Open Source Robotics with Scilab/Scicos

M. Morelli

mmorelli@users.sf.net

Centro “E.Piaggio”
Facoltà di Ingegneria
Università di Pisa
Pisa, Italy

1st HeDiSC Workshop On Open Source Software for Control Systems

Outline

1. Modelling robot manipulators with Scilab
   - Rigid body transformations
   - Building serial-link manipulator models
2. Simulating robot manipulators with Scicos
   - How RTSS works with Scicos
   - Control of robot manipulators in joint space
   - Developing RTAI-based centralized controllers with RTAI-Lab

RTSS–the Robotics Toolbox for Scilab/Scicos

MATLAB® Robotics Toolbox (MRT) as source of inspiration

About MRT [Corke, 1996]

- Developed by Dr. Peter Corke (CSIRO, Australia);
- very popular toolbox (more than 10500 downloads!);
- release 8 (December 2008) is under GNU LGPL;
- available at `http://petercorke.com/`.

Purpose of MRT

Enable students and teachers to better understand the theoretical concepts behind classical robotics.

RTSS–the Robotics Toolbox for Scilab/Scicos

MATLAB® Robotics Toolbox (MRT) as source of inspiration

Features

- Manipulation of fundamental datatypes such as:
  - homogeneous transformations;
  - quaternions;
  - trajectories.
- Functions for serial-link manipulators:
  - forward and inverse kinematics;
  - differential kinematics;
  - forward and inverse dynamics.
- SIMULINK® blockset library.

RTSS–the Robotics Toolbox for Scilab/Scicos

MATLAB® Robotics Toolbox (MRT) as source of inspiration

Points of strenght

Based on MATLAB®, MRT takes advantage of:

- a powerful environment for linear algebra;
- a powerful environment for graphical presentation;
- an easy integration with other toolboxes;
- the SIMULINK® environment for dynamic systems simulation.

Main drawback

MRT is an open source software, requiring a proprietary and expensive software environment to run.

RTSS–the Robotics Toolbox for Scilab/Scicos

The idea

Developing a port of MRT to Scilab/Scicos, a powerful open source environment for numerical computation.

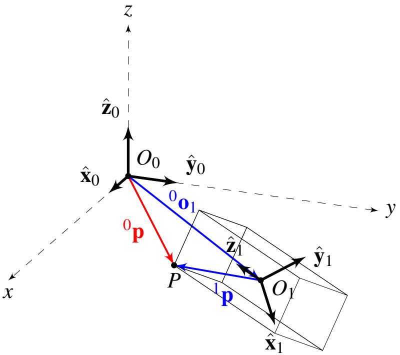
About RTSS

- Developed at Centro “E. Piaggio”, University of Pisa, since 2007;
- Free software licensed under GNU GPL;
- more than 3000 downloads;
- available at `http://rtss.sourceforge.net/`.

Outline

Representing 3D translations and orientations

Homogeneous transformations



Point P represented in different coordinate frames

Coordinate transformation

$${}^0\mathbf{p} = {}^0\mathbf{o}_1 + {}^0\mathbf{R}_1\, {}^1\mathbf{p}$$

Compact representation

$$\begin{bmatrix} {}^0\mathbf{p} \\ 1 \end{bmatrix} = \begin{bmatrix} {}^0\mathbf{R}_1 & {}^0\mathbf{o}_1 \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} {}^1\mathbf{p} \\ 1 \end{bmatrix}$$

Representing 3D translations and orientations

Playing with homogeneous transformations

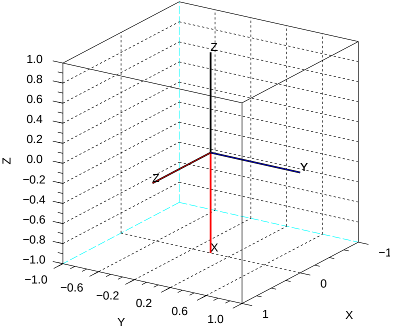
Example

Create the homogeneous transform

$$ {}^{w}\mathbf{T}_{b} = Transl_{\hat{\mathbf{x}}}(0.5\mathrm{m})Rot_{\hat{\mathbf{y}}}(\pi/2) $$

Solution

```
-->Twb = rt_transl(0.5, 0, 0)*..
      rt_roty(%pi/2),
 Twb  =

   0.    0.    1.    0.5
   0.    1.    0.    0.
 - 1.    0.    0.    0.
   0.    0.    0.    1.
```



Orientation of frame $\{\mathcal{B}\}$ (colored) with respect to frame $\{\mathcal{W}\}$ (black)

Representing 3D translations and orientations

Other representations of orientation

RTSS also provides full support for other representations:

- Euler angles (ZYZ);
- Roll/Pitch/Yaw angles;
- angle and axis;
- unit quaternion.

Euler angles (ZYZ)

Find a vector of Euler angles describing the rotational part of

$$^{w}\mathbf{T}_b = Transl_{\hat{\mathbf{x}}}(0.5\text{m})Rot_{\hat{\mathbf{y}}}(\pi/2) \\ Rot_{\hat{\mathbf{z}}}(-\pi/2)$$

Solution

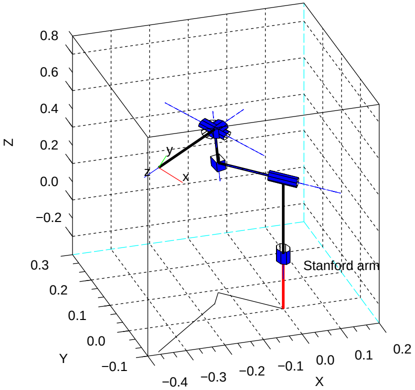
```
-->Twb = rt_transl(0.5, 0, 0) *..
            rt_roty(%pi/2) *..
               rt_rotz(-%pi/2);
-->Eul = rt_tr2eul(Twb),
 Eul =

   0.   1.5707963 - 1.5707963
```

Outline

1. Modelling robot manipulators with Scilab
   - Rigid body transformations
   - Building serial-link manipulator models
2. Simulating robot manipulators with Scicos
   - How RTSS works with Scicos
   - Control of robot manipulators in joint space
   - Developing RTAI-based centralized controllers with RTAI-Lab

On the robot modelling capabilities of RTSS

An overview



Visualization of Stanford arm at a generic pose – created by RTSS

Robot mechanical structure

- Robots with a fixed base;
- open kinematic chains;
- arbitrary number of constituent links;

Articulation between consecutive links

- Single DOF joint: revolute (R) or prismatic (P).

On the robot modelling capabilities of RTSS

N-DOF manipulator kinematic and dynamic model equations

Forward kinematic model equation

$$^{w}\mathbf{T}_e(\mathbf{q}) = {}^{w}\mathbf{T}_0 \underbrace{{}^{0}\mathbf{A}_1(q_1)\ldots{}^{i-1}\mathbf{A}_i(q_i)\ldots{}^{n-1}\mathbf{A}_n(q_n)}_{{}^{0}\mathbf{T}_n(\mathbf{q})} {}^{n}\mathbf{T}_e$$

Dynamic model equation (via Newton-Euler formulation)

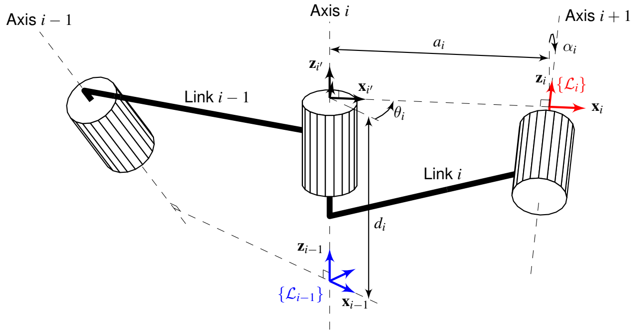
$$\mathbf{B}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{F}_v\dot{\mathbf{q}} + \mathbf{F}_s\mathbf{sgn}(\dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}) = \boldsymbol{\tau} - \mathbf{J}^T(\mathbf{q})\mathbf{h}_e$$

How to model the whole manipulator?

By describing the kinematic and the dynamic model of each joint-link pair in the manipulator.

Kinematic model of a joint-link pair

The standard Denavit-Hartenberg notation



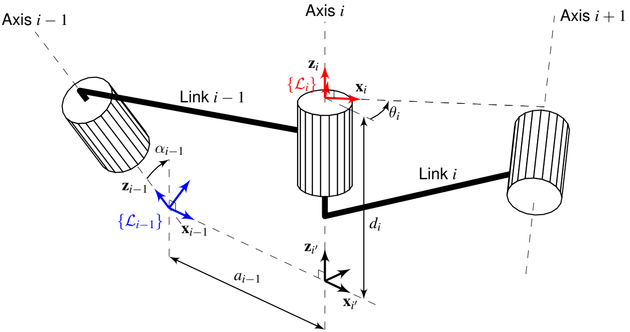
Standard DH frame assignment

Specifying $\{\mathcal{L}_i\}$ with respect to $\{\mathcal{L}_{i-1}\}$

$$^{i-1}\mathbf{A}_i(q_i) = Transl_{\hat{\mathbf{z}}}(d_i)Rot_{\hat{\mathbf{z}}}(\theta_i)Transl_{\hat{\mathbf{x}}}(a_i)Rot_{\hat{\mathbf{x}}}(\alpha_i)$$

Kinematic model of a joint-link pair

The modified Denavit-Hartenberg notation



Modified DH frame assignment

Specifying $\{\mathcal{L}_i\}$ with respect to $\{\mathcal{L}_{i-1}\}$

$$^{i-1}\mathbf{A}_i(q_i) = Rot_{\hat{\mathbf{x}}}(\alpha_{i-1})Transl_{\hat{\mathbf{x}}}(a_{i-1})Rot_{\hat{\mathbf{z}}}(\theta_i)Transl_{\hat{\mathbf{z}}}(d_i)$$

Dynamic model of a joint-link pair

Link inertial parameters (10)

m	link mass
r_x, r_y, r_z	link COM
$I_{xx}, I_{yy}, I_{zz}, I_{xy}, I_{xz}, I_{yz}$	six components of the inertia tensor about the link COM

Actuator model parameters (5)

J_m	moment of inertia of the rotor
G	reduction gear ratio: joint speed/link speed
B	viscous friction coefficient
τ_C^+	Coulomb friction (positive rotation) coefficient
τ_C^-	Coulomb friction (negative rotation) coefficient

Modelling a direct-drive planar elbow manipulator

Pelican: experimental robot arm at CICESE (Mexico), Robotics lab.

The Pelican prototype robot arm [Kelly et al., 2005]

Geometric and kinematic properties

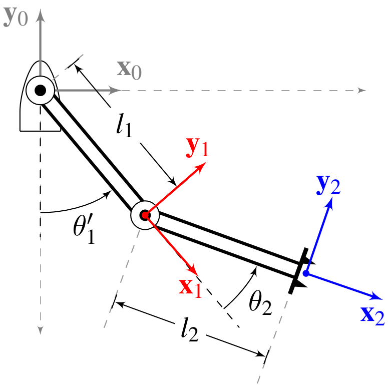
- Vertical planar manipulator;
- two rigid links connected via revolute joints.

Link	Notation	Value (m)
1	l_1	0.26
2	l_2	0.26

Link lengths of Pelican

Modelling a direct-drive planar elbow manipulator

A kinematic model based on the standard Denavit-Hartenberg notation



Standard DH frame assignment

Joint angle of axis 1 has an offset of $-\pi/2$ rad, according to DH notation.

Link	α_i	a_i	θ_i	d_i
1	0	l_1	${\theta_1'}^\star$	0
2	0	l_2	$\theta_2^\star$	0

Denavit-Hartenberg table

Modelling a direct-drive planar elbow manipulator

A kinematic model based on the standard Denavit-Hartenberg notation

Link	α_i	a_i	θ_i	d_i
1	0	l_1	$\theta_1'^{\star}$	0
2	0	l_2	$\theta_2^{\star}$	0

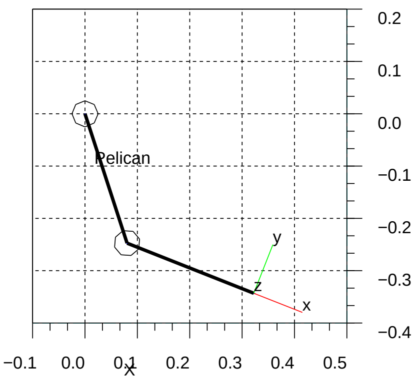
Denavit-Hartenberg table
(previous slide)

Building the kinematic model: the formal way

```
-->L1 = rt_link([0,l1,0,0,0], "standard");
-->L2 = rt_link([0,l2,0,0,0], "standard");
-->CL = list(L1, L2);
-->pel = rt_robot(CL, "Pelican",..
   "CICESE, Robotics Lab.",..
   "Kelly et al. 2005");
-->pel.offset = [-%pi/2; 0];
```

Modelling a direct-drive planar elbow manipulator

Model validation by simulation: comparison of results with reasonable expectations



Top view of Pelican pose at
$\mathbf{q} = [\ \pi/10 \quad 7\pi/25\ ]$ rad

Example: Where is the tool?

Pose: $\mathbf{q} = [\ \pi/10 \quad 7\pi/25\ ]$ rad

Forward kinematics

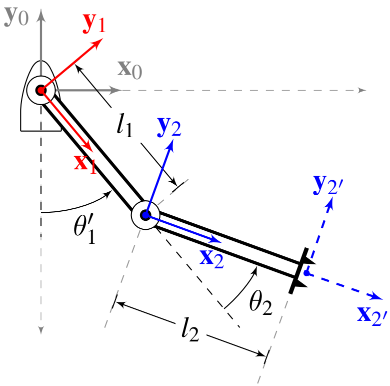
```
-->>q = [%pi/10, 7/25*%pi];
-->T02 = rt_fkine(pel, q),
 T02  =

    0.9298     0.3681     0.       0.3221
  - 0.3681     0.9298     0.     - 0.3430
    0.         0.         1.       0.
    0.         0.         0.       1.
```

Draw the Pelican pose

```
-->ws = [-0.1, 0.5, -0.4, 0.2, -1, 1];
-->rt_plot(pel, q, "workspace", ws);
```

Modelling a direct-drive planar elbow manipulator

An equivalent kinematic description using the modified Denavit-Hartenberg notation



Modified DH frame assignment

Link	α_{i-1}	a_{i-1}	θ_i	d_i
1	0	0	$\theta_1'^{\star}$	0
2	0	l_1	$\theta_2^{\star}$	0

Modified Denavit-Hartenberg table

Modelling a direct-drive planar elbow manipulator

An equivalent kinematic description using the modified Denavit-Hartenberg notation

Link	α_{i-1}	a_{i-1}	θ_i	d_i
1	0	0	$\theta_1'^{\star}$	0
2	0	l_1	$\theta_2^{\star}$	0

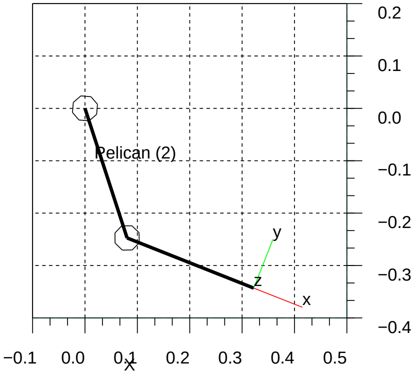
Modified Denavit-Hartenberg table (previous slide)

Building the kinematic model: the quick way

```
-->DH = [0,0,0,0;0,l1,0,0];
-->pelm = rt_robot(DH, "Pelican (2)",..
   "CICESE,Robotics Lab.", "Modified DH");
-->pelm.mdh = 1;
-->pelm.tool = rt_transl(l2,0,0);
-->pelm.offset = [-%pi/2; 0];
```

Modelling a direct-drive planar elbow manipulator

Model validation through kinematic simulation



MDH-based Pelican, at

$$\mathbf{q} = [\ \pi/10 \quad 7\pi/25\ ]\ \mathrm{rad}$$

Show that the two approaches are equivalent.

Forward kinematics

```
-->q = [%pi/10, 7*%pi/25];
-->T02m = rt_fkine(pelm, q),
 T02m  =

    0.9298    0.3681    0.     0.3221
  - 0.3681    0.9298    0.   - 0.3430
    0.        0.        1.     0.
    0.        0.        0.     1.

-->T02 - T02m
 ans  =

    0.    0.    0.    0.
    0.    0.    0.    0.
    0.    0.    0.    0.
    0.    0.    0.    0.
```

Modelling a direct-drive planar elbow manipulator

The dynamics of the Pelican arm: standard vs modified DH frame assignments

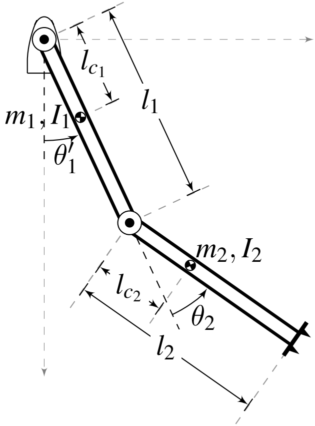


Diagram of the Pelican

Question

Which are the parameters that depend on the adopted frame assignment?

Modelling a direct-drive planar elbow manipulator

The dynamics of the Pelican arm: standard vs modified DH frame assignments

Parameter	Is it convention dependent?
Link mass	No
Inertia tensor about link COM	No (in this case of study)
Distance to link COM	Yes
Actuator/transmission	No

Dependence of physical parameters on the frame assignments

Modelling a direct-drive planar elbow manipulator

The dynamics of the Pelican arm: inertial parameters

Description	Notation	Value	Units
Mass of link 1	m_1	6.5225	kg
Mass of link 2	m_2	2.0458	kg
Inertia rel. to COM (link 1)	I_1	0.1213	$\mathrm{kg\ m^2}$
Inertia rel. to COM (link 2)	I_2	0.0116	$\mathrm{kg\ m^2}$

Convention-independent inertial parameters

Link	Notation	Value (m)	
		Standard DH	Modified DH
1	l_{c_1}	-0.1617	0.0983
2	l_{c_2}	-0.2371	0.0229

Distance to the link COM (convention-dependent)

Modelling a direct-drive planar elbow manipulator

The dynamics of the Pelican arm: actuator and transmission parameters

Actuators of Pelican

Two brushless DC motors located at the base and at the elbow.

Description	Motor/Joint 1 Sym.	Motor/Joint 1 Value	Motor/Joint 2 Sym.	Motor/Joint 2 Value	Units
Inertia (COM)	J_{m_1}	0.012	J_{m_2}	0.0025	$\mathrm{kg\ m^2}$
Gear ratio	G_1	1:1	G_2	1:1	
Viscous frict.	B_1	0.2741	B_2	0.1713	$\mathrm{Nm\ s/rad}$
Coulomb frict.	τ_{C_1}	1.29	τ_{C_2}	0.965	Nm

Actuator/transmission parameters

Modelling a direct-drive planar elbow manipulator

Simplified dynamic model based on the standard DH frame assignments

Robot model without actuators and transmissions.

Inertial parameters and gravity acceleration vector

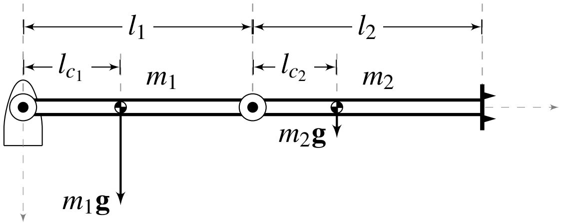
```
-->CL(1).I = [0,0,I1,0,0,0];
-->CL(2).I = [0,0,I2,0,0,0];
-->CL(1).m = m1;
-->CL(2).m = m2;
-->CL(1).r = [lc1;0;0];
-->CL(2).r = [lc2;0;0];
-->pel = rt_robot(pel, CL);
-->pel.gravity = [0;9.81;0]; // X-Y plane
```

Show 1st link data in detail

```
-->rt_showlink(pel.links(1)),
 // kinematic data displayed here...
 m     = 6.5225
!r     =    -0.1617  !
!                    !
!           0        !
!                    !
!           0        !
!I     =  0  0  0       !
!                       !
!         0  0  0       !
!                       !
!         0  0  0.1213  !
 Jm    =
 G     =
 B     = 0
!Tc    =  0  0  !
```

Modelling a direct-drive planar elbow manipulator

Robot dynamics model validation: reasonable expectations



Stretched Pelican (in X-direction)

Gravitational torque contribution at $\mathbf{q} = \begin{bmatrix} \pi/2 & 0 \end{bmatrix}$ $(\dot{\mathbf{q}} = \ddot{\mathbf{q}} = 0)$

$$\begin{bmatrix} \tau_{g_1} \\ \tau_{g_2} \end{bmatrix} = \begin{bmatrix} g(m_1 l_{c_1} + m_2(l_1 + l_{c_2})) \\ g m_2 (l_{c_2} + l_2) \end{bmatrix}$$

Modelling a direct-drive planar elbow manipulator

Robot dynamics model validation: simulation results

Comparison between expectations and simulation results

```
-->etau = [g*(m1*lc1 + m2*(l1 + lc2)), g*m2*lc2],
 etau =

   11.967401 0.4595869

-->etau - rt_frne(pel, [%pi/2,0], [0,0], [0,0]),
 ans =

   0. - 5.551D-17
```

Modelling a direct-drive planar elbow manipulator

Complete dynamic model based on the standard DH frame assignments

Robot model including the dynamics of actuators and transmissions.

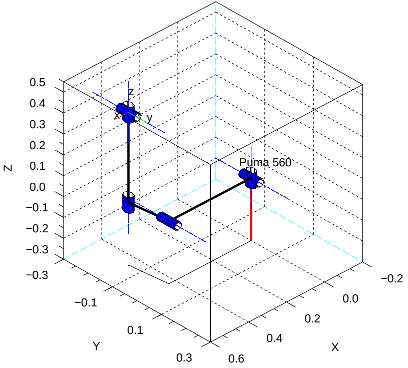
Modelling motor parameters

```
-->CL(1).Jm = Jm1;
-->CL(2).Jm = Jm2;
-->CL(1).G = G1;
-->CL(2).G = G2;
-->CL(1).B = B1;
-->CL(2).B = B2;
-->CL(1).Tc = Tc1*[1,1];
-->CL(2).Tc = Tc2*[1,1];
```

Launch the movie!

Ready-to-use manipulator models provided by RTSS

The Unimation Puma 560 arm



Standard DH-based Puma 560 at its zero angle pose

- Kinematic and dynamic data;
- Standard and modified DH-based models;
- quantities in standard SI units.

Create a Puma 560 robot

```
-->//Standard DH-based robot object (p560)
-->exec <PATH>/models/rt_puma560.sce;

-->//Modified DH-based robot object (p560m)
-->exec <PATH>/models/rt_puma560akb.sce;
```

Ready-to-use manipulator models provided by RTSS

The Stanford manipulator

Kinematic description of the Stanford manipulator

```
// alpha A   theta    D    sigma
stanford_dh = [..
-%pi/2    0     0     0.412 0
%pi/2     0     0     0.154 0
0         0   -%pi/2  0     1
-%pi/2    0     0     0     0
%pi/2     0     0     0     0
0         0     0     0.263 0   ];
```

- Kinematic and dynamic data;
- Standard DH-based model;
- quantities in standard SI units.

Create a Stanford manipulator

```
-->//Standard DH-based robot object (stanf)
-->exec <PATH>/models/rt_stanford.sce;
```

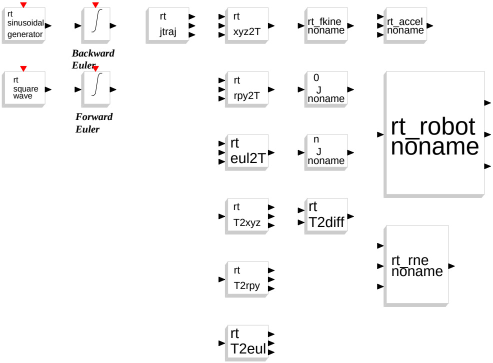
Outline

1. Modelling robot manipulators with Scilab
   - Rigid body transformations
   - Building serial-link manipulator models
2. Simulating robot manipulators with Scicos
   - How RTSS works with Scicos
   - Control of robot manipulators in joint space
   - Developing RTAI-based centralized controllers with RTAI-Lab

The `Robotics` palette

A library of ready-to-use blocks for robotics simulations

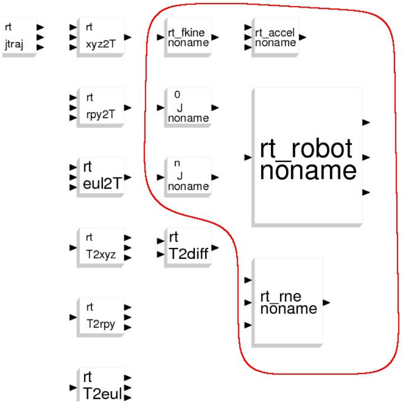
Available palettes after the installation of RTSS



Blocks in the `Robotics` palette

Proper setting of model-based blocks

How to set the robot model to be simulated in a block operating on robot objects



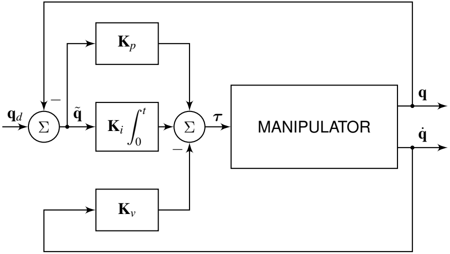
Blocks operating on robot objects

- Users must specify the robot model to be simulated as block parameter.
- It must be a symbolic parameter defined in the context of the diagram.

Outline

1. Modelling robot manipulators with Scilab
   - Rigid body transformations
   - Building serial-link manipulator models
2. Simulating robot manipulators with Scicos
   - How RTSS works with Scicos
   - Control of robot manipulators in joint space
   - Developing RTAI-based centralized controllers with RTAI-Lab

A centralized PID controller for regulation
Short review of the mathematics behind the PID control



PID controller for regulation
($\mathbf{q}_d = const.$)

The PID control law

$$\boldsymbol{\tau} = \mathbf{K}_p\tilde{\mathbf{q}} - \mathbf{K}_v\dot{\mathbf{q}} + \mathbf{K}_i \int_0^t \tilde{\mathbf{q}}(\sigma)\mathrm{d}\sigma$$

where

- $\tilde{\mathbf{q}} = \mathbf{q}_d - \mathbf{q} \in \mathbb{R}^n$;
- $\dot{\tilde{\mathbf{q}}} = -\dot{\mathbf{q}}$ (pos. control);
- $\mathbf{K}_p, \mathbf{K}_v, \mathbf{K}_i \in \mathbb{R}^{n\times n}$ symmetric and positive definite.

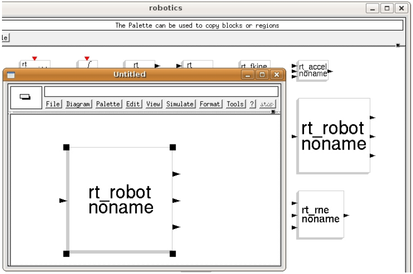
A centralized PID controller for regulation

Pelican's controller design in five steps (1)

Step 1: In an empty Scicos diagram, place a copy of the forward dynamics block (FDB).

Actions

1. Open up an empty diagram;
2. select the `Robotics` palette;
3. copy the FDB in the diagram.



The diagram after step 1

A centralized PID controller for regulation

Pelican's controller design in five steps (2)

Step 2: In the context of the diagram, define the symbolic parameters for all the blocks.

PID gains

$$\mathbf{K}_p = \mathrm{diag}\{30\} \qquad [\mathrm{Nm/rad}]$$
$$\mathbf{K}_v = \mathrm{diag}\{7, 3\} \qquad [\mathrm{Nm\ s/rad}]$$
$$\mathbf{K}_i = \mathrm{diag}\{70, 100\} \qquad [\mathrm{Nm/(rad\ s)}]$$

Initial and desired joint conditions

$$\mathbf{q}_0 = \dot{\mathbf{q}}_0 = \mathbf{0} \qquad [\mathrm{rad,\ rad/s}]$$
$$\mathbf{q}_d = \begin{bmatrix} \pi & \pi/3 \end{bmatrix}^T \qquad [\mathrm{rad}]$$

Scilab script in the context of the diagram

```
pelnf = rt_nofriction(pel,'coulomb');
qd = [%pi; %pi/3];
q0 = [0; 0];
Kp = diag([30,30]);
Kv = diag([7,3]);
Ki = diag([70,100]);
t_end = 6; // Final simulation time
```

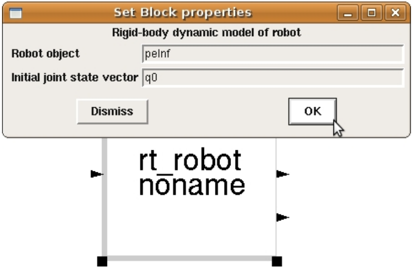
A centralized PID controller for regulation

Pelican's controller design in five steps (3)

Step 3: For the FDB, specify the robot model to be simulated as block parameter.

Actions

1. Open up the FDB;
2. Specify its block parameters;
3. Press the `OK` button.



FDB dialog box

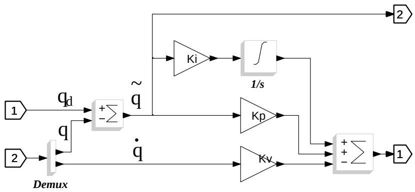
A centralized PID controller for regulation

Pelican's controller design in five steps (4)

Step 4: In an empty Super block (ESB), construct the PID controller.

Actions (palettes)

1. Copy an ESB in the diagram (`Others`);
2. edit the ESB. Connect:
   - I/O ports (`Sources`, `Sinks`);
   - Sums, Integrals, Gains (`Linear`);
   - Demux (`Branching`);
3. close the SB.



Scheme of the PID controller

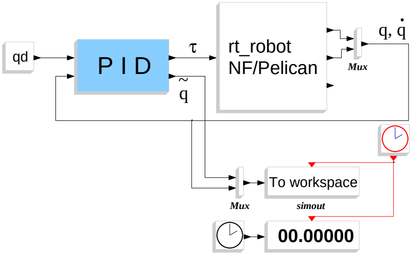
A centralized PID controller for regulation

Pelican's controller design in five steps (5)

Step 5: Complete the main diagram and construct the feedback loop.

Actions (palettes)

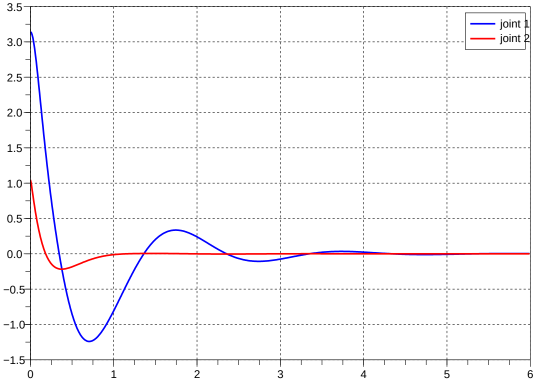
1. Complete the diagram. Copy:
   - Constant, Time, Activ. Clock (`Sources`);
   - Display, Workspace (`Sinks`);
   - Muxes (`Branching`);
2. connect the blocks.



The main diagram after step 5

A centralized PID controller for regulation

Analysis of simulation results

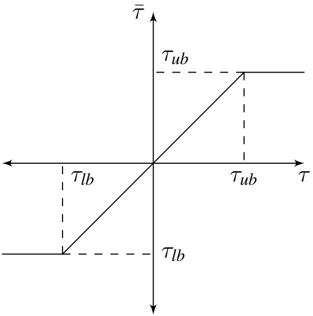


Graph of position errors

Launch the movie!

Considering physical limitations on the actuators
The problem of actuators saturation

All actuators saturate.



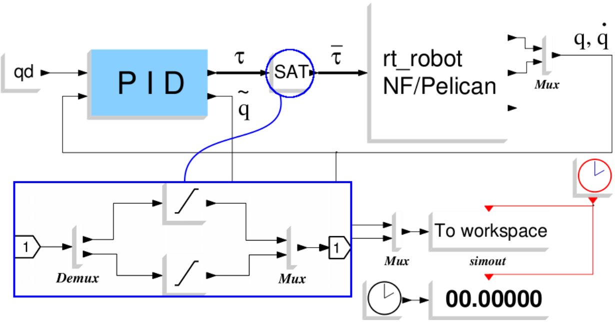
Curve of actuator saturation

Motor	Model	Peak torque (Nm)
1	1015-B	15
2	1004-C	4

DM series, motor data (Parker Compumotor)

Considering physical limitations on the actuators
Saturation in feedback loop

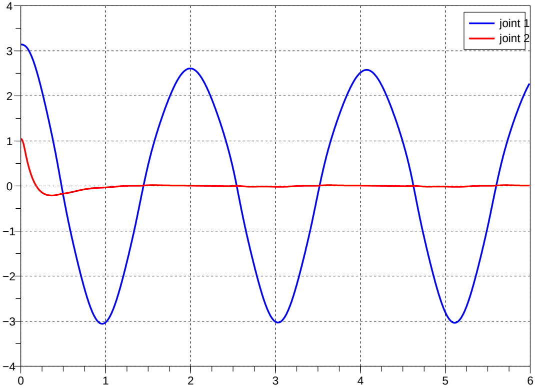
The presence of saturation elements in the feedback loop must be taken into account (`Non_linear` palette).



System with saturations of actuators

Considering physical limitations on the actuators

Analysis of simulation results: the phenomenon of integrator windup



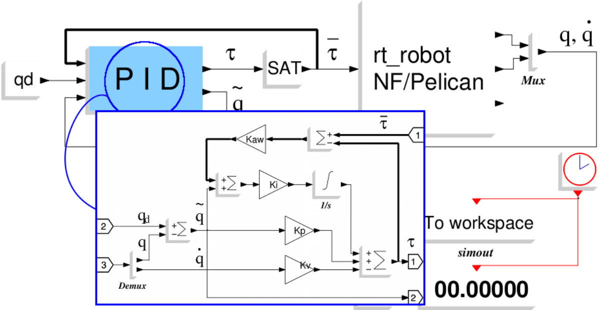
Graph of position errors

Launch the movie!

Considering physical limitations on the actuators

Anti-windup scheme with internal feedback

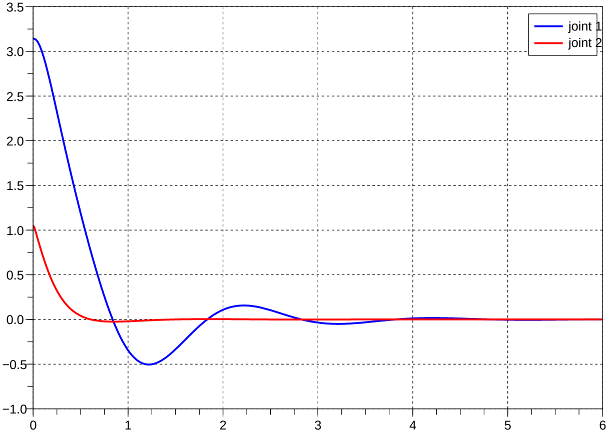
We should prevent integrator state from unstable updating as actuators saturate.



Internal controller feedback acting on τ

Considering physical limitations on the actuators

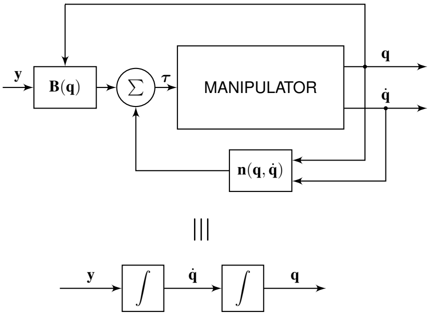
Analysis of simulation results



Graph of position errors

Model-based centralized controller for tracking

Short review of the mathematics behind the inverse dynamics control



Exact lineatization performed by inverse dynamics control

The dynamic model of a robot arm can be rewritten as

$$\mathbf{B}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{n}(\mathbf{q}, \dot{\mathbf{q}}) = \boldsymbol{\tau}$$

Inverse dynamics control law

$$\boldsymbol{\tau} = \mathbf{B}(\mathbf{q})\mathbf{y} + \mathbf{n}(\mathbf{q}, \dot{\mathbf{q}})$$

which leads the system to

$$\ddot{\mathbf{q}} = \mathbf{y}$$

Model-based centralized controller for tracking

Short review of the mathematics behind the inverse dynamics control

Typical choice for $\mathbf{y}$ is

$$\mathbf{y} = \ddot{\mathbf{q}}_d + \mathbf{K}_v\dot{\tilde{\mathbf{q}}} + \mathbf{K}_p\tilde{\mathbf{q}}$$

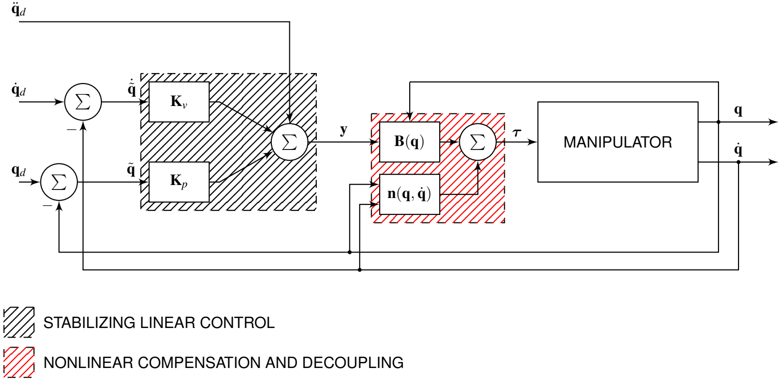
leading to the error dynamics equation

$$\ddot{\tilde{\mathbf{q}}} + \mathbf{K}_v\dot{\tilde{\mathbf{q}}} + \mathbf{K}_p\tilde{\mathbf{q}} = \mathbf{0}$$

which converges to zero with a speed depending on the matrices $\mathbf{K}_v$ and $\mathbf{K}_p$ chosen.

Model-based centralized controller for tracking
Inverse dynamics controller design

The inner feedback loop is based on the robot dynamic model.

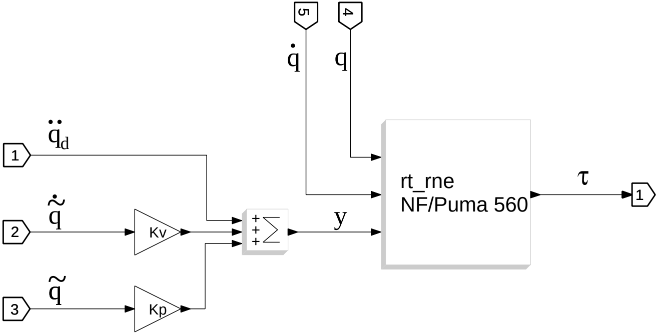


Block scheme of joint space inverse dynamics control

Model-based centralized controller for tracking

Scicos block diagram for joint space inverse dynamics controller

Key component: the inverse dynamics block (IDB) of the `Robotics` palette.



Inverse dynamics controller for the Puma 560 robot

Inverse dynamics controller design

Implementation and robustness issues

Pratical issues of the inverse dynamics control

- Parameters of the dynamical model must be accurately known;
- control input must be computed in real time.

In the case of imperfect compensation, the control vector $\boldsymbol{\tau}$ should be expressed as

$$\boldsymbol{\tau} = \hat{\mathbf{B}}(\mathbf{q})\mathbf{y}' + \hat{\mathbf{n}}(\mathbf{q}, \dot{\mathbf{q}})$$

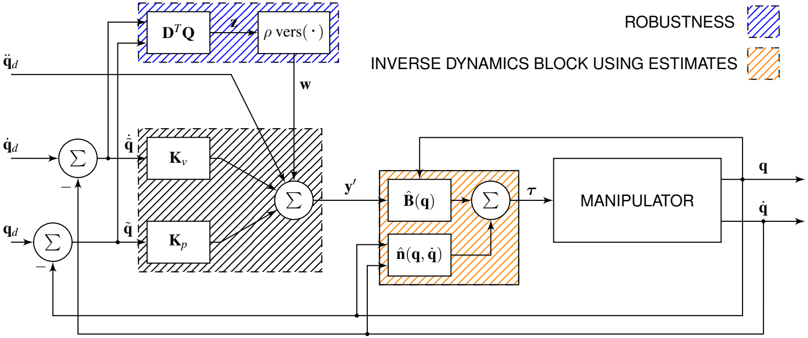
where

- $\hat{\mathbf{B}}$ and $\hat{\mathbf{n}}$ represent the estimates of $\mathbf{B}$ and $\mathbf{n}$,
- $\mathbf{y}' = \mathbf{y} + \mathbf{w}$ provides robustness to the control system.

Inverse dynamics controller design

Adding robustness to the inverse dynamics control system

$\mathbf{w}$ must guarantee robustness to the uncertainty described by $\hat{\mathbf{B}}$ and $\hat{\mathbf{n}}$.



Block scheme of joint space robust control

Inverse dynamics controller design

Including model uncertainty in the inverse dynamics controller

Estimates $\hat{\mathbf{B}}$ and $\hat{\mathbf{n}}$ can be modeled by using the function `rt_perturb()`.

How it works

It takes two inputs:

- Robot object (`rob`);
- perturb. percentage (`pp`).

It randomly modifies `rob` link masses and inertias in function of `pp`.

How to use it

- In the context of the diagram, “perturb” an existing robot;
- specify the modified robot as IDB block parameter.

Inverse dynamics controller design

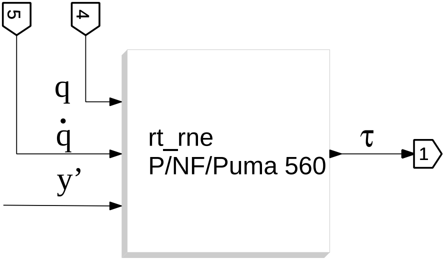
Including model uncertainty in the Puma 560 inverse dynamics controller

Example

Perturb Puma 560 link masses and inertias with a 25% disturb.

Scilab script in the context of the diagram

```
p560nf = rt_nofriction(p560,'coulomb');
p560p = rt_perturb(p560nf, 0.25);
// Other symbolic parameters
```



IDB with perturbed, friction-free Puma 560 as block parameter

First-order closed-loop inverse kinematics

On the generation of joint space reference inputs to the motion control system

Considerations regarding all the above control schemes

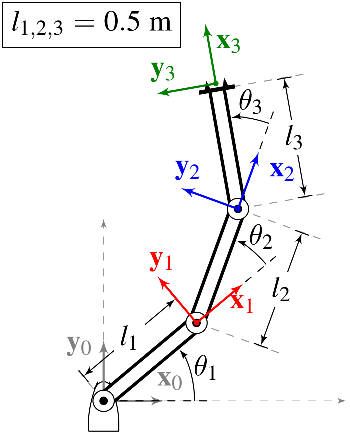
- Vectors $\mathbf{q}_d$, $\dot{\mathbf{q}}_d$ and $\ddot{\mathbf{q}}_d$ were always assumed available;
- joint space references were computed from two joint coordinate poses directly specified by the user.

However, motion specifications are usually assigned in the operational space.

Inverse kinematics algorithms transform task space references into joint space references.

First-order closed-loop inverse kinematics

Kinematic inversion of a simple three-link RRR planar arm



RRR arm at generic pose

Link	α_i	a_i	θ_i	d_i
1	0	l_1	$\theta_1^\star$	0
2	0	l_2	$\theta_2^\star$	0
3	0	l_3	$\theta_3^\star$	0

Denavit-Hartenberg table

Robot kinematic model

```
dh_123=[0,l_123,0,0];
R3arm = rt_robot([dh_123;dh_123;dh_123],..
        "RRR arm", "", "Sciavicco-Siciliano");
```

First-order closed-loop inverse kinematics

Kinematic inversion of a simple three-link RRR planar arm

Simulation scenario [Sciavicco and Siciliano, 2000]

- $\mathbf{q}(0) = \begin{bmatrix} \pi & -\pi/2 & -\pi/2 \end{bmatrix}^T$ [rad];
- circular desired motion trajectory for $0 \leq t \leq 4$ s:

$$\mathbf{x}_d(t) = \begin{bmatrix} \mathbf{p}_d(t) \\ \phi_d(t) \end{bmatrix} = \begin{bmatrix} 0.25(1 - \cos \pi t) \\ 0.25(2 + \sin \pi t) \\ \sin \frac{\pi}{24} t \end{bmatrix};$$

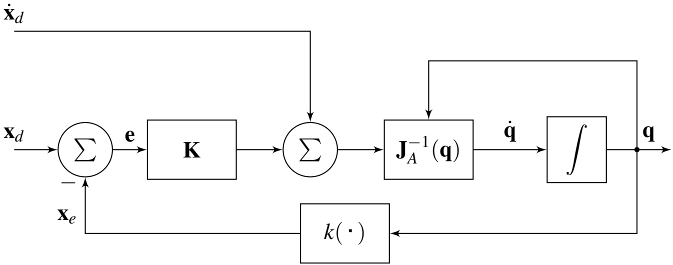
- forward Euler numerical integration scheme ($\Delta t = 1$ ms);
- final simulation time $t_{end} = 5$ s.

First-order closed-loop inverse kinematics

The jacobian inverse algorithm

The Jacobian inverse algorithm integrates the joint velocity vector

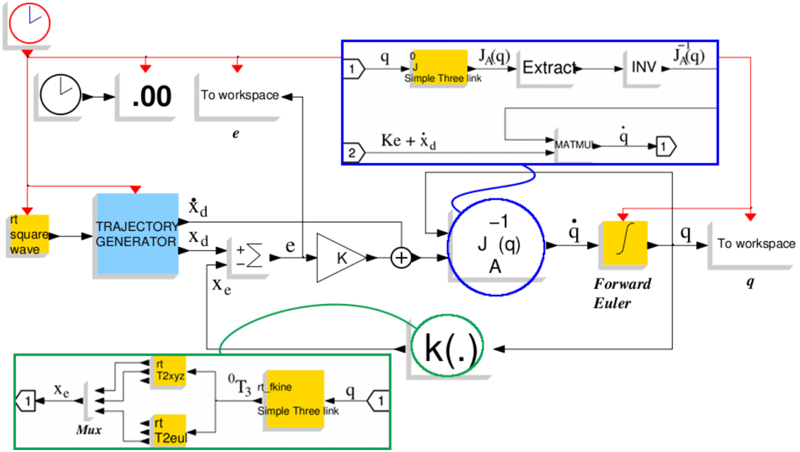
$$\dot{\mathbf{q}} = \mathbf{J}_A^{-1}(\mathbf{q})(\dot{\mathbf{x}}_d + \mathbf{K}\mathbf{e})$$



Jacobian inverse algorithm

First-order closed-loop inverse kinematics

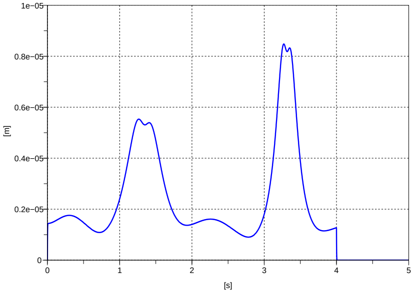
Scicos block diagram for the jacobian inverse algorithm



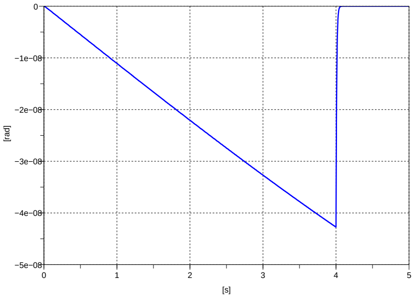
The jacobian inverse algorithm and the blocks $\mathbf{J}_A^{-1}(\mathbf{q})$ and $k(\cdot)$

First-order closed-loop inverse kinematics

Analysis of simulation results



Time history of the norm of end-effector position error



Time history of the end-effector orientation error

Launch the movie!

Outline

1. Modelling robot manipulators with Scilab
   - Rigid body transformations
   - Building serial-link manipulator models
2. Simulating robot manipulators with Scicos
   - How RTSS works with Scicos
   - Control of robot manipulators in joint space
   - Developing RTAI-based centralized controllers with RTAI-Lab

Inverse dynamics controller for the Pelican arm

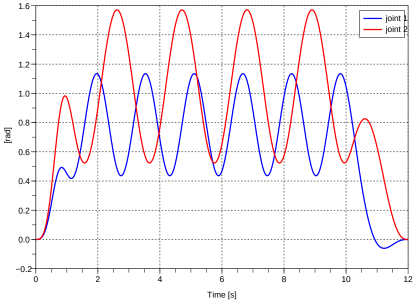
Desired reference trajectories in joint space

Positions

- Sinusoidal term (0–10 s);
- homing traj. (10–12 s);
- sampling time: $5\ \mathrm{ms}$.

Velocities and accelerations

- By direct differentiation.

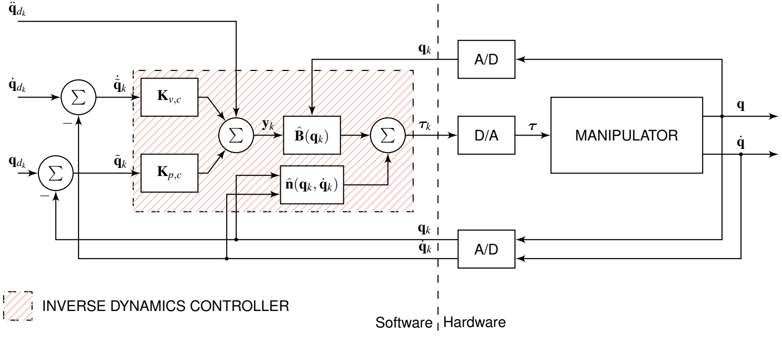


Graphs of the reference positions against time

Inverse dynamics controller for the Pelican arm

On the digital implementation of the control system

Digital implementation of the control law amounts to discretizing the inner and outer control loops.

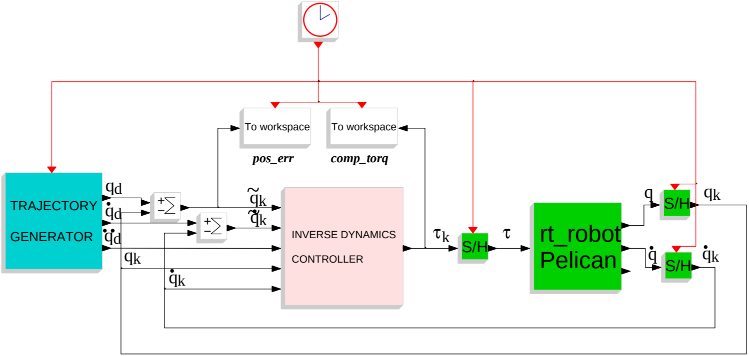


Digital implementation of the inverse dynamics control scheme

Inverse dynamics controller for the Pelican arm

Scicos block diagram for simulation

Trajectory generator uses a set of "From Workspace" blocks from the standard `Sources` palette.



Scicos block diagram for the control system (simulation)

Inverse dynamics controller for the Pelican arm

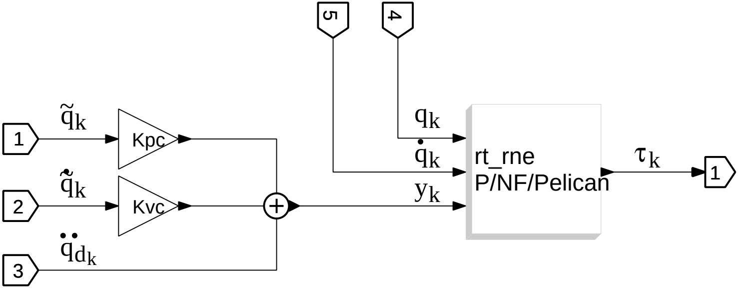
A look inside the "Inverse dynamics controller" block

Coded forms of the PD gains

$$
\begin{aligned}
\mathbf{K}_{p,c} &= \mathrm{diag}\{1500, 14000\} \quad [\mathrm{s}^{-2}] \\
\mathbf{K}_{v,c} &= \mathrm{diag}\{76.21, 353.71\} \quad [\mathrm{s}^{-1}]
\end{aligned}
$$

Context of the diagram

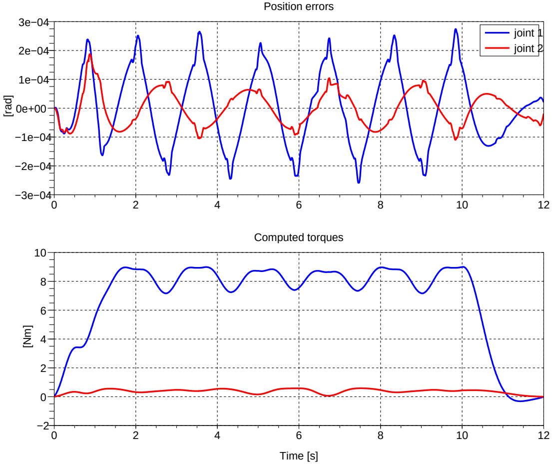
```
pelnf = rt_nofriction(pel, 'coulomb');
pelp = rt_perturb(pelnf, 0.25);
Kpc = diag([1500, 14000]);
Kvc = diag([76.21, 353.71]);
```



Block diagram for the digital controller

Inverse dynamics controller for the Pelican arm

Analysis of simulation results

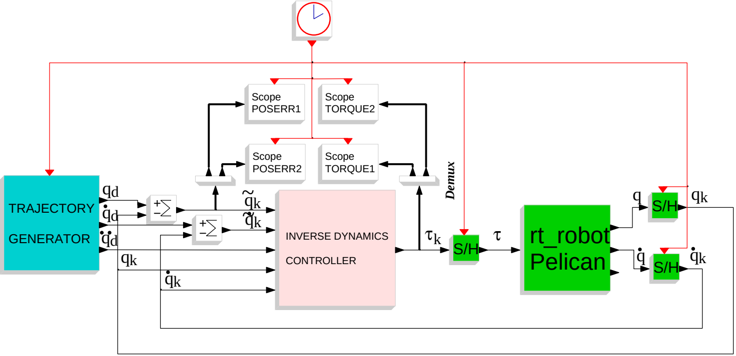


Graph of position errors and computed torques against the time

Inverse dynamics controller for the Pelican arm

Scicos block diagram for real time code generation

Difference between the diagrams for simulation and real time code generation: trajectory generator and scope blocks.

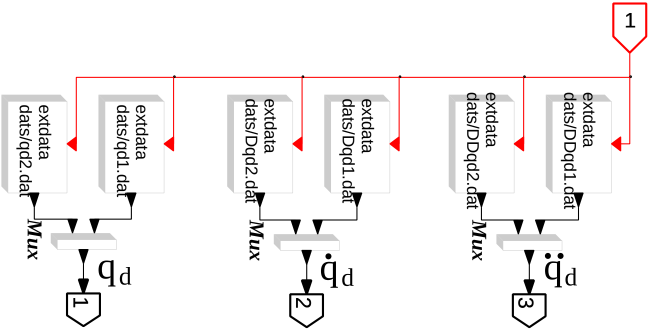


Scicos block diagram for the control system (real time control)

Inverse dynamics controller for the Pelican arm

A look inside the trajectory generator

Trajectory generator uses a set of “extdata” blocks from the `RTAI-Lib` palette.

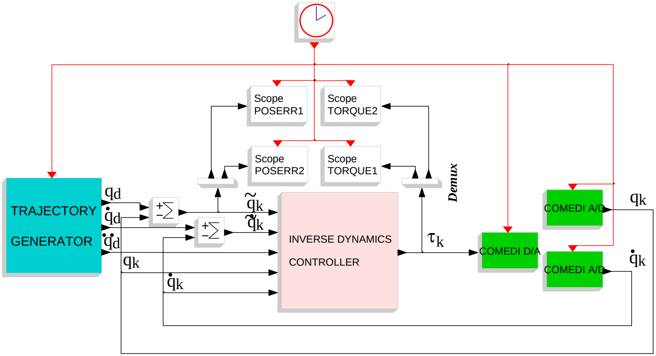


Block diagram for the trajectory generator (real time control)

Inverse dynamics controller for the Pelican arm

What if the robot arm were physically available?

The mathematical representation of the robot should be substituted by `COMEDI` DAC/ADC blocks from `RTAI-Lib`.



Block diagram for real time control with a set of `COMEDI` blocks

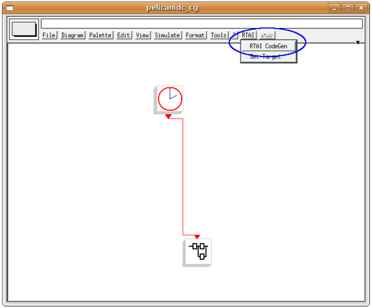
Inverse dynamics controller for the Pelican arm

Standalone, hard real time controller generation in two steps (1)

Step 1: Excluding the Clock, construct a super block (SB) out of the diagram for real time control (RTC).

Actions

1. Use the `Region-to-Super-block` facility to construct the SB;
2. click the `RTAI CodeGen` button;
3. click on the SB.

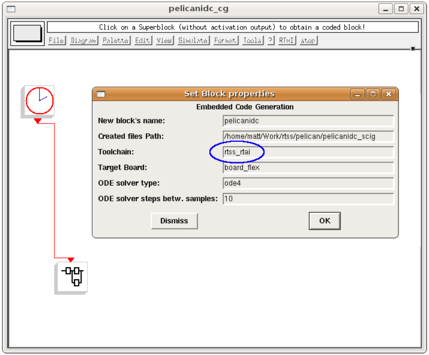


RTC diagram after step 1

Inverse dynamics controller for the Pelican arm

Standalone, hard real time controller generation in two steps (2)

Step 2: Adjust the properties for code generation by setting `rtss_rtai` as Toolchain and press `OK`.

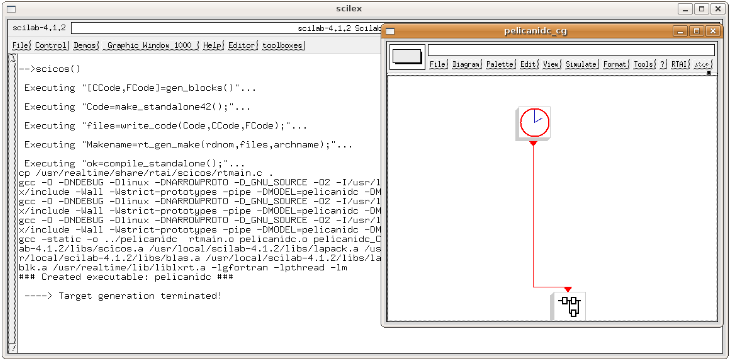


`RTAI CodeGen` dialog box

Inverse dynamics controller for the Pelican arm

Standalone, hard real time controller generation

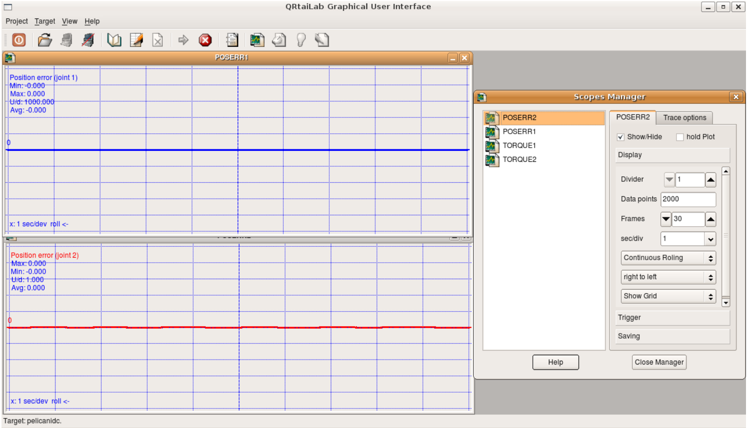
The compilation starts and completes. An executable file called `pelicanidc` is created.



Compilation output in the Scilab window

Inverse dynamics controller for the Pelican arm

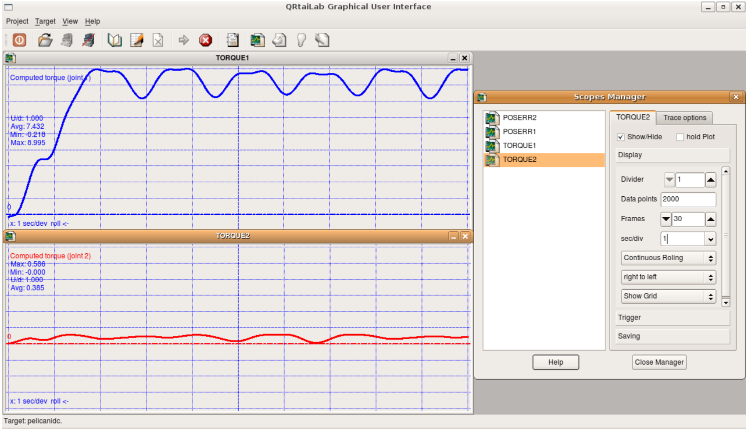
Monitoring the real time controller with the `QRtaiLab` Graphical User Interface



`QRtaiLab` with the scope manager and the position errors scopes

Inverse dynamics controller for the Pelican arm

Monitoring the real time controller with the `QRtaiLab` Graphical User Interface



`QRtaiLab` with the scope manager and the computed torques scopes

Summary

Main features of RTSS

- Modelling and simulation of robotic manipulators in the Scilab/Scicos environment;
- Development of soft/hard real time control systems with the Scicos-HIL toolbox and the Scicos RTAI Code Generator.

Current and future developments in RTSS

- Full compatibility with the Scicos RTAI Code Generator;
- support for 3D closed-chain robot systems;

Further readings about RTSS

General information: `http://rtss.sourceforge.net/`

- Introduction and key features;
- software download and licensing information;
- support and contributions.

Development reference source: `http://sourceforge.net/apps/mediawiki/rtss/`

- Roadmap and updates about the status of development;
- technical documentation for developers and advanced users;
- notes about the compatibility among different Scilab versions.

References

- P.I. Corke.
  A robotics toolbox for MATLAB.
  IEEE Robotics and Automation Magazine, vol. 3, pp. 24–32, Mar. 1996.
- R. Kelly, V. Santibáñez and A. Loría.
  Control of Robot Manipulators in Joint Space.
  Advanced Textbooks in Control and Signal Processing, Springer-Verlag, London, 2005.
- L. Sciavicco and B. Siciliano.
  Modelling and Control of Robot Manipulators.
  Advanced Textbooks in Control and Signal Processing, London, UK: Springer-Verlag, 2nd ed., 2000.