi & \sin\phi\sin\theta \\ 0 & 0 & 0 & 0 & \sin\phi & -\cos\phi\sin\theta \\ 0 & 0 & 0 & 1 & 0 & \cos\theta \end{pmatrix}$$

Where:

- ϕ and θ are two of the euler angle of the platform (vector q)

Definition at line 1344 of file stewart.cpp.

References q.

Referenced by Transform().

4.26.3.9 ReturnMatrix Stewart::jacobian_dot ()

Return time deriative of the inverse jacobian matrix of the platform.

Eq:

$$\frac{dJ^{-1}}{dt} = \frac{dJ_1^{-1}}{dt}J_2^{-1} + J_1^{-1}\frac{dJ_2^{-1}}{dt}$$

$$\frac{dJ_1^{-1}}{dt} = \begin{pmatrix} (\omega_1 \times n_1)^T & ((\omega \times a_{w1}) \times n_1 + a_{w1} \times (\omega_1 \times n_1))^T \\ \vdots & \vdots \\ (\omega_6 \times n_6)^T & ((\omega \times a_{w6}) \times n_6 + a_{w6} \times (\omega_6 \times n_6))^T \end{pmatrix}$$

$$\frac{dJ_2^{-1}}{dt} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -\dot{\phi}\sin\phi & \dot{\phi}\cos\phi\sin\theta + \dot{\theta}\sin\phi\cos\theta \\ 0 & 0 & 0 & 0 & \dot{\phi}\cos\phi & \dot{\phi}\sin\phi\sin\theta + \dot{\theta}\cos\phi\cos\theta \\ 0 & 0 & 0 & 0 & 0 & -\dot{\theta}\sin\theta \end{pmatrix}$$

Where:

- ω_i is the angular speed vector of each link
- n is the unit vector of the link
- ω is the angular speed vector of the platform
- a_{wi} is the position vector of the attachment point of the link to the platform
- ϕ and θ are two of the Euler angle (vector q)
- $\dot{\phi}$ and $\dot{\theta}$ are two of the Euler angle speed (vector dq)

Definition at line 1389 of file stewart.cpp.

References dl, dq, IJ1, IJ2, L, LinkStewart::L, Links, Omega, q, and LinkStewart::UnitV.

Referenced by Find_ddl(), and Find_Mc_Nc_Gc().

4.26.3.10 ReturnMatrix Stewart::Find_dl ()

Return the extension rate of the links in a vector.

Eq:

$\dot{l} = J^{-1}\dot{q}$

Where:

- J^{-1} is the inverse Jacobian matrix of the platform
- $\dot{q}$ is the dq vector

Definition at line 1451 of file stewart.cpp.

References dq, and Jacobian.

Referenced by set_dq().

4.26.3.11 ReturnMatrix Stewart::Find_ddl ()

Return the extension acceleration of the links in a vector.

Eq:

$\ddot{l} = J^{-1}\ddot{q} + \frac{dJ^{-1}}{dt}\dot{q}$

Where:

- J^{-1} is the inverse jacobian matrix of the platform
- $\ddot{q}$ is the ddq vector

Definition at line 1472 of file stewart.cpp.

References ddq, dq, Jacobian, and jacobian_dot().

Referenced by set_ddq(), and set_dq().

4.26.3.12 ReturnMatrix Stewart::Find_C (const Real *Gravity* = `GRAVITY`)

Return intermediate matrix C for the dynamics calculations.

Eqs:

$\ddot{x}_g = \ddot{x} + \alpha \times \bar{r} + \omega(\omega \times \bar{r})$

$\bar{r} =^w R_p \cdot {}^p\bar{r}$

$\bar{I}_p =^w R_p^p \bar{I}_p^w R_p^T$

$$C = \left(\begin{array}{c} m_p G - m_p\ddot{x}_g - \sum F_i^n \\ m_p\bar{r} \times G - m_p(\bar{r} \times \ddot{x}_g - \bar{I}_p\alpha + \bar{I}_p\omega \times \omega - \sum a_{wi} \times F_i^n - \sum M_i \end{array} \right)$$

Where:

- $\ddot{x}_g$ is the acceleration of the platform center of mass.
- $\ddot{x}$ is the acceleration of the platform center (first three elements of the ddq vector).
- α is the angular acceleration of the platform.
- $\bar{r}$ is the platform center of mass in the world referential.
- ω is the angular speed of the platform.
- ${}^wR_p$ is the rotational matrix of the two referentials (world and platform).
- ${}^p\bar{r}$ is the vector of the center of mass of the platform with reference to the local frame (platform).
- ${}^p\bar{I}_p$ is the constant mass moments of inertia of the platform with reference to the local frame (platform).

- m_p is the mass of the platform.

- G is the gravity.

- F_i^n is the normal force transferred from the platform to the link.

- $\bar{I}_p$ is the constant mass moments of inertia of the platform in the world referential.

- a_{wi} is the position of the attachment point of each link to the platform in the world referential.

- M_i is the moment transferred from the platform to the link (not present is the spherical joint is at the platform end).

Definition at line 1512 of file stewart.cpp.

References Alpha, ddq, gravity, Links, LinkStewart::Moment(), mp, Omega, pIp, pR, UJointAtBase, and wRp.

Referenced by JointSpaceForceVct().

4.26.3.13 ReturnMatrix Stewart::Torque (const Real *Gravity* = `GRAVITY`)

Return the torque vector of the platform.

Parameters:

Gravity,: Gravity (9.81)

Eq:

$\tau = J^{-T}F$

Where:

- J is the Jacobian matrix of the platform.

- F is the joint space force vector (JointSpaceForceVct()).

Definition at line 1625 of file stewart.cpp.

References ddl, dl, Jacobian, JointSpaceForceVct(), and Links.

Referenced by Find_h(), Find_M(), and stewartmain().

4.26.3.14 ReturnMatrix Stewart::JointSpaceForceVct (const Real *Gravity* = `GRAVITY`)

Return a vector containing the six actuation force components.

Parameters:

Gravity,: Gravity (9.81)

See the description of LinkStewart::ActuationForce().

Definition at line 1595 of file stewart.cpp.

References Find_C(), Find_InvJacob1(), IJ1, and Links.

Referenced by Torque().

4.26.3.15 ReturnMatrix Stewart::InvPosKine ()

Return the lenght of the links in a vector.

The goal of the inverse kinematic is to find the lenght of each of the six links from the position of the platform (X,Y,Z,ψ,θ,ϕ).

Definition at line 1429 of file stewart.cpp.

References L, and Links.

Referenced by stewartmain().

4.26.3.16 ReturnMatrix Stewart::ForwardKine (const ColumnVector *guess_q*, const ColumnVector *l_given*, const Real *tolerance* = `0.001`)

Return the position vector of the platform (vector q).

Parameters:

guess_q,: Approximation of real position
l_given,: Lenght of the 6 links
tolerance,: Ending criterion

The Newton-Raphson method is used to solve the forward kinematic problem. It is a numerical iterative technic that simplify the solution. An approximation of the answer has to be guess for this method to work.

Eq:

$q_i = q_{i-1} - J_{q_{i-1}}(l_{q_{i-1}} - l)$

Where:

- q_i is the position vector of the platform at the ith iteration.
- q_{i-1} is the position vector of the platform at the (i-1)th iteration.
- $J_{q_{i-1}}$ is the Jacobian matrix of the platform at the position of the q_{i-1} vector.
- $l_{q_{i-1}}$ is the lenght vector of the links at the (i-1)th position of the platform.
- l is the real lenght vector of the links.

Definition at line 1567 of file stewart.cpp.

References Jacobian, L, Links, q, and set_q().

Referenced by stewartmain().

4.26.3.17 ReturnMatrix Stewart::Find_h (const Real *Gravity* = `GRAVITY`)

Return the intermediate matrix corresponding to the Coriolis and centrifugal + gravity force/torque components.

Parameters:

Gravity,: Gravity (9.81)

h is found by setting the ddq vector to zero and then calling the torque routine. The vector returned by Torque() is equal to h.

Definition at line 1646 of file stewart.cpp.

References set_ddq(), and Torque().

Referenced by Find_Mc_Nc_Gc(), and ForwardDyn().

4.26.3.18 ReturnMatrix Stewart::Find_M ()

Return the intermediate matrix corresponding to the inertia matrix of the machine.

M is found by setting the dq and Gravity vectors to zero and the ddq vector to zero except for the ith element that is set to one. Then, the ith row of M is equal to the matrix returned by Torque().

Definition at line 1663 of file stewart.cpp.

References dq, set_ddq(), set_dq(), and Torque().

Referenced by Find_Mc_Nc_Gc(), and ForwardDyn().

4.26.3.19 ReturnMatrix Stewart::ForwardDyn (const ColumnVector *T*, const Real *Gravity* = `GRAVITY`)

Return the acceleration vector of the platform (ddq).

Parameters:

T,: torque vector

Gravity,: Gravity (9.81)

Eq:

$ddq = M^{-1}(\tau - h)$

Where:

- M is from Find_M() routine.
- τ is the torque vector.
- h is from Find_h() routine.

Definition at line 1701 of file stewart.cpp.

References Find_h(), and Find_M().

Referenced by stewartmain().

4.26.3.20 void Stewart::Find_Mc_Nc_Gc (Matrix & *Mc*, Matrix & *Nc*, Matrix & *Gc*)

Return(!) the intermediates matrix for forward dynamics with actuator dynamics.

Parameters:

Mc,: Inertia matrix of the machine

Nc,: Coriolis and centrifugal force/torque component

Gc,: Gravity force/torque component

Eq:

$K_a = \frac{p}{2\pi n} I_{6\times6}$ $M_a = \frac{2\pi}{np}(J_s + n^2 J_m) I_{6\times6}$ $V_a = \frac{2\pi}{np}(b_s + n^2 b_m) I_{6\times6}$ $M_c = K_a J^T M + M_a J^{-1}$ $N_c = K_a J^T N + (V_a J^{-1} + M_a \frac{dJ^{-1}}{dt}) dq$ $G_c = K_a J^T G$ Where:

- p is the pitch of the ballscrew.
- n is the gear ratio.

- $I_{6\times6}$ is the Identity matrix.
- J_s is the mass moment of inertia of the ballscrew.
- J_m is the mass moment of inertia of the motor.
- b_s is the viscous damping coefficient of the ballscrew.
- b_m is the viscous damping coefficient of the motor.
- J is the Jacobian matrix of the platform.

Definition at line 1736 of file stewart.cpp.

References bm, bs, dq, Find_h(), Find_M(), Jacobian, jacobian_dot(), Jm, Js, M_PI, n, p, and set_dq().

Referenced by ForwardDyn_AD().

4.26.3.21 ReturnMatrix Stewart::ForwardDyn_AD (const ColumnVector *Command*, const Real *t*)

Return the acceleration of the platform (Stewart platform mechanism dynamics including actuator dynamics).

Parameters:

Command,: Vector of the 6 motors voltages.

t,: period of time use to find the currents (di/dt)

Voltages with back emf:

$V' = V - J^{-1}\dot{q}(\frac{2\pi}{p})K_b$

Currents:

$I = \frac{I_{6\times6}}{L}e^{(-R\cdot t/L)}V'$

Motor torque:

$\tau_m = IK_t$

Platform acceleration:

$\ddot{q} = M_c^{-1}(\tau_m - Nc - Gc)$

Where:

- J is the Jacobian matrix of the platform.
- $\dot{q}$ is the dq vector.

- p is the pitch of the ballscrew.
- K_b is the motor back emf constant.
- L is the motor armature inductance.
- R is the motor armature resistance.
- K_t is the motor torque constant.
- M_c, N_c and G_c are from Find_Mc_Nc_Gc().

Definition at line 1792 of file stewart.cpp.

References dq, Find_Mc_Nc_Gc(), Jacobian, Kb, Kt, L, M_PI, p, and R.

Referenced by stewartmain().

4.27 Trajectory_Select Class Reference

`#include <trajectory.h>`

4.27.1 Detailed Description

Trajectory class selection.

Definition at line 164 of file trajectory.h.

Public Member Functions

- Trajectory_Select ()

  Constructor.

- **Trajectory_Select** (const std::string &filename)
- Trajectory_Select & operator= (const Trajectory_Select &x)

  Overload = operator.

- void **set_trajectory** (const std::string &filename)

Public Attributes

- short type

  Cartesian or joint space.

- Spl_path path

  Spl_path instance.

- Spl_Quaternion path_quat

  Spl_Quaternion instance.

Private Attributes

- bool quaternion_active

  Using Spl_Quaternion.

Chapter 5

ROBOOP, A Robotics Object Oriented Package in C++ File Documentation

5.1 bench.cpp File Reference

5.1.1 Detailed Description

A benchmark file.

Prints the time, on the console, to perform certain operations.

Definition in file bench.cpp.

```
#include <time.h>
#include "robot.h"
#include "stewart.h"
```

Defines

- #define NTRY 2000

Functions

- int stewartmain (void)
- int main (void)

Variables

- static const char rcsid [ ] = "$Id: bench.cpp,v 1.20 2005/07/01 16:16:35 gourdeau Exp $"

  RCS/CVS version.

- Real Stewart_Ini [ ]
- Real Stewart_q [ ]
- Real Stewart_qg [ ]
- Real Stewart_l [ ]
- Real Stewart_dq [ ]
- Real Stewart_ddq [ ]
- Real Stewart_tddq [ ]
- Real Comm [ ]
- Real Tau [ ]
- const Real PUMA560_data [ ]

5.1.2 Variable Documentation

5.1.2.1 Real Comm[ ]

Initial value:

```
{1, 0, 0, -10, 0, 0}
```

Definition at line 96 of file bench.cpp.

Referenced by stewartmain().

5.1.2.2 const Real PUMA560_data[ ]

Initial value:

```
{0, 0, 0, 0, M_PI/2.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.35, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0.4318, 0, 0, 0, 0, 17.4, -0.3638, 0.006, 0.2275, 0.13, 0, 0, 0.524, 0, 0.539, 0,
 0, 0, 0.15005, 0.0203, -M_PI/2.0, 0, 0, 0, 4.8, -0.0203, -0.0141, 0.07, 0.066, 0, 0, 0.086,
 0, 0, 0.4318, 0.0, M_PI/2.0, 0, 0, 0, 0.82, 0, 0.019, 0, 0.0018, 0, 0, 0.0013, 0, 0.0018, 0,
 0, 0, 0, 0.0, -M_PI/2.0, 0, 0, 0, 0.34, 0.0, 0.0, 0.0, 0.0003, 0.0, 0.0, 0.0004, 0.0, 0.000
 0, 0, 0, 0, 0, 0, 0, 0, 0.09, 0.0, 0.0, 0.032, 0.00015, 0.0, 0.0, 0.00015, 0.0, 0.00004, 0,
```

Definition at line 175 of file bench.cpp.

Referenced by dynamics_demo(), and main().

5.1.2.3 Real Stewart_ddq[ ]

Initial value:

```
{-10.0, -10.0, -10, -10.0, -10, -10}
```

Definition at line 92 of file bench.cpp.

Referenced by stewartmain().

5.1.2.4 Real Stewart_dq[ ]

Initial value:

```
{0.2, 0.3, -0.4, 0.1, -1.4, 0.1}
```

Definition at line 90 of file bench.cpp.

Referenced by stewartmain().

5.1.2.5 Real Stewart_Ini[ ]

Initial value:

```
{1.758, 2.8, -1.015, 0.225, 0.0, -0.228, 3.358, 0.05, 4.237, 0.1406, 10, 12.5, 0.5, 0.35, 0.0, 0.0, 0.
 1.6021, 3.07, -0.925, 0.1125, 0.1949, -0.228, 3.358, 0.05, 4.237, 0.1406, 10, 12.5, 0.5, 0.35, 0.0, 0.
 -1.7580, 2.8, -1.015, -0.1125, 0.1949, -0.228, 3.358, 0.05, 4.237, 0.1406, 10, 12.5, 0.5, 0.35, 0.0,
 -1.6021, 3.07, -0.925, -0.225, 0.0, -0.228, 3.358, 0.05, 4.237, 0.1406, 10, 12.5, 0.5, 0.35, 0.0, 0.0,
 0.0, 2.8, 2.03, -0.1125, -0.1949, -0.228, 3.358, 0.05, 4.237, 0.1406, 10, 12.5, 0.5, 0.35, 0.0, 0.0,
 0.0, 3.07, 1.85, 0.1125, -0.1949, -0.228, 3.358, 0.05, 4.237, 0.1406, 10, 12.5, 0.5, 0.35, 0.0, 0.0,
 0.0, 0.0, -0.114, 1.001, 0.59, 0.843, 10, 0.12, 0.04, 0.5, 0.5, 0.5, 1.5, 0.5, 0.005, 5.44, 0.443}
```

Definition at line 75 of file bench.cpp.

Referenced by stewartmain().

5.1.2.6 Real Stewart_l[ ]

Initial value:

```
{3.0508, 3.2324, 3.2997, 3.4560, 3.5797, 3.6935}
```

Definition at line 88 of file bench.cpp.

Referenced by stewartmain().

5.1.2.7 Real Stewart_q[ ]

Initial value:

```
{0.2, 0.3, -0.4, 0.1, -1.4, 0.1}
```

Definition at line 84 of file bench.cpp.

Referenced by stewartmain().

5.1.2.8 Real Stewart_qg[ ]

Initial value:

```
{0.25, 0.25, -0.45, 0.07, -1.7, 0.07}
```

Definition at line 86 of file bench.cpp.

Referenced by stewartmain().

5.1.2.9 Real Stewart_tddq[ ]

Initial value:

```
{0, 0, 0, 0, 0, 0}
```

Definition at line 94 of file bench.cpp.

Referenced by stewartmain().

5.1.2.10 Real Tau[ ]

Initial value:

```
{126.219689, 789.968672, 0.708602, 79.122963, 81.806978, -31.61797}
```

Definition at line 98 of file bench.cpp.

Referenced by stewartmain().

5.2 clik.cpp File Reference

5.2.1 Detailed Description

Clik member functions.

Definition in file clik.cpp.

`#include "quaternion.h"`

`#include "clik.h"`

Variables

- static const char rcsid [ ] = "$Id: clik.cpp,v 1.6 2006/05/16 16:11:15 gourdeau Exp $"

  RCS/CVS version.

5.3 clik.h File Reference

5.3.1 Detailed Description

Header file for Clik class definitions.

Definition in file clik.h.

`#include "robot.h"`

Classes

- class Clik

  Handle Closed Loop Inverse Kinematics scheme.

Defines

- #define CLICK_DH 1

  Using Clik under DH notation.

- #define CLICK_mDH 2

  Using Clik under modified DH notation.

- #define CLICK_mDH_min_para 3

  Using Clik under modified DH notation with minimum intertial parameters.

Variables

- static const char header_clik_rcsid [ ] = "$Id: clik.h,v 1.6 2006/05/16 16:11:15 gourdeau Exp $"

  RCS/CVS version.

5.4 comp_dq.cpp File Reference

5.4.1 Detailed Description

Delta torque (linearized dynamics).

Definition in file comp_dq.cpp.

```
#include "robot.h"
```

Variables

- static const char rcsid [ ] = "$Id: comp_dq.cpp,v 1.17 2004/07/06 02:16:36 gourdeau Exp $"

  RCS/CVS version.

5.5 comp_dqp.cpp File Reference

5.5.1 Detailed Description

Delta torque (linearized dynamics).

Definition in file comp_dqp.cpp.

`#include "robot.h"`

Variables

- static const char rcsid [ ] = "$Id: comp_dqp.cpp,v 1.16 2004/07/06 02:16:36 gourdeau Exp $"

  RCS/CVS version.

5.6 config.cpp File Reference

5.6.1 Detailed Description

Configuration class functions.

Definition in file config.cpp.

```
#include "config.h"
```

Namespaces

- namespace **std**

Variables

- static const char rcsid [ ] = "$Id: config.cpp,v 1.20 2006/05/16 19:24:26 gourdeau Exp $"

  RCS/CVS version.

5.7 config.h File Reference

5.7.1 Detailed Description

Header file for Config class definitions.

Definition in file config.h.

```
#include <iostream>
#include <string>
#include <iomanip>
#include <fstream>
#include <boost/lexical_cast.hpp>
#include <sstream>
#include <vector>
```

Classes

- struct Data

  Basic data element used in Config class.

- class Config

  Handle configuration files.

Defines

- #define CAN_NOT_OPEN_FILE -1

  Return when can not open file.

- #define CAN_NOT_CREATE_FILE -2

  Return when can not create a file.

Typedefs

- typedef std::vector< Data > Conf_data

  Configuration data type.

Variables

- static const char header_config_rcsid [ ] = "$Id: config.h,v 1.18 2006/05/16 19:24:26 gourdeau Exp $"

  RCS/CVS version.

5.8 control_select.cpp File Reference

5.8.1 Detailed Description

Controller selection class.

Definition in file control_select.cpp.

`#include "config.h"`

`#include "control_select.h"`

`#include "trajectory.h"`

Variables

- static const char rcsid [ ] = "$Id: control_select.cpp,v 1.7 2006/05/16 19:24:26 gourdeau Exp $"

  RCS/CVS version.

5.9 control_select.h File Reference

5.9.1 Detailed Description

Header file for Control_Select class definitions.

Definition in file control_select.h.

`#include <string>`

`#include "controller.h"`

Classes

- class Control_Select

  Select controller class.

Defines

- #define NONE 0
- #define PD 1
- #define CTM 2
- #define RRA 3
- #define IMP 4
- #define CONTROLLER "CONTROLLER"
- #define PROPORTIONAL_DERIVATIVE "PROPORTIONAL_DERIVATIVE"
- #define COMPUTED_TORQUE_METHOD "COMPUTED_TORQUE_METHOD"
- #define RESOLVED_RATE_ACCELERATION "RESOLVED_RATE_ACCELERATION"
- #define IMPEDANCE "IMPEDANCE"

Variables

- static const char header_Control_Select_rcsid [ ] = "$Id: control_select.h,v 1.4 2006/05/16 16:11:15 gourdeau Exp $"

  RCS/CVS version.

5.10 controller.cpp File Reference

5.10.1 Detailed Description

Differents controllers class.

Definition in file controller.cpp.

```
#include "controller.h"
```

Variables

- static const char rcsid [ ] = "$Id: controller.cpp,v 1.3 2005/11/15 19:06:13 gourdeau Exp $"

  RCS/CVS version.

5.11 controller.h File Reference

5.11.1 Detailed Description

Header file for controller class definitions.

Definition in file controller.h.

```
#include "robot.h"
#include "quaternion.h"
```

Classes

- class Impedance

  Impedance controller class.

- class Resolved_acc

  Resolved rate acceleration controller class.

- class Computed_torque_method

  Computer torque method controller class.

- class Proportional_Derivative

  Proportional derivative controller class.

Defines

- #define WRONG_SIZE -1

  Return value when input vectors or matrix don't have the right size.

Variables

- static const char header_controller_rcsid [ ] = "$Id: controller.h,v 1.5 2006/05/16 16:11:15 gourdeau Exp $"

  RCS/CVS version.

5.12 delta_t.cpp File Reference

5.12.1 Detailed Description

Delta torque (linearized dynamics).

Definition in file delta_t.cpp.

`#include "robot.h"`

Variables

- static const char rcsid [ ] = "$Id: delta_t.cpp,v 1.17 2005/07/01 16:11:45 gourdeau Exp $"

  RCS/CVS version.

5.13 demo.cpp File Reference

5.13.1 Detailed Description

A demo file.

Demos for homogeneous transforms, kinematics, etc.

Definition in file demo.cpp.

`#include "gnugraph.h"`

`#include "quaternion.h"`

`#include "robot.h"`

`#include "utils.h"`

Functions

- void homogen_demo (void)
- void kinematics_demo (void)
- void dynamics_demo (void)
- int main (void)
- ReturnMatrix xdot (Real t, const Matrix &x)

Variables

- static const char rcsid [ ] = "$Id: demo.cpp,v 1.34 2006/05/16 16:27:43 gourdeau Exp $"

  RCS/CVS version.

- const Real RR_data [ ]
- const Real RR_data_mdh [ ]
- const Real RR_data_mdh_min_para [ ]
- const Real RP_data [ ]
- const Real PUMA560_data [ ]
- const Real PUMA560_motor [ ]
- const Real STANFORD_data [ ]
- Robot robot
- Matrix K
- ColumnVector q0

5.13.2 Variable Documentation

5.13.2.1 const Real PUMA560_data[ ]

Initial value:

```
{0, 0, 0, 0, M_PI/2.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.35, 0, 0, 0,
 0, 0, 0, 0.4318, 0, 0, 0, 0, 17.4, -0.3638, 0.006, 0.2275, 0.13, 0, 0, 0.524, 0, 0.539, 0,
 0, 0, 0.15005, 0.0203, -M_PI/2.0, 0, 0, 0, 4.8, -0.0203, -0.0141, 0.07, 0.066, 0, 0, 0.086,
 0, 0, 0.4318, 0.0, M_PI/2.0, 0, 0, 0, 0.82, 0, 0.019, 0, 0.0018, 0, 0, 0.0013, 0, 0.0018, 0,
 0, 0, 0, 0.0, -M_PI/2.0, 0, 0, 0, 0.34, 0.0, 0.0, 0.0, 0.0003, 0.0, 0.0, 0.0004, 0.0, 0.0003
 0, 0, 0, 0, 0, 0, 0, 0, 0.09, 0.0, 0.0, 0.032, 0.00015, 0.0, 0.0, 0.00015, 0.0, 0.00004, 0}
```

Definition at line 204 of file demo.cpp.

5.13.2.2 const Real PUMA560_motor[ ]

Initial value:

```
{200e-6, -62.6111, 1.48e-3, (.395 +.435)/2,
 200e-6, 107.815, .817e-3, (.126 + .071)/2,
 200e-6, -53.7063, 1.38e-3, (.132     + .105)/2,
 33e-6,   76.0364, 71.2e-6, (11.2e-3 + 16.9e-3)/2,
 33e-6,   71.923, 82.6e-6, (9.26e-3 + 14.5e-3)/2,
 33e-6,   76.686, 36.7e-6, (3.96e-3 + 10.5e-3)/2}
```

Definition at line 211 of file demo.cpp.

Referenced by dynamics_demo(), and main().

5.13.2.3 const Real RP_data[ ]

Initial value:

```
{0, 0, 0, 0, -M_PI/2.0, 0, 0, 0, 2.0, 0, 0, 0.0, 1.0, 0, 0, 1.0, 0, 1.0, 0, 0, 0, 0, 0,
 1, 0, 0, 0, 0, 0, 0, 0, 1.0, 0, 0,-1.0, 0.0833333, 0, 0, 0.0833333, 0, 0.0833333, 0, 0, 0,
```

Definition at line 201 of file demo.cpp.

Referenced by dynamics_demo(), and kinematics_demo().

5.13.2.4 const Real RR_data[ ]

Initial value:

```
{0, 0, 0, 1.0, 0, 0, 0, 0, 2.0,-0.5, 0, 0, 0, 0, 0, 0.1666666, 0, 0.1666666, 0, 0, 0, 0, 0,
 0, 0, 0, 1.0, 0, 0, 0, 0, 1.0,-0.5, 0, 0, 0, 0, 0, 0.0833333, 0, 0.0833333, 0, 0, 0, 0, 0}
```

Definition at line 191 of file demo.cpp.

Referenced by dynamics_demo(), and kinematics_demo().

5.13.2.5 const Real RR_data_mdh[ ]

Initial value:

```
{0, 0, 0, 1.0, 0, 0, 0, 0, 2.0, 0.5, 0, 0, 0, 0, 0, 0.1666666, 0, 0.1666666, 0, 0, 0, 0, 0,
 0, 0, 0, 1.0, 0, 0, 0, 0, 1.0, 0.5, 0, 0, 0, 0, 0, 0.0833333, 0, 0.0833333, 0, 0, 0, 0, 0}
```

Definition at line 194 of file demo.cpp.

5.13.2.6 const Real RR_data_mdh_min_para[ ]

Initial value:

```
{0, 0, 0, 1.0, 0, 0, 0, 0, 0, 0.0, 0, 0,     0, 0, 0, 0, 0.0, 1.666666, 0, 0, 0, 0, 0,
 0, 0, 0, 1.0, 0, 0, 0, 0, 0, 0.5, 0, 0, -0.25, 0, 0, 0, 0.0, 0.3333333, 0, 0, 0, 0, 0}
```

Definition at line 197 of file demo.cpp.

5.13.2.7 const Real STANFORD_data[ ]

Initial value:

```
{0.0, 0.0, 0.4120, 0.0, -M_PI/2, 0,0,0,9.29, 0.0, 0.0175, -0.1105, 0.276, 0.0, 0, 0.255, 0.0, 0.071,0,0,
 0.0, 0.0, 0.1540, 0.0, M_PI/2.0, 0,0,0,5.01, 0.0, -1.054, 0.0, 0.108, 0.0, 0.0, 0.018, 0.0, 0.1,0,0,0,
 1.0, -M_PI/2.0, 0.0, 0.0, 0.0, 0,0,0,4.25, 0.0, 0.0, -6.447, 2.51, 0.0, 0.0, 2.51, 0.0, 0.006,0,0,0,0,
 0.0, 0.0, 0.0, 0.0, -M_PI/2.0, 0,0,0,1.08, 0.0, 0.092, -0.054, 0.002, 0.0, 0.0, 0.001, 0.0, 0.001,0,0,
 0.0, 0.0, 0.0, 0.0,  M_PI/2.0, 0,0,0,0.63, 0.0, 0.0, 0.566, 0.003, 0.0, 0.0, 0.003, 0.0, 0.0004,0,0,0,
 0.0, 0.0, 0.2630, 0.0, 0.0, 0,0,0,0.51, 0.0, 0.0, 1.5540, 0.013, 0.0, 0.0, 0.013, 0.0, 0.0003,0,0,0,0,
```

Definition at line 219 of file demo.cpp.

5.14 demo_2dof_pd.cpp File Reference

5.14.1 Detailed Description

A demo file.

This demo file shows a two degree of freedom robots controller by a pd controller. The robot is define by the file "conf/rr_dh.conf", while the controller is defined by the file "conf/pd_2dof.conf". The desired joint trajectory is defined by the file "conf/q_-2dof.dat";

Definition in file demo_2dof_pd.cpp.

`#include "gnugraph.h"`

`#include "controller.h"`

`#include "control_select.h"`

`#include "dynamics_sim.h"`

`#include "robot.h"`

`#include "trajectory.h"`

Classes

- class New_dynamics

  This is an example of customize Dynamics class.

Functions

- int main ()

Variables

- static const char rcsid [ ] = "$Id: demo_2dof_pd.cpp,v 1.2 2006/05/16 16:27:43 gourdeau Exp $"

  RCS/CVS version.

5.15 dynamics.cpp File Reference

5.15.1 Detailed Description

Manipulator dynamics functions.

Definition in file dynamics.cpp.

`#include "robot.h"`

Variables

- static const char rcsid [ ] = "$Id: dynamics.cpp,v 1.34 2006/05/19 18:32:30 gourdeau Exp $"

  RCS/CVS version.

5.16 dynamics_sim.cpp File Reference

5.16.1 Detailed Description

Basic dynamics simulation class.

Definition in file dynamics_sim.cpp.

`#include "dynamics_sim.h"`

`#include "robot.h"`

Variables

- static const char rcsid [ ] = "$Id: dynamics_sim.cpp,v 1.6 2006/05/19 21:05:57 gourdeau Exp $"

    RCS/CVS version.

5.17 dynamics_sim.h File Reference

5.17.1 Detailed Description

Header file for Dynamics definitions.

Definition in file dynamics_sim.h.

`#include "control_select.h"`

`#include "quaternion.h"`

`#include "trajectory.h"`

`#include "utils.h"`

Classes

- class Dynamics

  Dynamics simulation handling class.

Variables

- static const char header_dynamics_sim_rcsid [ ] = "$Id: dynamics_sim.h,v 1.4 2006/05/16 16:11:15 gourdeau Exp $"

  RCS/CVS version.

5.18 gnugraph.cpp File Reference

5.18.1 Detailed Description

Graphics functions.

Definition in file gnugraph.cpp.

`#include "gnugraph.h"`

Functions

- short set_plot2d (const char ∗title_graph, const char ∗x_axis_title, const char ∗y_axis_title, const char ∗label, LineType_en enLineType, const Matrix &xdata, const Matrix &ydata, int start_y, int end_y)
- short set_plot2d (const char ∗title_graph, const char ∗x_axis_title, const char ∗y_axis_title, const vector< char ∗ > label, LineType_en enLineType, const Matrix &xdata, const Matrix &ydata, const vector< int > &data_select)
- short set_plot3d (const Matrix &xyz, const string &title_graph, const string &x_axis_title, const string &y_axis_title, const string &z_axis_title)

Variables

- static const char rcsid [ ] = "$Id: gnugraph.cpp,v 1.44 2006/05/19 17:49:58 gourdeau Exp $"

  RCS/CVS version.

- char ∗ curvetype [ ]

5.18.2 Variable Documentation

5.18.2.1 char∗ curvetype[ ]

Initial value:

```
{"lines",
 "points",
 "linespoints",
 "impulses",
 "dots",
 "steps",
 "boxes"}
```

Definition at line 81 of file gnugraph.cpp.

Referenced by GNUcurve::dump(), and Plot2d::gnuplot().

5.19 gnugraph.h File Reference

5.19.1 Detailed Description

Header file for graphics definitions.

Definition in file gnugraph.h.

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdexcept>
#include <boost/shared_ptr.hpp>
#include "newmatap.h"
#include "newmatio.h"
#include <sys/stat.h>
#include <sstream>
#include <vector>
```

Classes

- class GNUcurve

  Object for one curve.

- class Plot2d

  2d plot object.

- class Plot3d

  3d plot object.

- class IO_matrix_file

  Read and write data at every iterations in a file.

- class Plot_file

  Creates a graphic from a data file.

Defines

- #define GNUPLOT "gnuplot"

- #define WANT_STRING
- #define WANT_STREAM
- #define WANT_FSTREAM
- #define WANT_MATH
- #define OUT_OF_MEMORY -1
- #define X_Y_DATA_NO_MATCH -2
- #define LABELS_NBR_NO_MATCH -3
- #define NCURVESMAX 10
- #define IO_COULD_NOT_OPEN_FILE -1
- #define IO_MISMATCH_SIZE -2
- #define IO_DATA_EMPTY -3
- #define IO_MISMATCH_ELEMENT_NBR -4
- #define PROBLEM_FILE_READING -5

Typedefs

- typedef boost::shared_ptr< GNUcurve > PSHR_Curve
- typedef std::vector< PSHR_Curve > VectorCurves

Enumerations

- enum LineType_en {

  LINES, **DATAPOINTS**, **LINESPOINTS**, **IMPULSES**,

  DOTS, **STEPS**, **BOXES** }

Functions

- short set_plot2d (const char ∗title_graph, const char ∗x_axis_title, const char ∗y_axis_title, const char ∗label, LineType_en enLineType, const Matrix &xdata, const Matrix &ydata, int start_y, int end_y)
- short set_plot2d (const char ∗title_graph, const char ∗x_axis_title, const char ∗y_axis_title, const vector< char ∗ > label, LineType_en enLineType, const Matrix &xdata, const Matrix &ydata, const vector< int > &data_select)
- short **set_plot3d** (const Matrix &xyz, const std::string &title_graph, const std::string &x_axis_title, const std::string &y_axis_title, const std::string &z_axis_title)

Variables

- static const char header_gnugraph_rcsid [ ] = "$Id: gnugraph.h,v 1.13 2006/05/16 19:24:26 gourdeau Exp $"

  RCS/CVS version.

5.20 homogen.cpp File Reference

5.20.1 Detailed Description

Homogen transformation functions.

Definition in file homogen.cpp.

```
#include "utils.h"
```

Functions

- ReturnMatrix trans (const ColumnVector &a)

  Translation.

- ReturnMatrix rotx (const Real alpha)

  Rotation around x axis.

- ReturnMatrix roty (const Real beta)

  Rotation around x axis.

- ReturnMatrix rotz (const Real gamma)

  Rotation around z axis.

- ReturnMatrix rotk (const Real theta, const ColumnVector &k)

  Rotation around arbitrary axis.

- ReturnMatrix rpy (const ColumnVector &a)

  Roll Pitch Yaw rotation.

- ReturnMatrix eulzxz (const ColumnVector &a)

  Euler ZXZ rotation.

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

  Rotation around an arbitrary line.

- ReturnMatrix irotk (const Matrix &R)

  Obtain axis from a rotation matrix.

- ReturnMatrix irpy (const Matrix &R)

  Obtain Roll, Pitch and Yaw from a rotation matrix.

- ReturnMatrix ieulzxz (const Matrix &R)

  Obtain Roll, Pitch and Yaw from a rotation matrix.

Variables

- static const char rcsid [ ] = "$Id: homogen.cpp,v 1.15 2006/11/15 18:35:17 gourdeau Exp $"

  RCS/CVS version.

5.21 invkine.cpp File Reference

5.21.1 Detailed Description

Inverse kinematics solutions.

Definition in file invkine.cpp.

`#include <stdexcept>`

`#include "robot.h"`

Defines

- #define NITMAX 1000

  def maximum number of iterations in inv_kin

- #define ITOL 1e-6

  def tolerance for the end of iterations in inv_kin

Variables

- static const char rcsid [ ] = "$Id: invkine.cpp,v 1.8 2006/05/16 16:11:15 gourdeau Exp $"

  RCS/CVS version.

5.22 kinemat.cpp File Reference

5.22.1 Detailed Description

Kinematics functions.

Definition in file kinemat.cpp.

`#include "robot.h"`

Variables

- static const char rcsid [ ] = "$Id: kinemat.cpp,v 1.31 2004/08/16 00:37:53 gourdeau Exp $"

  RCS/CVS version.

5.23 quaternion.cpp File Reference

5.23.1 Detailed Description

Quaternion functions.

Definition in file quaternion.cpp.

`#include "quaternion.h"`

Functions

- Quaternion operator ∗ (const Real c, const Quaternion &q)

  Overload ∗ operator, multiplication by a scalar.

- Quaternion operator ∗ (const Quaternion &q, const Real c)

  Overload ∗ operator, multiplication by a scalar.

- Quaternion operator/ (const Real c, const Quaternion &q)

  Overload / operator, division by a scalar.

- Quaternion operator/ (const Quaternion &q, const Real c)
- ReturnMatrix Omega (const Quaternion &q, const Quaternion &q_dot)

  Return angular velocity from a quaternion and it's time derivative.

- short Integ_quat (Quaternion &dquat_present, Quaternion &dquat_past, Quaternion &quat, const Real dt)

  Trapezoidal quaternion integration.

- Real Integ_Trap_quat_s (const Quaternion &present, Quaternion &past, const Real dt)

  Trapezoidal quaternion scalar part integration.

- ReturnMatrix Integ_Trap_quat_v (const Quaternion &present, Quaternion &past, const Real dt)

  Trapezoidal quaternion vector part integration.

- Quaternion Slerp (const Quaternion &q0, const Quaternion &q1, const Real t)

  Spherical Linear Interpolation.

- Quaternion Slerp_prime (const Quaternion &q0, const Quaternion &q1, const Real t)

  Spherical Linear Interpolation derivative.

- Quaternion Squad (const Quaternion &p, const Quaternion &a, const Quaternion &b, const Quaternion &q, const Real t)

  Spherical Cubic Interpolation.

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

  Spherical Cubic Interpolation derivative.

Variables

- static const char rcsid [ ] = "$Id: quaternion.cpp,v 1.18 2005/11/15 19:25:58 gourdeau Exp $"

  RCS/CVS version.

5.23.2 Function Documentation

5.23.2.1 ReturnMatrix Omega (const Quaternion & *q*, const Quaternion & *q_dot*)

Return angular velocity from a quaternion and it's time derivative.

See Quaternion::dot for explanation.

Definition at line 560 of file quaternion.cpp.

References BASE_FRAME, Quaternion::E(), and Quaternion::v().

Referenced by Spl_Quaternion::quat_w().

5.23.2.2 Quaternion operator $*$ (const Real *c*, const Quaternion & *q*)

Overload $*$ operator, multiplication by a scalar.

$q = [s, v]$ and let $r \in R$. Then $rq = qr = [r, 0][s, v] = [rs, rv]$

The result is not necessarily a unit quaternion even if q is a unit quaternions.

Definition at line 516 of file quaternion.cpp.

Referenced by operator $*$().

5.23.2.3 Quaternion operator/ (const Real *c*, const Quaternion & *q*)

Overload / operator, division by a scalar.

Same explanation as multiplication by scaler.

Definition at line 542 of file quaternion.cpp.

Referenced by operator/().

5.23.2.4 Quaternion Slerp (const Quaternion & *q0*, const Quaternion & *q1*, const Real *t*)

Spherical Linear Interpolation.

Cite_:Dam

The quaternion $q(t)$ interpolate the quaternions q_0 and q_1 given the parameter t along the quaternion sphere.

$$q(t) = c_0(t)q_0 + c_1(t)q_1$$

where c_0 and c_1 are real functions with $0 \leq t \leq 1$. As t varies between 0 and 1. the values $q(t)$ varies uniformly along the circular arc from q_0 and q_1. The angle between $q(t)$ and q_0 is $\cos(t\theta)$ and the angle between $q(t)$ and q_1 is $\cos((1-t)\theta)$. Taking the dot product of $q(t)$ and q_0 yields

$$\cos(t\theta) = c_0(t) + \cos(\theta)c_1(t)$$

and taking the dot product of $q(t)$ and q_1 yields

$$\cos((1-t)\theta) = \cos(\theta)c_0(t) + c_1(t)$$

These are two equations with c_0 and c_1. The solution is

$$c_0 = \frac{\sin((1-t)\theta)}{\sin(\theta)}$$

$$c_1 = \frac{\sin(t\theta)}{sin(\theta)}$$

The interpolation is then

$$Slerp(q_0, q_1, t) = \frac{q_0 \sin((1-t)\theta) + q_1 \sin(t\theta)}{\sin(\theta)}$$

If q_0 and q_1 are unit quaternions the $q(t)$ is also a unit quaternions. For unit quaternions we have

$$Slerp(q_0, q_1, t) = q_0(q_0^{-1}q_1)^t$$

For t = 0 and t = 1 we have

$$q_0 = Slerp(q_0, q_1, 0)$$

$$q_1 = Slerp(q_0, q_1, 1)$$

It is customary to choose the sign G on q1 so that q0.Gq1 >=0 (the angle between q0 ang Gq1 is acute). This choice avoids extra spinning caused by the interpolated rotations.

Definition at line 631 of file quaternion.cpp.

References q0.

Referenced by Spl_Quaternion::quat(), Spl_Quaternion::quat_w(), Slerp_prime(), Squad(), and Squad_prime().

5.23.2.5 Quaternion Slerp_prime (const Quaternion & *q0*, const Quaternion & *q1*, const Real *t*)

Spherical Linear Interpolation derivative.

Cite_: Dam

The derivative of the function q^t where q is a constant unit quaternion is

$$\frac{d}{dt}q^t = q^t log(q)$$

Using the preceding equation the Slerp derivative is then

$$Slerp'(q_0, q_1, t) = q_0(q_0^{-1}q_1)^t log(q_0^{-1}q_1)$$

It is customary to choose the sign G on q1 so that q0.Gq1 >=0 (the angle between q0 ang Gq1 is acute). This choice avoids extra spinning caused by the interpolated rotations. The result is not necessary a unit quaternion.

Definition at line 692 of file quaternion.cpp.

References q0, and Slerp().

Referenced by Spl_Quaternion::quat_w().

5.23.2.6 Quaternion Squad (const Quaternion & *p*, const Quaternion & *a*, const Quaternion & *b*, const Quaternion & *q*, const Real *t*)

Spherical Cubic Interpolation.

Cite_: Dam

Let four quaternions be q_i (p), s_i (a), s_{i+1} (b) and q_{i+1} (q) be the ordered vertices of a quadrilateral. Obtain c from q_i to q_{i+1} interpolation. Obtain d from s_i to s_{i+1} interpolation. Obtain e, the final result, from c to d interpolation.

$$Squad(q_i, s_i, s_{i+1}, q_{i+1}, t) = Slerp(Slerp(q_i, q_{i+1}, t), Slerp(s_i, s_{i+1}, t), 2t(1-t))$$

The intermediate quaternion s_i and s_{i+1} are given by

$$s_i = q_i exp\Big(- \frac{log(q_i^{-1}q_{i+1}) + log(q_i^{-1}q_{i-1})}{4}\Big)$$

Definition at line 725 of file quaternion.cpp.

References Slerp().

Referenced by Spl_Quaternion::quat(), and Spl_Quaternion::quat_w().

5.23.2.7 Quaternion Squad_prime (const Quaternion & *p*, const Quaternion & *a*, const Quaternion & *b*, const Quaternion & *q*, const Real *t*)

Spherical Cubic Interpolation derivative.

Cite_: www.magic-software.com

The derivative of the function q^t where q is a constant unit quaternion is

$$\frac{d}{dt}q^t = q^t log(q)$$

Recalling that $log(q) = [0, v\theta]$ (see Quaternion::Log()). If the power is a function we have

$$\frac{d}{dt}q^{f(t)} = f'(t)q^{f(t)}log(q)$$

If q is a function of time and the power is differentiable function of time we have

$$\frac{d}{dt}(q(t))^{f(t)} = f'(t)(q(t))^{f(t)}log(q) + f(t)(q(t))^{f(t)-1}q'(t)$$

Using these last three equations Squad derivative can be define. Let $U(t) = Slerp(p, q, t)$, $V(t) = Slerp(q, b, t)$, $W(t) = U(t)^{-1}V(t)$. We then have $Squad(p, a, b, q, t) = Slerp(U(t), V(t), 2t(1-t)) = U(t)W(t)^{2t(1-t)}$

$$Squad'(p, a, b, q, t) = \frac{d}{dt}\left[UW^{2t(1-t)}\right]$$

$$Squad'(p, a, b, q, t) = U\frac{d}{dt}\left[W^{2t(1-t)}\right] + U'\left[W^{2t(1-t)}\right]$$

$$Squad'(p, a, b, q, t) = U\left[(2-4t)W^{2t(1-t)}log(W) + 2t(1-t)W^{2t(1-t)-1}W'\right] + U'\left[W^{2t(1-t)}\right]$$

where $U' = Ulog(p^{-1}q)$, $V' = Vlog(a^{-1}, b)$, $W' = U^{-1}V' - U^{-2}U'V$

The result is not necessarily a unit quaternion even if all the input quaternions are unit.

Definition at line 751 of file quaternion.cpp.

References Quaternion::i(), Quaternion::power(), and Slerp().

Referenced by Spl_Quaternion::quat_w().

5.24 quaternion.h File Reference

5.24.1 Detailed Description

Quaternion class.

Definition in file quaternion.h.

`#include "robot.h"`

Classes

- class Quaternion

  Quaternion class definition.

Defines

- #define BASE_FRAME 0
- #define BODY_FRAME 1
- #define EPSILON 0.0000001

Functions

- Quaternion operator $*$ (const Real c, const Quaternion &rhs)

  Overload $*$ operator, multiplication by a scalar.

- Quaternion operator $*$ (const Quaternion &lhs, const Real c)

  Overload $*$ operator, multiplication by a scalar.

- Quaternion operator/ (const Real c, const Quaternion &rhs)

  Overload / operator, division by a scalar.

- Quaternion operator/ (const Quaternion &lhs, const Real c)
- ReturnMatrix Omega (const Quaternion &q, const Quaternion &q_dot)

  Return angular velocity from a quaternion and it's time derivative.

- short Integ_quat (Quaternion &dquat_present, Quaternion &dquat_past, Quaternion &quat, const Real dt)

  Trapezoidal quaternion integration.

- Real Integ_Trap_quat_s (const Quaternion &present, Quaternion &past, const Real dt)

Trapezoidal quaternion scalar part integration.

- ReturnMatrix Integ_Trap_quat_v (const Quaternion &present, Quaternion &past, const Real dt)

  Trapezoidal quaternion vector part integration.

- Quaternion Slerp (const Quaternion &q0, const Quaternion &q1, const Real t)

  Spherical Linear Interpolation.

- Quaternion Slerp_prime (const Quaternion &q0, const Quaternion &q1, const Real t)

  Spherical Linear Interpolation derivative.

- Quaternion Squad (const Quaternion &p, const Quaternion &a, const Quaternion &b, const Quaternion &q, const Real t)

  Spherical Cubic Interpolation.

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

  Spherical Cubic Interpolation derivative.

Variables

- static const char header_quat_rcsid [ ] = "$Id: quaternion.h,v 1.12 2005/11/15 19:25:58 gourdeau Exp $"

  RCS/CVS version.

5.24.2 Function Documentation

5.24.2.1 ReturnMatrix Omega (const Quaternion & *q*, const Quaternion & *q_dot*)

Return angular velocity from a quaternion and it's time derivative.

See Quaternion::dot for explanation.

Definition at line 560 of file quaternion.cpp.

References BASE_FRAME, Quaternion::E(), and Quaternion::v().

Referenced by Spl_Quaternion::quat_w().

5.24.2.2 Quaternion operator $*$ (const Real *c*, const Quaternion & *q*)

Overload $*$ operator, multiplication by a scalar.

$q = [s, v]$ and let $r \in R$. Then $rq = qr = [r, 0][s, v] = [rs, rv]$

The result is not necessarily a unit quaternion even if q is a unit quaternions.

Definition at line 516 of file quaternion.cpp.

References Quaternion::s(), Quaternion::set_s(), Quaternion::set_v(), and Quaternion::v().

5.24.2.3 Quaternion operator/ (const Real *c*, const Quaternion & *q*)

Overload / operator, division by a scalar.

Same explanation as multiplication by scaler.

Definition at line 542 of file quaternion.cpp.

References Quaternion::s(), Quaternion::set_s(), Quaternion::set_v(), and Quaternion::v().

5.24.2.4 Quaternion Slerp (const Quaternion & *q0*, const Quaternion & *q1*, const Real *t*)

Spherical Linear Interpolation.

Cite_:Dam

The quaternion $q(t)$ interpolate the quaternions q_0 and q_1 given the parameter t along the quaternion sphere.

$$q(t) = c_0(t)q_0 + c_1(t)q_1$$

where c_0 and c_1 are real functions with $0 \leq t \leq 1$. As t varies between 0 and 1. the values $q(t)$ varies uniformly along the circular arc from q_0 and q_1. The angle between $q(t)$ and q_0 is $\cos(t\theta)$ and the angle between $q(t)$ and q_1 is $\cos((1-t)\theta)$. Taking the dot product of $q(t)$ and q_0 yields

$$\cos(t\theta) = c_0(t) + \cos(\theta)c_1(t)$$

and taking the dot product of $q(t)$ and q_1 yields

$$\cos((1-t)\theta) = \cos(\theta)c_0(t) + c_1(t)$$

These are two equations with c_0 and c_1. The solution is

$$c_0 = \frac{\sin((1-t)\theta)}{\sin(\theta)}$$

$$c_1 = \frac{\sin(t\theta)}{sin(\theta)}$$

The interpolation is then

$$Slerp(q_0, q_1, t) = \frac{q_0 \sin((1-t)\theta) + q_1 \sin(t\theta)}{\sin(\theta)}$$

If q_0 and q_1 are unit quaternions the $q(t)$ is also a unit quaternions. For unit quaternions we have

$$Slerp(q_0, q_1, t) = q_0(q_0^{-1}q_1)^t$$

For t = 0 and t = 1 we have

$$q_0 = Slerp(q_0, q_1, 0)$$

$$q_1 = Slerp(q_0, q_1, 1)$$

It is customary to choose the sign G on q1 so that q0.Gq1 >=0 (the angle between q0 ang Gq1 is acute). This choice avoids extra spinning caused by the interpolated rotations.

Definition at line 631 of file quaternion.cpp.

References q0.

Referenced by Spl_Quaternion::quat(), Spl_Quaternion::quat_w(), Slerp_prime(), Squad(), and Squad_prime().

5.24.2.5 Quaternion Slerp_prime (const Quaternion & *q0*, const Quaternion & *q1*, const Real *t*)

Spherical Linear Interpolation derivative.

Cite_: Dam

The derivative of the function q^t where q is a constant unit quaternion is

$$\frac{d}{dt}q^t = q^t log(q)$$

Using the preceding equation the Slerp derivative is then

$$Slerp'(q_0, q_1, t) = q_0(q_0^{-1}q_1)^t log(q_0^{-1}q_1)$$

It is customary to choose the sign G on q1 so that q0.Gq1 >=0 (the angle between q0 ang Gq1 is acute). This choice avoids extra spinning caused by the interpolated rotations. The result is not necessary a unit quaternion.

Definition at line 692 of file quaternion.cpp.

References q0, and Slerp().

Referenced by Spl_Quaternion::quat_w().

5.24.2.6 Quaternion Squad (const Quaternion & *p*, const Quaternion & *a*, const Quaternion & *b*, const Quaternion & *q*, const Real *t*)

Spherical Cubic Interpolation.

Cite_: Dam

Let four quaternions be q_i (p), s_i (a), s_{i+1} (b) and q_{i+1} (q) be the ordered vertices of a quadrilateral. Obtain c from q_i to q_{i+1} interpolation. Obtain d from s_i to s_{i+1} interpolation. Obtain e, the final result, from c to d interpolation.

$$Squad(q_i, s_i, s_{i+1}, q_{i+1}, t) = Slerp(Slerp(q_i, q_{i+1}, t), Slerp(s_i, s_{i+1}, t), 2t(1-t))$$

The intermediate quaternion s_i and s_{i+1} are given by

$$s_i = q_i exp\Big(- \frac{log(q_i^{-1} q_{i+1}) + log(q_i^{-1} q_{i-1})}{4} \Big)$$

Definition at line 725 of file quaternion.cpp.

References Slerp().

Referenced by Spl_Quaternion::quat(), and Spl_Quaternion::quat_w().

5.24.2.7 Quaternion Squad_prime (const Quaternion & *p*, const Quaternion & *a*, const Quaternion & *b*, const Quaternion & *q*, const Real *t*)

Spherical Cubic Interpolation derivative.

Cite_: www.magic-software.com

The derivative of the function q^t where q is a constant unit quaternion is

$$\frac{d}{dt} q^t = q^t log(q)$$

Recalling that $log(q) = [0, v\theta]$ (see Quaternion::Log()). If the power is a function we have

$$\frac{d}{dt} q^{f(t)} = f'(t) q^{f(t)} log(q)$$

If q is a function of time and the power is differentiable function of time we have

$$\frac{d}{dt} (q(t))^{f(t)} = f'(t)(q(t))^{f(t)} log(q) + f(t)(q(t))^{f(t)-1} q'(t)$$

Using these last three equations Squad derivative can be define. Let $U(t) = Slerp(p, q, t)$, $V(t) = Slerp(q, b, t)$, $W(t) = U(t)^{-1} V(t)$. We then have $Squad(p, a, b, q, t) = Slerp(U(t), V(t), 2t(1-t)) = U(t) W(t)^{2t(1-t)}$

$$Squad'(p, a, b, q, t) = \frac{d}{dt} \Big[U W^{2t(1-t)} \Big]$$

$$Squad'(p,a,b,q,t) = U\frac{d}{dt}\left[W^{2t(1-t)}\right] + U'\left[W^{2t(1-t)}\right]$$

$$Squad'(p,a,b,q,t) = U\left[(2-4t)W^{2t(1-t)}log(W)+2t(1-t)W^{2t(1-t)-1}W'\right]+U'\left[W^{2t(1-t)}\right]$$

where $U' = Ulog(p^{-1}q)$, $V' = Vlog(a^{-1},b)$, $W' = U^{-1}V' - U^{-2}U'V$

The result is not necessarily a unit quaternion even if all the input quaternions are unit.

Definition at line 751 of file quaternion.cpp.

References Quaternion::i(), Quaternion::power(), and Slerp().

Referenced by Spl_Quaternion::quat_w().

5.25 robot.cpp File Reference

5.25.1 Detailed Description

Initialisation of differents robot class.

Definition in file robot.cpp.

`#include <time.h>`

`#include "config.h"`

`#include "robot.h"`

Functions

- void perturb_robot (Robot_basic &robot, const double f)

  Modify a robot.
- bool Rhino_DH (const Robot_basic &robot)

  Return true if the robot is like a Rhino on DH notation.
- bool Puma_DH (const Robot_basic &robot)

  Return true if the robot is like a Puma on DH notation.
- bool Schilling_DH (const Robot_basic &robot)

  Return true if the robot is like a Schilling on DH notation.
- bool Rhino_mDH (const Robot_basic &robot)

  Return true if the robot is like a Rhino on modified DH notation.
- bool Puma_mDH (const Robot_basic &robot)

  Return true if the robot is like a Puma on modified DH notation.
- bool Schilling_mDH (const Robot_basic &robot)

  Return true if the robot is like a Schilling on modified DH notation.

Variables

- static const char rcsid [ ] = "$Id: robot.cpp,v 1.50 2006/05/16 19:24:26 gourdeau Exp $"

  RCS/CVS version.

- Real fourbyfourident [ ] = {1.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,1.0}

  Used to initialize a 4×4 *matrix.*

- Real threebythreeident [ ] = {1.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,1.0}

  Used to initialize a 3×3 *matrix.*

5.25.2 Function Documentation

5.25.2.1 void perturb_robot (Robot_basic & *robot*, const double *f*)

Modify a robot.

Parameters:

robot,: Robot_basic reference.

f,: Percentage of erreur between 0 and 1.

f represents an error to added on the robot inertial parameter. f is between 0 (no error) and 1 (100% error).

Definition at line 1446 of file robot.cpp.

References Link::get_B(), Link::get_Cf(), Robot_basic::get_dof(), Robot_basic::get_fix(), Link::get_I(), Link::get_Im(), Link::get_m(), Robot_basic::links, robot, Link::set_B(), Link::set_Cf(), Link::set_I(), Link::set_Im(), and Link::set_m().

5.25.2.2 bool Puma_DH (const Robot_basic & *robot*)

Return true if the robot is like a Puma on DH notation.

Compare the robot DH table with the Puma DH table. The function return true if the tables are similar (same alpha and similar a and d parameters).

Definition at line 1516 of file robot.cpp.

References Link::get_a(), Link::get_alpha(), Link::get_d(), Robot_basic::get_dof(), Link::get_joint_type(), isZero(), Robot_basic::links, and robot.

Referenced by Robot::robotType_inv_kin().

5.25.2.3 bool Puma_mDH (const Robot_basic & *robot*)

Return true if the robot is like a Puma on modified DH notation.

Compare the robot DH table with the Puma DH table. The function return true if the tables are similar (same alpha and similar a and d parameters).

Definition at line 1615 of file robot.cpp.

References Link::get_a(), Link::get_alpha(), Link::get_d(), Robot_basic::get_dof(), Link::get_joint_type(), isZero(), Robot_basic::links, and robot.

Referenced by mRobot_min_para::robotType_inv_kin(), and mRobot::robotType_inv_kin().

5.25.2.4 bool Rhino_DH (const Robot_basic & *robot*)

Return true if the robot is like a Rhino on DH notation.

Compare the robot DH table with the Puma DH table. The function return true if the tables are similar (same alpha and similar a and d parameters).

Definition at line 1483 of file robot.cpp.

References Link::get_a(), Link::get_alpha(), Link::get_d(), Robot_basic::get_dof(), Link::get_joint_type(), isZero(), Robot_basic::links, and robot.

Referenced by Robot::robotType_inv_kin().

5.25.2.5 bool Rhino_mDH (const Robot_basic & *robot*)

Return true if the robot is like a Rhino on modified DH notation.

Compare the robot DH table with the Puma DH table. The function return true if the tables are similar (same alpha and similar a and d parameters).

Definition at line 1583 of file robot.cpp.

References Link::get_a(), Link::get_alpha(), Link::get_d(), Robot_basic::get_dof(), Link::get_joint_type(), isZero(), Robot_basic::links, and robot.

Referenced by mRobot_min_para::robotType_inv_kin(), and mRobot::robotType_inv_kin().

5.25.2.6 bool Schilling_DH (const Robot_basic & *robot*)

Return true if the robot is like a Schilling on DH notation.

Compare the robot DH table with the Schilling DH table. The function return true if the tables are similar (same alpha and similar a and d parameters).

Definition at line 1549 of file robot.cpp.

References Link::get_a(), Link::get_alpha(), Link::get_d(), Robot_basic::get_dof(), Link::get_joint_type(), isZero(), Robot_basic::links, and robot.

Referenced by Robot::robotType_inv_kin().

5.25.2.7 bool Schilling_mDH (const Robot_basic & *robot*)

Return true if the robot is like a Schilling on modified DH notation.

Compare the robot DH table with the Schilling DH table. The function return true if the tables are similar (same alpha and similar a and d parameters).

Definition at line 1649 of file robot.cpp.

References Link::get_a(), Link::get_alpha(), Link::get_d(), Robot_basic::get_dof(), Link::get_joint_type(), isZero(), Robot_basic::links, and robot.

Referenced by mRobot_min_para::robotType_inv_kin(), and mRobot::robotType_inv_kin().

5.26 robot.h File Reference

5.26.1 Detailed Description

Robots class definitions.

Definition in file robot.h.

`#include "utils.h"`

Classes

- class Link

  Link definitions.

- class Robot_basic

  Virtual base robot class.

- class Robot

  DH notation robot class.

- class mRobot

  Modified DH notation robot class.

- class mRobot_min_para

  Modified DH notation and minimal inertial parameters robot class.

Functions

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

  Modify a robot.

- bool Rhino_DH (const Robot_basic &robot)

  Return true if the robot is like a Rhino on DH notation.

- bool Puma_DH (const Robot_basic &robot)

  Return true if the robot is like a Puma on DH notation.

- bool Schilling_DH (const Robot_basic &robot)

  Return true if the robot is like a Schilling on DH notation.

- bool Rhino_mDH (const Robot_basic &robot)

Return true if the robot is like a Rhino on modified DH notation.

- bool Puma_mDH (const Robot_basic &robot)

  Return true if the robot is like a Puma on modified DH notation.

- bool Schilling_mDH (const Robot_basic &robot)

  Return true if the robot is like a Schilling on modified DH notation.

Variables

- static const char header_rcsid [ ] = "$Id: robot.h,v 1.52 2006/05/16 16:11:15 gourdeau Exp $"

  RCS/CVS version.

5.26.2 Function Documentation

5.26.2.1 void perturb_robot (Robot_basic & *robot*, const double *f*)

Modify a robot.

Parameters:

robot,: Robot_basic reference.

f,: Percentage of erreur between 0 and 1.

f represents an error to added on the robot inertial parameter. f is between 0 (no error) and 1 (100% error).

Definition at line 1446 of file robot.cpp.

References Link::get_B(), Link::get_Cf(), Robot_basic::get_dof(), Robot_basic::get_-fix(), Link::get_I(), Link::get_Im(), Link::get_m(), Robot_basic::links, robot, Link::set_B(), Link::set_Cf(), Link::set_I(), Link::set_Im(), and Link::set_m().

5.26.2.2 bool Puma_DH (const Robot_basic & *robot*)

Return true if the robot is like a Puma on DH notation.

Compare the robot DH table with the Puma DH table. The function return true if the tables are similar (same alpha and similar a and d parameters).

Definition at line 1516 of file robot.cpp.

References Link::get_a(), Link::get_alpha(), Link::get_d(), Robot_basic::get_dof(), Link::get_joint_type(), isZero(), Robot_basic::links, and robot.

Referenced by Robot::robotType_inv_kin().

5.26.2.3 bool Puma_mDH (const Robot_basic & *robot*)

Return true if the robot is like a Puma on modified DH notation.

Compare the robot DH table with the Puma DH table. The function return true if the tables are similar (same alpha and similar a and d parameters).

Definition at line 1615 of file robot.cpp.

References Link::get_a(), Link::get_alpha(), Link::get_d(), Robot_basic::get_dof(), Link::get_joint_type(), isZero(), Robot_basic::links, and robot.

Referenced by mRobot::robotType_inv_kin(), and mRobot_min_para::robotType_inv_kin().

5.26.2.4 bool Rhino_DH (const Robot_basic & *robot*)

Return true if the robot is like a Rhino on DH notation.

Compare the robot DH table with the Puma DH table. The function return true if the tables are similar (same alpha and similar a and d parameters).

Definition at line 1483 of file robot.cpp.

References Link::get_a(), Link::get_alpha(), Link::get_d(), Robot_basic::get_dof(), Link::get_joint_type(), isZero(), Robot_basic::links, and robot.

Referenced by Robot::robotType_inv_kin().

5.26.2.5 bool Rhino_mDH (const Robot_basic & *robot*)

Return true if the robot is like a Rhino on modified DH notation.

Compare the robot DH table with the Puma DH table. The function return true if the tables are similar (same alpha and similar a and d parameters).

Definition at line 1583 of file robot.cpp.

References Link::get_a(), Link::get_alpha(), Link::get_d(), Robot_basic::get_dof(), Link::get_joint_type(), isZero(), Robot_basic::links, and robot.

Referenced by mRobot::robotType_inv_kin(), and mRobot_min_para::robotType_inv_kin().

5.26.2.6 bool Schilling_DH (const Robot_basic & *robot*)

Return true if the robot is like a Schilling on DH notation.

Compare the robot DH table with the Schilling DH table. The function return true if the tables are similar (same alpha and similar a and d parameters).

Definition at line 1549 of file robot.cpp.

References Link::get_a(), Link::get_alpha(), Link::get_d(), Robot_basic::get_dof(), Link::get_joint_type(), isZero(), Robot_basic::links, and robot.

Referenced by Robot::robotType_inv_kin().

5.26.2.7 bool Schilling_mDH (const Robot_basic & *robot*)

Return true if the robot is like a Schilling on modified DH notation.

Compare the robot DH table with the Schilling DH table. The function return true if the tables are similar (same alpha and similar a and d parameters).

Definition at line 1649 of file robot.cpp.

References Link::get_a(), Link::get_alpha(), Link::get_d(), Robot_basic::get_dof(), Link::get_joint_type(), isZero(), Robot_basic::links, and robot.

Referenced by mRobot::robotType_inv_kin(), and mRobot_min_para::robotType_inv_kin().

5.27 rtest.cpp File Reference

5.27.1 Detailed Description

A test file.

Compares results with Peter Corke MATLAB toolbox

Definition in file rtest.cpp.

`#include "robot.h"`

`#include "quaternion.h"`

`#include "precisio.h"`

`#include <fstream>`

Functions

- int main (void)

Variables

- static const char rcsid [ ] = "$Id: rtest.cpp,v 1.15 2005/07/01 17:44:53 gourdeau Exp $"

  RCS/CVS version.

- const Real PUMA560_data_DH [ ]
- Real PUMA560_data_mDH [ ]
- const Real PUMA560_motor [ ]
- const Real STANFORD_data_DH [ ]

5.27.2 Variable Documentation

5.27.2.1 const Real PUMA560_data_DH[ ]

Initial value:

```
{0, 0, 0, 0, M_PI/2.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.35, 0, 0, 0,
 0, 0, 0, 0.4318, 0, 0, 0, 0, 17.4, -0.3638, 0.006, 0.2275, 0.13, 0, 0, 0.524, 0, 0.539, 0,
 0, 0, 0.15005, 0.0203, -M_PI/2.0, 0, 0, 0, 4.8, -0.0203, -0.0141, 0.07, 0.066, 0, 0, 0.086, 0, 0.0125,
 0, 0, 0.4318, 0.0, M_PI/2.0, 0, 0, 0, 0.82, 0, 0.019, 0, 0.0018, 0, 0, 0.0013, 0, 0.0018, 0,
 0, 0, 0, 0.0, -M_PI/2.0, 0, 0, 0, 0.34, 0.0, 0.0, 0.0, 0.0003, 0.0, 0.0, 0.0004, 0.0, 0.0003, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0.09, 0.0, 0.0, 0.032, 0.00015, 0.0, 0.0, 0.00015, 0.0, 0.00004, 0}
```

Definition at line 61 of file rtest.cpp.

Referenced by main().

5.27.2.2 Real PUMA560_data_mDH[ ]

Initial value:

```
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.35, 0,
 0, 0, 0, 0.0, -M_PI/2, 0, 0, 0, 17.4, 0.068, 0.006, -0.016, 0.13, 0, 0, 0.524, 0, 0.539, 0,
 0, 0, -0.15005, 0.4318, 0, 0, 0, 0, 4.8, 0, -0.070, 0.014, 0.066, 0, 0, 0.0125, 0, 0.066, 0,
 0, 0, -0.4318, 0.0203, -M_PI/2.0, 0, 0, 0, 0.82, 0.0, 0.0, -0.019, 0.0018, 0, 0, 0.0018, 0, 0
 0, 0, 0, 0, M_PI/2.0, 0, 0, 0, 0.34, 0, 0, 0.0, 0.0003, 0, 0, 0.0003, 0, 0.0004, 0,
 0, 0, 0, 0, -M_PI/2, 0, 0, 0, 0.09, 0, 0, 0.032, 0.00015, 0, 0, 0.00015, 0, 0.00004, 0}
```

Definition at line 68 of file rtest.cpp.

Referenced by main().

5.27.2.3 const Real PUMA560_motor[ ]

Initial value:

```
{200e-6, -62.6111, 1.48e-3, 0,
 200e-6, 107.815, .817e-3, 0,
 200e-6, -53.7063, 1.38e-3, 0,
 33e-6,  76.0364, 71.2e-6, 0,
 33e-6,  71.923,  82.6e-6, 0,
 33e-6,  76.686,  36.7e-6, 0}
```

Definition at line 76 of file rtest.cpp.

5.27.2.4 const Real STANFORD_data_DH[ ]

Initial value:

```
{0.0, 0.0, 0.4120, 0.0, -M_PI/2, 0,0,0,9.29, 0.0, 0.0175, -0.1105, 0.276, 0.0, 0, 0.255, 0.0,
 0.0, 0.0, 0.1540, 0.0, M_PI/2.0, 0,0,0,5.01, 0.0, -1.054, 0.0, 0.108, 0.0, 0.0, 0.018, 0.0,
 1.0, -M_PI/2.0, 0.0, 0.0, 0.0, 0,0,0,4.25, 0.0, 0.0, -6.447, 2.51, 0.0, 0.0, 2.51, 0.0, 0.00
 0.0, 0.0, 0.0, 0.0, -M_PI/2.0, 0,0,0,1.08, 0.0, 0.092, -0.054, 0.002, 0.0, 0.0, 0.001, 0.0,
 0.0, 0.0, 0.0, 0.0,  M_PI/2.0, 0,0,0,0.63, 0.0, 0.0, 0.566, 0.003, 0.0, 0.0, 0.003, 0.0, 0.0
 0.0, 0.0, 0.2630, 0.0, 0.0, 0,0,0,0.51, 0.0, 0.0, 1.5540, 0.013, 0.0, 0.0, 0.013, 0.0, 0.000
```

Definition at line 84 of file rtest.cpp.

Referenced by main().

5.28 sensitiv.cpp File Reference

5.28.1 Detailed Description

Delta torque (linearized dynamics).

Definition in file sensitiv.cpp.

```
#include "robot.h"
```

Variables

- static const char rcsid [ ] = "$Id: sensitiv.cpp,v 1.13 2004/07/06 02:16:37 gourdeau Exp $"

  RCS/CVS version.

5.29 stewart.cpp File Reference

5.29.1 Detailed Description

Initialisation of Stewart platform class.

Definition in file stewart.cpp.

```
#include "config.h"
```

```
#include "stewart.h"
```

Variables

- static const char rcsid [ ] = "$Id: stewart.cpp,v 1.6 2006/05/16 19:24:26 gourdeau Exp $"

  RCS/CVS version.

5.30 stewart.h File Reference

5.30.1 Detailed Description

Stewart class definitions.

Definition in file stewart.h.

`#include "utils.h"`

Classes

- class LinkStewart

  LinkStewart definitions.

- class Stewart

  Stewart definitions.

Variables

- static const char header_stewart_rcsid [ ] = "$Id: stewart.h,v 1.2 2006/05/16 16:11:15 gourdeau Exp $"

  RCS/CVS version.

5.31 trajectory.cpp File Reference

5.31.1 Detailed Description

Trajectory member functions.

Definition in file trajectory.cpp.

```
#include "trajectory.h"
```

Variables

- static const char rcsid [ ] = "$Id: trajectory.cpp,v 1.8 2006/05/16 19:24:26 gourdeau Exp $"

  RCS/CVS version.

5.32 trajectory.h File Reference

5.32.1 Detailed Description

Header file for trajectory generation class.

Definition in file trajectory.h.

`#include <sstream>`

`#include <map>`

`#include "quaternion.h"`

`#include "utils.h"`

Classes

- class Spl_cubic

  Natural cubic splines class.

- class Spl_path

  Cartesian or joint space trajectory.

- class Spl_Quaternion

  Cubic quaternions spline.

- class Trajectory_Select

  Trajectory class selection.

Defines

- #define K_ZER0 1
- #define BAD_DATA -1
- #define EXTRAPOLLATION -2
- #define NOT_IN_RANGE -3
- #define NONE 0
- #define JOINT_SPACE 1
- #define CARTESIAN_SPACE 2

Typedefs

- typedef std::map< Real, ColumnVector, less< Real > > point_map

Data at control points.

- typedef std::map< Real, Quaternion, less< Real > > quat_map

  Data at control points.

Variables

- static const char header_trajectory_rcsid [ ] = "$Id: trajectory.h,v 1.10 2006/05/16 19:24:26 gourdeau Exp $"

  RCS/CVS version.

5.33 utils.cpp File Reference

5.33.1 Detailed Description

Utility functions.

Definition in file utils.cpp.

`#include "utils.h"`

Defines

- #define PGROW -0.20
- #define PSHRNK -0.25
- #define FCOR 0.06666666
- #define SAFETY 0.9
- #define ERRCON 6.0E-4
- #define MAXSTP 10000
- #define TINY 1.0e-30

Functions

- ReturnMatrix x_prod_matrix (const ColumnVector &x)

  Cross product matrix.

- ReturnMatrix pinv (const Matrix &M)

  Matrix pseudo inverse using SVD.

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

  Trapezoidal integration.

- void Runge_Kutta4_Real_time (ReturnMatrix(∗xdot)(Real time, const Matrix &xin, bool &exit, bool &init), const Matrix &xo, Real to, Real tf, int nsteps)

  Fixed step size fourth-order Runge-Kutta integrator.

- void Runge_Kutta4_Real_time (ReturnMatrix(∗xdot)(const Real time, const Matrix &xin), const Matrix &xo, const Real to, const Real tf, const int nsteps)
- void Runge_Kutta4 (ReturnMatrix(∗xdot)(Real time, const Matrix &xin), const Matrix &xo, Real to, Real tf, int nsteps, RowVector &tout, Matrix &xout)

  Fixed step size fourth-order Runge-Kutta integrator.

- ReturnMatrix rk4 (const Matrix &x, const Matrix &dxdt, Real t, Real h, ReturnMatrix(∗xdot)(Real time, const Matrix &xin))

  Compute one Runge-Kutta fourth order step.

- void rkqc (Matrix &x, Matrix &dxdt, Real &t, Real htry, Real eps, Matrix &xscal, Real &hdid, Real &hnext, ReturnMatrix(∗xdot)(Real time, const Matrix &xin))

  Compute one adaptive step based on two rk4.

- 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)

  Integrate the ordinary differential equation xdot from time to to time tf using an adaptive step size strategy.

- ReturnMatrix sign (const Matrix &x)

  Sign of a matrix.

- short sign (const Real x)

  Sign of real.

Variables

- static const char rcsid [ ] = "$Id: utils.cpp,v 1.26 2006/05/16 16:11:15 gourdeau Exp $"

  RCS/CVS version.

5.33.2 Function Documentation

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

Integrate the ordinary differential equation xdot from time to to time tf using an adaptive step size strategy.

adapted from: Numerical Recipes in C, The Art of Scientific Computing, Press, William H. and Flannery, Brian P. and Teukolsky, Saul A. and Vetterling, William T., Cambridge University Press, 1988.

Definition at line 347 of file utils.cpp.

References MAXSTP, rkqc(), and TINY.

Referenced by dynamics_demo().

5.33.2.2 ReturnMatrix pinv (const Matrix & *M*)

Matrix pseudo inverse 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 $1/\sigma_i < tol$

Definition at line 99 of file utils.cpp.

References epsilon, and pinv().

Referenced by pinv().

5.33.2.3 ReturnMatrix rk4 (const Matrix & *x*, const Matrix & *dxdt*, Real *t*, Real *h*, ReturnMatrix($*$)(Real time, const Matrix &xin) *xdot*)

Compute one Runge-Kutta fourth order step.

adapted from: Numerical Recipes in C, The Art of Scientific Computing, Press, William H. and Flannery, Brian P. and Teukolsky, Saul A. and Vetterling, William T., Cambridge University Press, 1988.

Definition at line 260 of file utils.cpp.

Referenced by rkqc().

5.33.2.4 void rkqc (Matrix & *x*, Matrix & *dxdt*, Real & *t*, Real *htry*, Real *eps*, Matrix & *xscal*, Real & *hdid*, Real & *hnext*, ReturnMatrix($*$)(Real time, const Matrix &xin) *xdot*)

Compute one adaptive step based on two rk4.

adapted from: Numerical Recipes in C, The Art of Scientific Computing, Press, William H. and Flannery, Brian P. and Teukolsky, Saul A. and Vetterling, William T., Cambridge University Press, 1988.

Definition at line 295 of file utils.cpp.

References ERRCON, FCOR, PGROW, PSHRNK, rk4(), and SAFETY.

Referenced by odeint().

5.34 utils.h File Reference

5.34.1 Detailed Description

Utility header file.

Definition in file utils.h.

```
#include <stdio.h>
#include <limits>
#include "newmatap.h"
#include "newmatio.h"
```

Defines

- #define WANT_STRING
- #define WANT_STREAM
- #define WANT_FSTREAM
- #define WANT_MATH
- #define M_PI 3.14159265358979
- #define GRAVITY 9.81

Functions

- double deg2rad (const double angle_deg)
- double rad2deg (const double angle_rad)
- ReturnMatrix x_prod_matrix (const ColumnVector &x)

  Cross product matrix.

- ReturnMatrix pinv (const Matrix &M)

  Matrix pseudo inverse using SVD.

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

  Trapezoidal integration.

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

  Fixed step size fourth-order Runge-Kutta integrator.

- void **Runge_Kutta4_Real_time** (ReturnMatrix(∗xdot)(Real time, const Matrix &xin), const Matrix &xo, Real to, Real tf, int nsteps)

- void Runge_Kutta4_Real_time (ReturnMatrix(∗xdot)(Real time, const Matrix &xin, bool &exit, bool &init), const Matrix &xo, Real to, Real tf, int nsteps)

  Fixed step size fourth-order Runge-Kutta integrator.

- 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)

  Integrate the ordinary differential equation xdot from time to to time tf using an adaptive step size strategy.

- ReturnMatrix sign (const Matrix &x)

  Sign of a matrix.

- short sign (const Real x)

  Sign of real.

- bool isZero (const double x)
- ReturnMatrix trans (const ColumnVector &a)

  Translation.

- ReturnMatrix rotx (const Real alpha)

  Rotation around x axis.

- ReturnMatrix roty (const Real beta)

  Rotation around x axis.

- ReturnMatrix rotz (const Real gamma)

  Rotation around z axis.

- ReturnMatrix rotk (const Real theta, const ColumnVector &k)

  Rotation around arbitrary axis.

- ReturnMatrix rpy (const ColumnVector &a)

  Roll Pitch Yaw rotation.

- ReturnMatrix eulzxz (const ColumnVector &a)

  Euler ZXZ rotation.

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

  Rotation around an arbitrary line.

- ReturnMatrix irotk (const Matrix &R)

  Obtain axis from a rotation matrix.

- ReturnMatrix irpy (const Matrix &R)

  Obtain Roll, Pitch and Yaw from a rotation matrix.

- ReturnMatrix ieulzxz (const Matrix &R)

  Obtain Roll, Pitch and Yaw from a rotation matrix.

Variables

- static const char header_utils_rcsid [ ] = "$Id: utils.h,v 1.10 2006/05/16 16:11:15 gourdeau Exp $"

  RCS/CVS version.

- Real fourbyfourident [ ]

  Used to initialize a 4×4 *matrix.*

- Real threebythreeident [ ]

  Used to initialize a 3×3 *matrix.*

- const double epsilon = 0.0000001

5.34.2 Function Documentation

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

Integrate the ordinary differential equation xdot from time to to time tf using an adaptive step size strategy.

adapted from: Numerical Recipes in C, The Art of Scientific Computing, Press, William H. and Flannery, Brian P. and Teukolsky, Saul A. and Vetterling, William T., Cambridge University Press, 1988.

Definition at line 347 of file utils.cpp.

References MAXSTP, rkqc(), and TINY.

Referenced by dynamics_demo().

5.34.2.2 ReturnMatrix pinv (const Matrix & *M*)

Matrix pseudo inverse 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 $1/\sigma_i < tol$

Definition at line 99 of file utils.cpp.

References epsilon, and pinv().

Referenced by pinv().

Index