The stack of tasks

Florent Lamiraux, Olivier Stasse and Nicolas Mansard

CNRS-LAAS, Toulouse, France
The stack of tasks

Introduction

Theoretical foundations

Software
Outline

Introduction

Theoretical foundations

Software
The stack of tasks provides a control framework for real-time redundant manipulator control

- implementation of a data-flow,
- control of the graph by python scripting,
- task-based hierarchical control,
- portable: tested on HRP-2, Nao, Romeo.
Introduction

The stack of tasks provides a control framework for real-time redundant manipulator control

- implementation of a data-flow,
- control of the graph by python scripting,
- task-based hierarchical control,
- portable: tested on HRP-2, Nao, Romeo.
Introduction

The stack of tasks provides a control framework for real-time redundant manipulator control

- implementation of a data-flow,
- control of the graph by python scripting,
- task-based hierarchical control,
- portable: tested on HRP-2, Nao, Romeo.
Introduction

The stack of tasks provides a control framework for real-time redundant manipulator control

- implementation of a data-flow,
- control of the graph by python scripting,
- task-based hierarchical control,
- portable: tested on HRP-2, Nao, Romeo.
Introduction

The stack of tasks provides a control framework for real-time redundant manipulator control

- implementation of a data-flow,
- control of the graph by python scripting,
- task-based hierarchical control,
- portable: tested on HRP-2, Nao, Romeo.
Outline

Introduction

Theoretical foundations

Software
Rigid body $\mathcal{B}$

- Configuration represented by an homogeneous matrix

$$
M_{\mathcal{B}} = \begin{pmatrix}
R_{\mathcal{B}} & t_{\mathcal{B}} \\
0 & 0 & 0 & 1
\end{pmatrix} \in SE(3)
$$

$R_{\mathcal{B}} \in SO(3) \iff R_{\mathcal{B}}^T R_{\mathcal{B}} = I_3$ and $\det(R) = 1$

Point $x \in \mathbb{R}^3$ in local frame of $\mathcal{B}$ is moved to $y \in \mathbb{R}^3$ in global frame:

$$
\begin{pmatrix}
y \\
1
\end{pmatrix} = M_{\mathcal{B}} \begin{pmatrix}
x \\
1
\end{pmatrix}
$$
Rigid body $\mathcal{B}$

- Configuration represented by an homogeneous matrix

\[
M_\mathcal{B} = \begin{pmatrix}
R_\mathcal{B} & t_\mathcal{B} \\
0 & 0 & 0 & 1
\end{pmatrix} \in SE(3)
\]

$R_\mathcal{B} \in SO(3) \iff R_\mathcal{B}^T R_\mathcal{B} = I_3$ and $\det(R) = 1$

Point $x \in \mathbb{R}^3$ in local frame of $\mathcal{B}$ is moved to $y \in \mathbb{R}^3$ in global frame:

\[
\begin{pmatrix}
y \\
1
\end{pmatrix} = M_\mathcal{B} \begin{pmatrix}
x \\
1
\end{pmatrix}
\]
Rigid body $\mathcal{B}$

- Configuration represented by an homogeneous matrix

\[
M_{\mathcal{B}} = \begin{pmatrix}
R_{\mathcal{B}} & t_{\mathcal{B}} \\
0 & 0 & 0 & 1
\end{pmatrix} \in SE(3)
\]

- $R_{\mathcal{B}} \in SO(3) \iff R_{\mathcal{B}}^T R_{\mathcal{B}} = I_3$ and $\det(R) = 1$

Point $\mathbf{x} \in \mathbb{R}^3$ in local frame of $\mathcal{B}$ is moved to $\mathbf{y} \in \mathbb{R}^3$ in global frame:

\[
\begin{pmatrix}
\mathbf{y} \\
1
\end{pmatrix} = M_{\mathcal{B}} \begin{pmatrix}
\mathbf{x} \\
1
\end{pmatrix}
\]
Rigid body \( \mathcal{B} \)

- Velocity represented by \((\mathbf{v}_\mathcal{B}, \omega_\mathcal{B}) \in \mathbb{R}^6\) where
  \[
  \dot{R}_\mathcal{B} = \hat{\omega}_\mathcal{B} R_\mathcal{B}
  \]
  and
  \[
  \hat{\omega} = \begin{pmatrix}
  0 & -\omega_3 & \omega_2 \\
  \omega_3 & 0 & -\omega_1 \\
  -\omega_2 & \omega_1 & 0
  \end{pmatrix}
  \]
  is the matrix corresponding to the cross product operator

- Velocity of point \( P \) on \( \mathcal{B} \)
  \[
  \mathbf{v}_P = \dot{t}_\mathcal{B} + \omega_\mathcal{B} \times \overrightarrow{O_\mathcal{B}P}
  \]
  where \( O_\mathcal{B} \) is the origin of the local frame of \( \mathcal{B} \).
Rigid body $\mathcal{B}$

- Velocity represented by $(\mathbf{v}_\mathcal{B}, \omega_\mathcal{B}) \in \mathbb{R}^6$ where
  \[
  \dot{R}_\mathcal{B} = \hat{\omega}_\mathcal{B} R_\mathcal{B}
  \]
  and
  \[
  \hat{\omega} = 
  \begin{pmatrix}
  0 & -\omega_3 & \omega_2 \\
  \omega_3 & 0 & -\omega_1 \\
  -\omega_2 & \omega_1 & 0 
  \end{pmatrix}
  \]
  is the matrix corresponding to the cross product operator

- Velocity of point $P$ on $\mathcal{B}$
  \[
  \mathbf{v}_P = \dot{t}_\mathcal{B} + \omega_\mathcal{B} \times \overrightarrow{O_\mathcal{B}P}
  \]
  where $O_\mathcal{B}$ is the origin of the local frame of $\mathcal{B}$. 

The stack of tasks
Rigid body $\mathcal{B}$

- Velocity represented by $(v_\mathcal{B}, \omega_\mathcal{B}) \in \mathbb{R}^6$ where

$$\dot{R}_\mathcal{B} = \hat{\omega}_\mathcal{B} R_\mathcal{B}$$

and

$$\hat{\omega} = \begin{pmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{pmatrix}$$

is the matrix corresponding to the cross product operator

- Velocity of point $P$ on $\mathcal{B}$

$$v_P = \dot{t}_\mathcal{B} + \omega_\mathcal{B} \times O_\mathcal{B}P$$

where $O_\mathcal{B}$ is the origin of the local frame of $\mathcal{B}$. 
Configuration space

- **Robot**: set of rigid-bodies linked by joints $B_0, \cdots, B_m$.
- **Configuration**: position in space of each body.

$$q = (q_{waist}, \theta_1, \cdots \theta_{n-6}) \in SE(3) \times \mathbb{R}^{n-6}$$

$q_{waist} = (x, y, z, \text{roll}, \text{pitch}, \text{yaw})$

- Position of $B_i$ depends on $q$:

$$M_{B_i}(q) \in SE(3)$$
**Configuration space**

- Robot: set of rigid-bodies linked by joints $B_0, \cdots B_m$.
- Configuration: position in space of each body.

\[
q = (q_{waist}, \theta_1, \cdots \theta_{n-6}) \in SE(3) \times \mathbb{R}^{n-6}
\]
\[
q_{waist} = (x, y, z, roll, pitch, yaw)
\]

- Position of $B_i$ depends on $q$:

\[
M_{B_i}(q) \in SE(3)
\]
Configuration space

- Robot: set of rigid-bodies linked by joints $B_0, \cdots B_m$.
- Configuration: position in space of each body.

$$q = (q_{\text{waist}}, \theta_1, \cdots \theta_{n-6}) \in SE(3) \times \mathbb{R}^{n-6}$$

$q_{\text{waist}} = (x, y, z, \text{roll}, \text{pitch}, \text{yaw})$

- Position of $B_i$ depends on $q$:

$$M_{B_i}(q) \in SE(3)$$
Velocity

- **Velocity:**

\[
\dot{q} = (\dot{x}, \dot{y}, \dot{z}, \omega_x, \omega_y, \omega_z, \dot{\theta}_1, \cdots \dot{\theta}_{n-6})
\]

\[
\omega \in \mathbb{R}^3
\]

- **Velocity of \( B_i \)**

\[
\left( \begin{array}{c}
\mathbf{v}_{B_i} \\
\mathbf{\omega}_{B_i}
\end{array} \right) (q, \dot{q}) = J_{B_i}(q) \cdot \dot{q} \in \mathbb{R}^6
\]
Velocity

Velocity:

\[ \dot{q} = (\dot{x}, \dot{y}, \dot{z}, \omega_x, \omega_y, \omega_z, \dot{\theta}_1, \cdots \dot{\theta}_{n-6}) \]

\[ \omega \in \mathbb{R}^3 \]

Velocity of \( B_i \)

\[
\left( \begin{array}{c}
  \mathbf{v}_{B_i} \\
  \mathbf{w}_{B_i}
\end{array} \right)(\mathbf{q}, \mathbf{\dot{q}}) = J_{B_i}(\mathbf{q}) \cdot \mathbf{\dot{q}} \in \mathbb{R}^6
\]
Velocity

- Velocity:
  \[ \dot{q} = (\dot{x}, \dot{y}, \dot{z}, \omega_x, \omega_y, \omega_z, \dot{\theta}_1, \cdots \dot{\theta}_{n-6}) \]
  \[ \omega \in \mathbb{R}^3 \]

- Velocity of \( B_i \)
  \[ \begin{pmatrix} v_{B_i} \\ \omega_{B_i} \end{pmatrix} (q, \dot{q}) = J_{B_i}(q) \dot{q} \in \mathbb{R}^6 \]

The stack of tasks
Task

Definition: function of the
- robot configuration,
- time and
- possibly external parameters

that should converge to 0:

\[ T \in C^\infty(\mathcal{C} \times \mathbb{R}, \mathbb{R}^m) \]

Example: position tracking of an end-effector \( B_{ee} \)

- \( M(q) \in SE(3) \) position of the end-effector,
- \( M^*(t) \in SE(3) \) reference position

\[
T(q, t) = \begin{pmatrix}
    t(M^*-1(t)M(q)) \\
    u_\theta(R^*-1(t)R(q))
\end{pmatrix}
\]

where
- \( t() \) is the translation part of an homogeneous matrix,
- \( R \) and \( R^* \) are the rotation part of \( M \) and \( M^* \)
**Task**

- **Definition:** function of the robot configuration, time and possibly external parameters that should converge to 0:

\[ T \in C^\infty(C \times \mathbb{R}, \mathbb{R}^m) \]

- **Example:** position tracking of an end-effector \( B_{ee} \)
  - \( M(q) \in SE(3) \) position of the end-effector,
  - \( M^*(t) \in SE(3) \) reference position

\[ T(q, t) = \begin{pmatrix} t(M^*-1(t)M(q)) \\ u_\theta(R^*-1(t)R(q)) \end{pmatrix} \]

where
  - \( t() \) is the translation part of an homogeneous matrix,
  - \( R \) and \( R^* \) are the rotation part of \( M \) and \( M^* \).
Task

- Definition: function of the
  - robot configuration,
  - time and
  - possibly external parameters
that should converge to 0:

\[ T \in C^\infty(C \times \mathbb{R}, \mathbb{R}^m) \]

- Example: position tracking of an end-effector \( B_{ee} \)
  - \( M(q) \in SE(3) \) position of the end-effector,
  - \( M^*(t) \in SE(3) \) reference position

\[
T(q, t) = \begin{pmatrix}
  t(M^*^{-1}(t)M(q)) \\
  u_\theta(R^*^{-1}(t)R(q))
\end{pmatrix}
\]

where
- \( t() \) is the translation part of an homogeneous matrix,
- \( R \) and \( R^* \) are the rotation part of \( M \) and \( M^* \).
**Task**

- **Definition**: function of the
  - robot configuration,
  - time and
  - possibly external parameters

that should converge to 0:

\[ T \in C^\infty(C \times \mathbb{R}, \mathbb{R}^m) \]

- **Example**: position tracking of an end-effector \( B_{ee} \)
  - \( M(q) \in SE(3) \) position of the end-effector,
  - \( M^*(t) \in SE(3) \) reference position

\[
T(q, t) = \begin{pmatrix}
  t(M^*^{-1}(t)M(q)) \\
  u_\theta(R^*^{-1}(t)R(q))
\end{pmatrix}
\]

where
- \( t() \) is the translation part of an homogeneous matrix,
- \( R \) and \( R^* \) are the rotation part of \( M \) and \( M^* \).
Task

- Definition: function of the robot configuration, time and possibly external parameters that should converge to 0:

  \[ T \in \mathcal{C}^\infty(\mathcal{C} \times \mathbb{R}, \mathbb{R}^m) \]

- Example: position tracking of an end-effector \( B_{ee} \)
  - \( M(q) \in SE(3) \) position of the end-effector,
  - \( M^*(t) \in SE(3) \) reference position

  \[ T(q, t) = \begin{pmatrix} t(M^*-1(t)M(q)) \\ u_\theta(R^*-1(t)R(q)) \end{pmatrix} \]

where
- \( t() \) is the translation part of an homogeneous matrix,
- \( R \) and \( R^* \) are the rotation part of \( M \) and \( M^* \).
Hierarchical task based control

Given

- a configuration $q$,
- two tasks of decreasing priorities:
  - $T_1 \in C^\infty(C \times \mathbb{R}, \mathbb{R}^{m_1})$,
  - $T_2 \in C^\infty(C \times \mathbb{R}, \mathbb{R}^{m_2})$,

compute a control vector $\dot{q}$

- that makes $T_1$ converge toward 0 and
- that makes $T_2$ converge toward 0 if possible.
Hierarchical task based control

Given

- a configuration $q$,
- two tasks of decreasing priorities:
  - $T_1 \in C^\infty(C \times \mathbb{R}, \mathbb{R}^{m_1})$,
  - $T_2 \in C^\infty(C \times \mathbb{R}, \mathbb{R}^{m_2})$,

compute a control vector $\dot{q}$

- that makes $T_1$ converge toward 0 and
- that makes $T_2$ converge toward 0 if possible.
Hierarchical task based control

Jacobian:

- we denote
  \[ J_i = \frac{\partial T_i}{\partial q} \text{ for } i \in \{1, 2\} \]
- then
  \[ \forall q \in \mathcal{C}, \forall t \in \mathbb{R}, \forall \dot{q} \in \mathbb{R}^n, \dot{T}_i = J_i(q, t)\dot{q} + \frac{\partial T_i}{\partial t}(q, t) \]

We try to enforce

- \[ \dot{T}_1 = -\lambda_1 T_1 \implies T_1(t) = e^{-\lambda_1 t} T_1(0) \to 0 \]
- \[ \dot{T}_2 = -\lambda_2 T_2 \implies T_2(t) = e^{-\lambda_2 t} T_2(0) \to 0 \]
- \( \lambda_1 \) and \( \lambda_2 \) are called the gains associated to \( T_1 \) and \( T_2 \).
Hierarchical task based control

Jacobian:

- we denote
  
  \[ J_i = \frac{\partial T_i}{\partial q} \text{ for } i \in \{1, 2\} \]

- then
  
  \[ \forall q \in \mathcal{C}, \forall t \in \mathbb{R}, \forall \dot{q} \in \mathbb{R}^n, \quad \dot{T}_i = J_i(q, t)\dot{q} + \frac{\partial T_i}{\partial t}(q, t) \]

We try to enforce

- \[ \dot{T}_1 = -\lambda_1 T_1 \quad \Rightarrow \quad T_1(t) = e^{-\lambda_1 t} T_1(0) \to 0 \]
- \[ \dot{T}_2 = -\lambda_2 T_2 \quad \Rightarrow \quad T_2(t) = e^{-\lambda_2 t} T_2(0) \to 0 \]
- \( \lambda_1 \) and \( \lambda_2 \) are called the gains associated to \( T_1 \) and \( T_2 \).
Hierarchical task based control

Jacobian:
▶ we denote
  
  \[ J_i = \frac{\partial T_i}{\partial q} \text{ for } i \in \{1, 2\} \]

▶ then
  
  \[ \forall q \in C, \forall t \in \mathbb{R}, \forall \dot{q} \in \mathbb{R}^n, \dot{T}_i = J_i(q, t)\dot{q} + \frac{\partial T_i}{\partial t}(q, t) \]

We try to enforce
▶ \[ \dot{T}_1 = -\lambda_1 T_1 \quad \Rightarrow \quad T_1(t) = e^{-\lambda_1 t} T_1(0) \rightarrow 0 \]
▶ \[ \dot{T}_2 = -\lambda_2 T_2 \quad \Rightarrow \quad T_2(t) = e^{-\lambda_2 t} T_2(0) \rightarrow 0 \]
▶ \( \lambda_1 \) and \( \lambda_2 \) are called the gains associated to \( T_1 \) and \( T_2 \).
Hierarchical task based control

Jacobian:
  - we denote
    - $J_i = \frac{\partial T_i}{\partial q}$ for $i \in \{1, 2\}$
  - then
    - $\forall q \in \mathbb{C}, \forall t \in \mathbb{R}, \forall \dot{q} \in \mathbb{R}^n$, $\dot{T}_i = J_i(q, t) \dot{q} + \frac{\partial T_i}{\partial t}(q, t)$

We try to enforce
  - $\dot{T}_1 = -\lambda_1 T_1 \Rightarrow T_1(t) = e^{-\lambda_1 t} T_1(0) \rightarrow 0$
  - $\dot{T}_2 = -\lambda_2 T_2 \Rightarrow T_2(t) = e^{-\lambda_2 t} T_2(0) \rightarrow 0$

$\lambda_1$ and $\lambda_2$ are called the gains associated to $T_1$ and $T_2$. 
Hierarchical task based control

Jacobian:
- we denote $J_i = \frac{\partial T_i}{\partial q}$ for $i \in \{1, 2\}$
- then
  $\forall q \in C, \forall t \in \mathbb{R}, \forall \dot{q} \in \mathbb{R}^n, \quad \dot{T}_i = J_i(q, t)\dot{q} + \frac{\partial T_i}{\partial t}(q, t)$

We try to enforce
- $\dot{T}_1 = -\lambda_1 T_1 \quad \Rightarrow \quad T_1(t) = e^{-\lambda_1 t} T_1(0) \to 0$
- $\dot{T}_2 = -\lambda_2 T_2 \quad \Rightarrow \quad T_2(t) = e^{-\lambda_2 t} T_2(0) \to 0$
- $\lambda_1$ and $\lambda_2$ are called the gains associated to $T_1$ and $T_2$. 
Moore Penrose pseudo-inverse

Given a matrix $A \in \mathbb{R}^{m \times n}$, the Moore Penrose pseudo inverse $A^+ \in \mathbb{R}^{n \times m}$ of $A$ is the unique matrix satisfying:

\[
AA^+ A = A
\]
\[
A^+ AA^+ = A^+
\]
\[
(AA^+)^T = AA^+
\]
\[
(A^+ A)^T = A^+ A
\]

Given a linear system:

\[
Ax = b, \quad A \in \mathbb{R}^{m \times n}, \ x \in \mathbb{R}^n, \ b \in \mathbb{R}^m
\]

$x = A^+ b$ minimizes

- $\|Ax - b\|$ over $\mathbb{R}^n$,
- $\|x\|$ over $\text{argmin} \|Ax - b\|$. 

The stack of tasks
Moore Penrose pseudo-inverse

Given a matrix $A \in \mathbb{R}^{m \times n}$, the Moore Penrose pseudo inverse $A^+ \in \mathbb{R}^{n \times m}$ of $A$ is the unique matrix satisfying:

$$AA^+A = A$$
$$A^+AA^+ = A^+$$
$$(AA^+)^T = AA^+$$
$$(A^+A)^T = A^+A$$

Given a linear system:

$$Ax = b, \quad A \in \mathbb{R}^{m \times n}, \quad x \in \mathbb{R}^n, \quad b \in \mathbb{R}^m$$

$x = A^+b$ minimizes

- $\|Ax - b\|$ over $\mathbb{R}^n$,
- $\|x\|$ over $\text{argmin} \|Ax - b\|$.
Moore Penrose pseudo-inverse

Given a matrix $A \in \mathbb{R}^{m \times n}$, the Moore Penrose pseudo inverse $A^+ \in \mathbb{R}^{n \times m}$ of $A$ is the unique matrix satisfying:

\[
AA^+ A = A \\
A^+ AA^+ = A^+ \\
(AA^+)^T = AA^+ \\
(A^+ A)^T = A^+ A
\]

Given a linear system:

\[
Ax = b, \quad A \in \mathbb{R}^{m \times n}, \ x \in \mathbb{R}^n, \ b \in \mathbb{R}^m
\]

$x = A^+ b$ minimizes

- $\|Ax - b\|$ over $\mathbb{R}^n$,
- $\|x\|$ over $\text{argmin} \|Ax - b\|$. 

The stack of tasks
Moore Penrose pseudo-inverse

Given a matrix $A \in \mathbb{R}^{m \times n}$, the Moore Penrose pseudo inverse $A^+ \in \mathbb{R}^{n \times m}$ of $A$ is the unique matrix satisfying:

\[
AA^+ A = A
\]
\[
A^+ AA^+ = A^+
\]
\[
(AA^+)^T = AA^+
\]
\[
(A^+ A)^T = A^+ A
\]

Given a linear system:

\[
Ax = b, \quad A \in \mathbb{R}^{m \times n}, \; x \in \mathbb{R}^n, \; b \in \mathbb{R}^m
\]

$x = A^+ b$ minimizes

\[
\|Ax - b\| \text{ over } \mathbb{R}^n,
\]
\[
\|x\| \text{ over } \arg\min \|Ax - b\|.
\]
Hierarchical task based control

Resolution of the first constraint:

\[ \dot{T}_1 = J_1 \dot{q} + \frac{\partial T_1}{\partial t} = -\lambda_1 T_1 \]  \hspace{1cm} (1)

\[ J_1 \dot{q} = -\lambda_1 T_1 - \frac{\partial T_1}{\partial t} \]  \hspace{1cm} (2)

\[ \dot{q}_1 \triangleq -J_1^+ (\lambda_1 T_1 + \frac{\partial T_1}{\partial t}) \]  \hspace{1cm} (3)

Where \( J_1^+ \) is the (Moore Penrose) pseudo-inverse of \( J_1 \).

\( \dot{q}_1 \) minimizes

\[ \| J_1 \dot{q} + \lambda_1 T_1 + \frac{\partial T_1}{\partial t} \| = \| \dot{T}_1 + \lambda_1 T_1 \| \]

\[ \| \dot{q} \| \text{ over argmin } \| J_1 \dot{q} + \lambda_1 T_1 + \frac{\partial T_1}{\partial t} \| \]

Hence,

\[ \text{if } \lambda_1 T_1 + \frac{\partial T_1}{\partial t} \text{ is in } \text{Im}(J_1), \text{ (1) is satisfied} \]
Hierarchical task based control

Resolution of the first constraint:

\[
\dot{T}_1 = J_1 \dot{q} + \frac{\partial T_1}{\partial t} = -\lambda_1 T_1 \tag{1}
\]

\[
J_1 \dot{q} = -\lambda_1 T_1 - \frac{\partial T_1}{\partial t} \tag{2}
\]

\[
\dot{q}_1 \triangleq -J_1^+ (\lambda_1 T_1 + \frac{\partial T_1}{\partial t}) \tag{3}
\]

Where \(J_1^+\) is the (Moore Penrose) pseudo-inverse of \(J_1\).

\(\dot{q}_1\) minimizes

\[
\|J_1 \dot{q} + \lambda_1 T_1 + \frac{\partial T_1}{\partial t}\| = \|\dot{T}_1 + \lambda_1 T_1\|
\]

\[
\|\dot{q}\| \text{ over argmin } \|J_1 \dot{q} + \lambda_1 T_1 + \frac{\partial T_1}{\partial t}\|
\]

Hence,

\[
\text{if } \lambda_1 T_1 + \frac{\partial T_1}{\partial t} \text{ is in } \text{Im}(J_1), \text{ (1) is satisfied}
\]
Hierarchical task based control

Resolution of the first constraint:

\[
\dot{T}_1 = J_1 \dot{q} + \frac{\partial T_1}{\partial t} = -\lambda_1 T_1 \tag{1}
\]

\[J_1 \dot{q} = -\lambda_1 T_1 - \frac{\partial T_1}{\partial t} \tag{2}\]

\[
\dot{q}_1 \triangleq -J_1^+ (\lambda_1 T_1 + \frac{\partial T_1}{\partial t}) \tag{3}
\]

Where \(J_1^+\) is the (Moore Penrose) pseudo-inverse of \(J_1\).

\(\dot{q}_1\) minimizes

\[\|J_1 \dot{q} + \lambda_1 T_1 + \frac{\partial T_1}{\partial t}\| = \|\dot{T}_1 + \lambda_1 T_1\| \]

\[\|\dot{q}\| \text{ over argmin } \|J_1 \dot{q} + \lambda_1 T_1 + \frac{\partial T_1}{\partial t}\| \]

Hence,

\[\text{if } \lambda_1 T_1 + \frac{\partial T_1}{\partial t} \text{ is in } \text{Im}(J_1), \text{ (1) is satisfied}\]
Hierarchical task based control

Resolution of the first constraint:

\[ \dot{T}_1 = J_1 \dot{q} + \frac{\partial T_1}{\partial t} = -\lambda_1 T_1 \]  \hfill (1)

\[ J_1 \dot{q} = -\lambda_1 T_1 - \frac{\partial T_1}{\partial t} \]  \hfill (2)

\[ \dot{q}_1 \triangleq -J_1^+ (\lambda_1 T_1 + \frac{\partial T_1}{\partial t}) \]  \hfill (3)

Where \( J_1^+ \) is the (Moore Penrose) pseudo-inverse of \( J_1 \).

\( \dot{q}_1 \) minimizes

\[ \| J_1 \dot{q} + \lambda_1 T_1 + \frac{\partial T_1}{\partial t} \| = \| \dot{T}_1 + \lambda_1 T_1 \| \]

\[ \| \dot{q} \| \text{ over argmin } \| J_1 \dot{q} + \lambda_1 T_1 + \frac{\partial T_1}{\partial t} \| \]

Hence,

\[ \text{if } \lambda_1 T_1 + \frac{\partial T_1}{\partial t} \text{ is in } Im(J_1), \text{ (1) is satisfied} \]
Hierarchical task based control

Resolution of the first constraint:

\[ \dot{T}_1 = J_1 \dot{q} + \frac{\partial T_1}{\partial t} = -\lambda_1 T_1 \quad (1) \]

\[ J_1 \dot{q} = -\lambda_1 T_1 - \frac{\partial T_1}{\partial t} \quad (2) \]

\[ \dot{q}_1 \triangleq -J_1^+ (\lambda_1 T_1 + \frac{\partial T_1}{\partial t}) \quad (3) \]

Where \( J_1^+ \) is the (Moore Penrose) pseudo-inverse of \( J_1 \).

\( \dot{q}_1 \) minimizes

\[ \| J_1 \dot{q} + \lambda_1 T_1 + \frac{\partial T_1}{\partial t} \| = \| \dot{T}_1 + \lambda_1 T_1 \| \]

\[ \| \dot{q} \| \text{ over argmin } \| J_1 \dot{q} + \lambda_1 T_1 + \frac{\partial T_1}{\partial t} \| \]

Hence,

\[ \text{if } \lambda_1 T_1 + \frac{\partial T_1}{\partial t} \text{ is in } \text{Im}(J_1), \text{ (1) is satisfied} \]
Hierarchical task based control

In fact
\[ \forall u \in \mathbb{R}^n, \quad J_1 (\dot{q}_1 + (I_n - J_1^+ J_1) u) = J_1 \dot{q}_1 \]

therefore,
\[ \dot{q} = \dot{q}_1 + (I_n - J_1^+ J_1) u \]

also minimizes \[ \|J_1 \dot{q} + \lambda_1 T_1 + \frac{\partial T_1}{\partial t}\| \].

\[ P_1 = (I_n - J_1^+ J_1) \] is a projector on \( J_1 \) kernel:
\[ J_1 P_1 = 0 \]
\[ \forall u \in \mathbb{R}^n, \text{ if } \dot{q} = P_1 u, \text{ then, } \dot{T}_1 = \frac{\partial T_1}{\partial t}. \]
Hierarchical task based control

In fact
\[ \forall u \in \mathbb{R}^n, \quad J_1 \left( \dot{q}_1 + (I_n - J_1^+ J_1)u \right) = J_1 \dot{q}_1 \]

therefore,
\[ \dot{q} = \dot{q}_1 + (I_n - J_1^+ J_1)u \]

also minimizes
\[ \| J_1 \dot{q} + \lambda_1 T_1 + \frac{\partial T_1}{\partial t} \|. \]

\[ P_1 = (I_n - J_1^+ J_1) \] is a projector on \( J_1 \) kernel:
\[ J_1 P_1 = 0 \]
\[ \forall u \in \mathbb{R}^n, \text{ if } \dot{q} = P_1 u, \text{ then, } \dot{T}_1 = \frac{\partial T_1}{\partial t}. \]
Hierarchical task based control

In fact
\[ \forall u \in \mathbb{R}^n, \quad J_1 (\dot{q}_1 + (I_n - J_1^+ J_1)u) = J_1 \dot{q}_1 \]

therefore,
\[ \dot{q} = \dot{q}_1 + (I_n - J_1^+ J_1)u \]

also minimizes \[ \| J_1 \dot{q} + \lambda_1 T_1 + \frac{\partial T_1}{\partial t} \|. \]

\[ P_1 = (I_n - J_1^+ J_1) \] is a projector on \( J_1 \) kernel:
\[ J_1 P_1 = 0 \]
\[ \forall u \in \mathbb{R}^n, \text{ if } \dot{q} = P_1 u, \text{ then, } \dot{T}_1 = \frac{\partial T_1}{\partial t}. \]
Hierarchical task based control

In fact

$$\forall u \in \mathbb{R}^n, \quad J_1 (\dot{q}_1 + (I_n - J_1^+ J_1)u) = J_1 \dot{q}_1$$

therefore,

$$\dot{q} = \dot{q}_1 + (I_n - J_1^+ J_1)u$$

also minimizes \(\| J_1 \dot{q} + \lambda_1 T_1 + \frac{\partial T_1}{\partial t} \| \).

\(P_1 = (I_n - J_1^+ J_1)\) is a projector on \(J_1\) kernel:

\(J_1 P_1 = 0\)

\(\forall u \in \mathbb{R}^n, \text{ if } \dot{q} = P_1 u, \text{ then, } \dot{T}_1 = \frac{\partial T_1}{\partial t}.\)
Controlling the second task

We have

\[
\begin{align*}
\dot{\mathbf{q}} &= \dot{\mathbf{q}}_1 + P_1 u \\
\dot{T}_2 &= J_2 \dot{\mathbf{q}} + \frac{\partial T_2}{\partial t} \\
\dot{T}_2 &= J_2 \dot{\mathbf{q}}_1 + \frac{\partial T_2}{\partial t} + J_2 P_1 u
\end{align*}
\]

We want

\[
\dot{T}_2 = -\lambda_2 T_2
\]

Thus

\[
\begin{align*}
-\lambda_2 T_2 &= J_2 \dot{\mathbf{q}}_1 + \frac{\partial T_2}{\partial t} + J_2 P_1 u \\
J_2 P_1 u &= -\lambda_2 T_2 - J_2 \dot{\mathbf{q}}_1 - \frac{\partial T_2}{\partial t}
\end{align*}
\]
Controlling the second task

We have

\[ \dot{q} = \dot{q}_1 + P_1 u \]
\[ \dot{T}_2 = J_2 \dot{q} + \frac{\partial T_2}{\partial t} \]
\[ \dot{T}_2 = J_2 \dot{q}_1 + \frac{\partial T_2}{\partial t} + J_2 P_1 u \]

We want

\[ \dot{T}_2 = -\lambda_2 T_2 \]

Thus

\[ -\lambda_2 T_2 = J_2 \dot{q}_1 + \frac{\partial T_2}{\partial t} + J_2 P_1 u \]
\[ J_2 P_1 u = -\lambda_2 T_2 - J_2 \dot{q}_1 - \frac{\partial T_2}{\partial t} \]
Controlling the second task

We have

\[ \dot{q} = \dot{q}_1 + P_1 u \]
\[ \dot{T}_2 = J_2 \dot{q} + \frac{\partial T_2}{\partial t} \]
\[ \dot{T}_2 = J_2 \dot{q}_1 + \frac{\partial T_2}{\partial t} + J_2 P_1 u \]

We want

\[ \dot{T}_2 = -\lambda_2 T_2 \]

Thus

\[ -\lambda_2 T_2 = J_2 \dot{q}_1 + \frac{\partial T_2}{\partial t} + J_2 P_1 u \]
\[ J_2 P_1 u = -\lambda_2 T_2 - J_2 \dot{q}_1 - \frac{\partial T_2}{\partial t} \]
Controlling the second task

Thus

\[-\lambda_2 T_2 = J_2 \dot{q}_1 + \frac{\partial T_2}{\partial t} + J_2 P_1 u\]

\[J_2 P_1 u = -\lambda_2 T_2 - J_2 \dot{q}_1 - \frac{\partial T_2}{\partial t}\]

\[u = -(J_2 P_1)^+ (\lambda_2 T_2 + J_2 \dot{q}_1 + \frac{\partial T_2}{\partial t})\]

\[\dot{q}_2 \triangleq \dot{q}_1 + P_1 u\]

\[= \dot{q}_1 - P_1 (J_2 P_1)^+ (\lambda_2 T_2 + J_2 \dot{q}_1 + \frac{\partial T_2}{\partial t})\]

minimizes \(\| \dot{T}_2 + \lambda_2 T_2 \|\) over \(\dot{q}_1 + \text{Ker } J_1\).
Controlling the second task

Thus

\[-\lambda_2 T_2 = J_2 \dot{q}_1 + \frac{\partial T_2}{\partial t} + J_2 P_1 u\]

\[J_2 P_1 u = -\lambda_2 T_2 - J_2 \dot{q}_1 - \frac{\partial T_2}{\partial t}\]

\[u = -(J_2 P_1)^+ (\lambda_2 T_2 + J_2 \dot{q}_1 + \frac{\partial T_2}{\partial t})\]

\[\dot{q}_2 \triangleq \dot{q}_1 + P_1 u\]

\[= \dot{q}_1 - P_1 (J_2 P_1)^+ (\lambda_2 T_2 + J_2 \dot{q}_1 + \frac{\partial T_2}{\partial t})\]

minimizes \[\|\dot{T}_2 + \lambda_2 T_2\|\] over \[\dot{q}_1 + \text{Ker } J_1\].
Example

- $T_1$: position of the feet + projection of center of mass,
- $T_2$: position of the right wrist.
Outline

Introduction

Theoretical foundations

Software

The stack of tasks
Architecture overview

Introduction
Theoretical foundations
Software

- jrl-dynamics
- abstract-robot-dynamics
- jrl-mathtools
libraries
dynamic-graph
- entity
- signal
- command
- pool
- factory
- solver
- task
- feature
- robot
- pool
- controller
- robot
- dynamic
- forward kinematics
- inverse dynamics
sot-pattern-generator
- walk motion generation
sot-core
- solvers
- task
- feature
sot-tools
- helper tools
dynamic-graph-python
- bindings
- remote interpreter
dynamic-graph-bridge
services
/run_command
/start_dynamic_graph
/stop_dynamic_graph
topics
/joint_state
/from/to signals
ros-electric
sot-application
- application dependent initializations
sot-hrp2-hrpsys
OpenHRP-3.0
sot-hrptrc-hrp2
OpenHRP-3.1
sot-romeo
sot-nao
abstract controllers

The stack of tasks
Libraries

- **jrl-mathtools**: implementation of small size matrices,
  - to be replaced by Eigen
- **jrl-mal**: abstract layer for matrices,
  - to be replaced by Eigen
- **abstract-robot-dynamics**: abstraction for humanoid robot description,
- **jrl-dynamics**: implementation of the above abstract interfaces,
- **jrl-walkgen**: ZMP based dynamic walk generation.
Libraries

- **jrl-mathtools**: implementation of small size matrices,
  - to be replaced by Eigen
- **jrl-mal**: abstract layer for matrices,
  - to be replaced by Eigen
- **abstract-robot-dynamics**: abstraction for humanoid robot description,
- **jrl-dynamics**: implementation of the above abstract interfaces,
- **jrl-walkgen**: ZMP based dynamic walk generation.
Libraries

- **jrl-mathtools**: implementation of small size matrices,
  - to be replaced by Eigen
- **jrl-mal**: abstract layer for matrices,
  - to be replaced by Eigen
- **abstract-robot-dynamics**: abstraction for humanoid robot description,
- **jrl-dynamics**: implementation of the above abstract interfaces,
- **jrl-walkgen**: ZMP based dynamic walk generation.
Libraries

- **jrl-mathtools**: implementation of small size matrices,
  - to be replaced by Eigen
- **jrl-mal**: abstract layer for matrices,
  - to be replaced by Eigen
- **abstract-robot-dynamics**: abstraction for humanoid robot description,
- **jrl-dynamics**: implementation of the above abstract interfaces,
- **jrl-walkgen**: ZMP based dynamic walk generation.
Libraries

- **jrl-mathtools**: implementation of small size matrices,
  - to be replaced by Eigen
- **jrl-mal**: abstract layer for matrices,
  - to be replaced by Eigen
- **abstract-robot-dynamics**: abstraction for humanoid robot description,
- **jrl-dynamics**: implementation of the above abstract interfaces,
- **jrl-walkgen**: ZMP based dynamic walk generation.
dynamic-graph

- **Entity**
  - **Signal**: synchronous interface
  - **Command**: asynchronous interface

- **Factory**
  - builds a new entity of requested type,
  - new entity types can be dynamically added (advanced).

- **Pool**
  - stores all instances of entities,
  - return reference to entity of given name.
introduction

theoretical foundations

software

Dynamic-Graph

- Entity
  - Signal: synchronous interface
  - Command: asynchronous interface

- Factory
  - builds a new entity of requested type,
  - new entity types can be dynamically added (advanced).

- Pool
  - stores all instances of entities,
  - return reference to entity of given name.

The stack of tasks
dynamic-graph

- **Entity**
  - Signal: synchronous interface
  - Command: asynchronous interface
- **Factory**
  - builds a new entity of requested type,
  - new entity types can be dynamically added (advanced).
- **Pool**
  - stores all instances of entities,
  - return reference to entity of given name.
Signal (class SignalTimeDependent)

Synchronous interface storing a given data type

- output signals:
  - recomputed by a callback function, or
  - set to constant value
  - warning: setting to constant value deactivate callback,

- input signals:
  - plugged by an output signal, or
  - set to constant value,
  - warning: setting to constant value unplugs,
Signal (class SignalTimeDependent)

Synchronous interface storing a given data type

- **output signals:**
  - recomputed by a callback function, or
  - set to constant value
  - **warning:** setting to constant value deactivate callback,

- **input signals:**
  - plugged by an output signal, or
  - set to constant value,
  - **warning:** setting to constant value unplugs,
Signal \texttt{(class SignalTimeDependent)}

Synchronous interface storing a given data type

- output signals:
  - recomputed by a callback function, or
  - set to constant value
  - \textbf{warning}: setting to constant value deactivate callback,

- input signals:
  - plugged by an output signal, or
  - set to constant value,
  - \textbf{warning}: setting to constant value unplugs,
Signal (class SignalTimeDependent)

Synchronous interface storing a given data type

- output signals:
  - recomputed by a callback function, or
  - set to constant value
  - warning: setting to constant value deactivate callback,
- input signals:
  - plugged by an output signal, or
  - set to constant value,
  - warning: setting to constant value unplugs,
**Signal (class SignalTimeDependent)**

Synchronous interface storing a given data type

- dependency relation: \( s_1 \) depends on \( s_2 \) if \( s_1 \) callback needs the value of \( s_2 \),
- each signal \( s \) stores time of last recomputation in member \( s.t_- \),
- \( s \) is said outdated at time \( t \) if
  - \( t > s.t_- \), and
  - one dependency \( s_{dep} \) of \( s \)
    - is out-dated or
    - has been recomputed later than \( s \): \( s_{dep}.t_- > s.t_- \),
- reading an out-dated signal triggers recomputation.
- New types can be dynamically added (advanced)
Signal (class SignalTimeDependent)

Synchronous interface storing a given data type

- dependency relation: \( s_1 \) depends on \( s_2 \) if \( s_1 \) callback needs the value of \( s_2 \),
- each signal \( s \) stores time of last recomputation in member \( s.t_\),
- \( s \) is said outdated at time \( t \) if
  - \( t > s.t_\), and
  - one dependency \( s_{dep} \) of \( s \)
    - is out-dated or
    - has been recomputed later than \( s \): \( s_{dep}.t_\geq s.t_\).
- reading an out-dated signal triggers recomputation.
- New types can be dynamically added (advanced)
Signal (class SignalTimeDependent)

Synchronous interface storing a given data type

- dependency relation: \( s_1 \) depends on \( s_2 \) if \( s_1 \) callback needs the value of \( s_2 \),
- each signal \( s \) stores time of last recomputation in member \( s.t_\),
- \( s \) is said outdated at time \( t \) if
  - \( t > s.t_\), and
  - one dependency \( s_{dep} \) of \( s \)
    - is out-dated or
    - has been recomputed later than \( s \): \( s_{dep}.t_ > s.t_\).
- reading an out-dated signal triggers recomputation.
- New types can be dynamically added (advanced)
Signal (class SignalTimeDependent)

Synchronous interface storing a given data type

- dependency relation: $s_1$ depends on $s_2$ if $s_1$ callback needs the value of $s_2$,
- each signal $s$ stores time of last recomputation in member $s.t_-$
- $s$ is said outdated at time $t$ if
  - $t > s.t_-$, and
  - one dependency $s_{\text{dep}}$ of $s$
    - is out-dated or
    - has been recomputed later than $s$: $s_{\text{dep}.t_}-> s.t_-$
- reading an out-dated signal triggers recomputation.
- New types can be dynamically added (advanced)
Signal (class SignalTimeDependent)

Synchronous interface storing a given data type

▶ dependency relation: \( s_1 \) depends on \( s_2 \) if \( s_1 \) callback needs the value of \( s_2 \),

▶ each signal \( s \) stores time of last recomputation in member \( s.t_\)

▶ \( s \) is said outdated at time \( t \) if
  ▶ \( t > s.t_\), and
  ▶ one dependency \( s_{\text{dep}} \) of \( s \)
    ▶ is out-dated or
    ▶ has been recomputed later than \( s \): \( s_{\text{dep}}.t_\ > s.t_\).

▶ reading an out-dated signal triggers recomputation.

▶ New types can be dynamically added (advanced)
**Signal** (*class SignalTimeDependent*)

Synchronous interface storing a given data type

- dependency relation: $s_1$ depends on $s_2$ if $s_1$ callback needs the value of $s_2$,
- each signal $s$ stores time of last recomputation in member $s.t_\_$
- $s$ is said outdated at time $t$ if
  - $t > s.t_\_$, and
  - one dependency $s.dep$ of $s$
    - is out-dated or
    - has been recomputed later than $s$: $s.dep.t_\_ > s.t_\_$
- reading an out-dated signal triggers recomputation.
- New types can be dynamically added (advanced)
Command

Asynchronous interface

- input in a fixed set of types,
- trigger an action,
- returns a result in the same set of types.
**dynamic-graph-python**

**Python bindings to** *dynamic-graph*

- **module** `dynamic_graph` **linked to** `libdynamic-graph.so`
  - **class** `Entity`
    - each C++ entity class declared in the factory generates a python class of the same name,
    - signals are instance members,
    - commands are bound to instance methods
  - **method** `help` lists commands
  - **method** `displaySignals` displays signals
- **class** `Signal`
  - **property** `value` to set and get signal value
- **remote interpreter** to be embedded into a robot controller
  (advanced)
Python bindings to dynamic-graph

- module `dynamic_graph` linked to `libdynamic-graph.so`
  - class `Entity`
    - each C++ entity class declared in the factory generates a python class of the same name,
    - signals are instance members,
    - commands are bound to instance methods
    - method `help` lists commands
    - method `displaySignals` displays signals
  - class `Signal`
    - property `value` to set and get signal value
  - remote interpreter to be embedded into a robot controller (advanced)
Python bindings to `dynamic-graph`

- `module dynamic_graph` linked to `libdynamic-graph.so`
- `class Entity`
  - each C++ entity class declared in the factory generates a python class of the same name,
  - signals are instance members,
  - commands are bound to instance methods
  - `method help` lists commands
  - `method displaySignals` displays signals
- `class Signal`
  - `property value` to set and get signal value
- remote interpreter to be embedded into a robot controller (advanced)
Python bindings to `dynamic-graph`

- **module** `dynamic_graph` linked to `libdynamic-graph.so`
- **class** Entity
  - each C++ entity class declared in the factory generates a python class of the same name,
  - signals are instance members,
  - commands are bound to instance methods
  - method `help` lists commands
  - method `displaySignals` displays signals
- **class** Signal
  - property `value` to set and get signal value

- remote interpreter to be embedded into a robot controller (advanced)
Python bindings to `dynamic-graph`:

- Module `dynamic_graph` linked to `libdynamic-graph.so`
  - Class `Entity`
    - Each C++ entity class declared in the factory generates a Python class of the same name,
    - Signals are instance members,
    - Commands are bound to instance methods
    - Method `help` lists commands
    - Method `displaySignals` displays signals
  - Class `Signal`
    - Property `value` to set and get signal value
  - Remote interpreter to be embedded into a robot controller (advanced)
Simple use case for illustration
  ▶ Definition of 2 entity types
    ▶ InvertedPendulum
      ▶ input signal: force
      ▶ output signal: state
    ▶ FeedbackController
      ▶ input signal: state
      ▶ output signal: force
>>> from dynamic_graph.tutorial import InvertedPendulum, FeedbackController
>>> a = InvertedPendulum ("IP")
>>> b = FeedbackController ("K")
>>> a.displaySignals ()
--- <IP> signal list:
|-- <Sig:InvertedPendulum(IP)::input(double)::force (Type Cst) AUTOPLUGGED
`-- <Sig:InvertedPendulum(IP)::output(vector)::state (Type Cst)
>>> a.help ()
Classical inverted pendulum dynamic model

List of commands:
-----------------
  getCartMass: Get cart mass
  getPendulumLength: Get pendulum length
  getPendulumMass: Get pendulum mass
  incr: Integrate dynamics for time step provided as input
  setCartMass: Set cart mass
  setPendulumLength: Set pendulum length
  setPendulumMass: Set pendulum mass

>>> a.help ("incr")
incr:
  Integrate dynamics for time step provided as input
  take one floating point number as input

>>>
>>> from dynamic_graph.tutorial import InvertedPendulum, FeedbackController
>>> a = InvertedPendulum ('IP')
>>> b = FeedbackController ('K')
>>> a.displaySignals ()
--- <IP> signal list:
|-- <Sig:InvertedPendulum(IP)::input(double)::force (Type Cst) AUTOPLUGGED
`-- <Sig:InvertedPendulum(IP)::output(vector)::state (Type Cst)
>>> a.help ()
Classical inverted pendulum dynamic model

List of commands:
-----------------
    getCartMass: Get cart mass
    getPendulumLength: Get pendulum length
    getPendulumMass: Get pendulum mass
    incr: Integrate dynamics for time step provided as input
    setCartMass: Set cart mass
    setPendulumLength: Set pendulum length
    setPendulumMass: Set pendulum mass
>>> a.help ('incr')

    incr:
        Integrate dynamics for time step provided as input
        take one floating point number as input

>>>
Introduction
Theoretical foundations
Software

dynamic-graph-tutorial

>>> from dynamic_graph.tutorial import InvertedPendulum, FeedbackController
>>> a = InvertedPendulum ('IP')
>>> b = FeedbackController ('K')
>>> a.displaySignals ()
--- <IP> signal list:
|-- <Sig:InvertedPendulum(IP)::input(double)::force (Type Cst) AUTOPLUGGED
`-- <Sig:InvertedPendulum(IP)::output(vector)::state (Type Cst)
>>> a.help ()
Classical inverted pendulum dynamic model

List of commands:
-----------------
getCartMass: Get cart mass
getPendulumLength: Get pendulum length
getPendulumMass: Get pendulum mass
incr: Integrate dynamics for time step provided as input
setCartMass: Set cart mass
setPendulumLength: Set pendulum length
setPendulumMass: Set pendulum mass

>>> a.help ('incr')
incr:

Integrate dynamics for time step provided as input

take one floating point number as input

>>>
>>> from dynamic_graph.tutorial import InvertedPendulum, FeedbackController
>>> a = InvertedPendulum ('IP')
>>> b = FeedbackController ('K')
>>> a.displaySignals ()
--- <IP> signal list:
|-- <Sig:InvertedPendulum(IP)::input(double)::force (Type Cst) AUTOPLUGGED
|-- <Sig:InvertedPendulum(IP)::output(vector)::state (Type Cst)
>>> a.help ()
Classical inverted pendulum dynamic model

List of commands:
----------------
getCartMass: Get cart mass
getPendulumLength: Get pendulum length
getPendulumMass: Get pendulum mass
incr: Integrate dynamics for time step provided as input
setCartMass: Set cart mass
setPendulumLength: Set pendulum length
setPendulumMass: Set pendulum mass

>>> a.help ('incr')
incr:

Integrate dynamics for time step provided as input

   take one floating point number as input

>>>
>>> from dynamic_graph.tutorial import InvertedPendulum, FeedbackController
>>> a = InvertedPendulum ('IP')
>>> b = FeedbackController ('K')
>>> a.displaySignals ()
--- <IP> signal list:
|-- <Sig:InvertedPendulum(IP)::input(double)::force (Type Cst) AUTOPLUGGED
'-- <Sig:InvertedPendulum(IP)::output(vector)::state (Type Cst)
>>> a.help ()
Classical inverted pendulum dynamic model

List of commands:
-----------------
    getCartMass: Get cart mass
    getPendulumLength: Get pendulum length
    getPendulumMass: Get pendulum mass
    incr: Integrate dynamics for time step provided as input
    setCartMass: Set cart mass
    setPendulumLength: Set pendulum length
    setPendulumMass: Set pendulum mass

>>> a.help ('incr')
incr:

Integrate dynamics for time step provided as input

    take one floating point number as input

>>>
dynamic-graph-tutorial

Package provides

- **C++ code of classes** `InvertedPendulum` and `FeedbackController`,
- explanation about how to create a new entity type in C++,
- information about how to create a command in C++,
- information about how to create a python module defining the bindings in cmake,
- python script that runs an example.
dynamic-graph-tutorial

Package provides

- C++ code of classes InvertedPendulum and FeedbackController,
- explanation about how to create a new entity type in C++,
- information about how to create a command in C++,
- information about how to create a python module defining the bindings in cmake,
- python script that runs an example.
Package provides

- C++ code of classes `InvertedPendulum` and `FeedbackController`,
- explanation about how to create a new entity type in C++,
- information about how to create a command in C++,
- information about how to create a python module defining the bindings in cmake,
- python script that runs an example.
Package provides

- **C++ code of classes** `InvertedPendulum` and `FeedbackController`,
- explanation about how to create a new entity type in C++,
- information about how to create a command in C++,
- information about how to create a python module defining the bindings in cmake,
- python script that runs an example.
dynamic-graph-tutorial

Package provides

- **C++ code of classes** `InvertedPendulum` and `FeedbackController`,
- explanation about how to create a new entity type in C++,
- information about how to create a command in C++,
- information about how to create a python module defining the bindings in cmake,
- python script that runs an example.
sot-core

Class FeatureAbstract

- function of the robot and environment states
  - position of an end-effector,
  - position of a feature in an image (visual servoing)
- with values in a Lie group $G$ ($SO(3)$, $SE(3)$, $\mathbb{R}^n$, ...),
- with a mapping $e$ from $G$ into $\mathbb{R}^m$ such that

$$e(0_G) = 0$$
sot-core

Class FeatureAbstract

- function of the robot and environment states
  - position of an end-effector,
  - position of a feature in an image (visual servoing)
- with values in a Lie group $G$ ($SO(3)$, $SE(3)$, $\mathbb{R}^n$, ...),
- with a mapping $e$ from $G$ into $\mathbb{R}^m$ such that
  \[ e(0_G) = 0 \]
Class **FeatureAbstract**

- function of the robot and environment states
  - position of an end-effector,
  - position of a feature in an image (visual servoing)
- with values in a Lie group $G$ ($SO(3)$, $SE(3)$, $\mathbb{R}^n$, ...),
- with a mapping $e$ from $G$ into $\mathbb{R}^m$ such that

\[
e(0_G) = 0
\]
Class FeatureAbstract

- function of the robot and environment states
  - position of an end-effector,
  - position of a feature in an image (visual servoing)
- with values in a Lie group $G$ ($SO(3)$, $SE(3)$, $\mathbb{R}^n$, ...),
- with a mapping $e$ from $G$ into $\mathbb{R}^m$ such that

$$e(0_G) = 0$$
When paired with a reference, features become *tasks*.

**Example**

\[
\text{error} = e(\text{value.position} - \text{reference.position})
\]

\[
\text{errordot: derivative of error when value.position is constant.}
\]
Feature

When paired with a reference, features become *tasks*.

▶ Example

```
FeaturePoint6d

position
velocity
```

```
FeaturePoint6d

Jq
position

error
errordot
jacobian
```

▶ error = \( e(\text{value}.\text{position} - \text{reference}.\text{position}) \)

▶ errordot: derivative of error when value.position is constant.

The stack of tasks
Feature

When paired with a reference, features become tasks.

Example

\[ \text{error} = e(\text{value.position} \ominus \text{reference.position}) \]

\[ \text{errordot}: \text{derivative of error when value.position is constant.} \]
Task

- Collection of features with a control gain,
- implements abstraction TaskAbstract

\[ \text{task} = -\text{controlGain} \cdot \text{error} \]
Solver SOT

Hierarchical task solver

- computes robot joint velocity
sot-dynamic

dynamic_graph.sot.dynamics.Dynami**c** builds a kinematic chain from a file and
  ▶ computes forward kinematics
    ▶ position and Jacobian of end effectors (wrists, ankles),
    ▶ position of center of mass
  ▶ computes dynamics
    ▶ inertia matrix.
sot-pattern-generator

dynamic_graph.sot.pattern_generator

- **Entity** PatternGenerator **produces** walk motions as
  - position and velocity of the feet
  - position and velocity of the center of mass
sot-application

dynamic_graph.sot.application

- Provide scripts for standard control graph initialization
  - depends on application: control mode (velocity, acceleration)
Packages specific to robots

sot-hrp2

- defines a class `Robot` that provides
  - ready to use features for feet, hands, gaze and center of mass,
  - ready to use tasks for the same end effectors,
  - an entity `Dynamic`,
  - an entity `Device` (interface with the robot control system)

sot-hrprtc-hrp2

- provide an RTC component to integrate sot-hrp2 into the robot controller.
Utilities

- `dynamic_graph.writeGraph (filename)`: writes the current graph in a file using graphviz dot format.
- `dynamic_graph.sot.core.FeaturePosition` wraps two `FeaturePoint6d`: a value and a reference,
- `MetaTask6d`:
- `MetaTaskPosture`:
- `MetaTaskKine6d`:
- `MetaTaskKinePosture`:
- `MetaTaskCom`:
Utilities

- `dynamic_graph.writeGraph (filename)`: writes the current graph in a file using graphviz dot format.
- `dynamic_graph.sot.core.FeaturePosition` wraps two `FeaturePoint6d`: a value and a reference,
- `MetaTask6d`:
- `MetaTaskPosture`:
- `MetaTaskKine6d`:
- `MetaTaskKinePosture`:
- `MetaTaskCom`:
Utilities

- `dynamic_graph.writeGraph (filename)`: writes the current graph in a file using graphviz dot format.
- `dynamic_graph.sot.core.FeaturePosition` wraps two `FeaturePoint6d`: a value and a reference,
- MetaTask6d:
  - MetaTaskPosture:
  - MetaTaskKine6d:
  - MetaTaskKinePosture:
  - MetaTaskCom:
Utilities

- `dynamic_graph.writeGraph (filename)`: writes the current graph in a file using graphviz dot format.
- `dynamic_graph.sot.core.FeaturePosition` wraps two `FeaturePoint6d`: a value and a reference,
- `MetaTask6d`:
- `MetaTaskPosture`:
  - `MetaTaskKine6d`:
  - `MetaTaskKinePosture`:
  - `MetaTaskCom`:
Utilities

- `dynamic_graph.writeGraph (filename)`: writes the current graph in a file using graphviz dot format.
- `dynamic_graph.sot.core.FeaturePosition` wraps two `FeaturePoint6d`: a value and a reference,
- `MetaTask6d`:
- `MetaTaskPosture`:
- `MetaTaskKine6d`:
- `MetaTaskKinePosture`:
- `MetaTaskCom`:

The stack of tasks
Utilities

- `dynamic_graph.writeGraph (filename)`: writes the current graph in a file using graphviz dot format.
- `dynamic_graph.sot.core.FeaturePosition` wraps two `FeaturePoint6d`: a value and a reference,
- MetaTask6d:
- MetaTaskPosture:
- MetaTaskKine6d:
- MetaTaskKinePosture:
- MetaTaskCom:
Utilities

- `dynamic_graph.writeGraph (filename)`: writes the current graph in a file using graphviz dot format.
- `dynamic_graph.sot.core.FeaturePosition` wraps two `FeaturePoint6d`: a value and a reference,
- `MetaTask6d`:
- `MetaTaskPosture`:
- `MetaTaskKine6d`:
- `MetaTaskKinePosture`:
- `MetaTaskCom`:
Installation

Through robotpkg

- git clone http://trac.laas.fr/git/robots/robotpkg.git
- cd robotpkg
- ./bootstrap/bootstrap --prefix=<your_prefix>
- cd motion/sot-dynamic

make install
Installation

Through github:

```
for each package,
  mkdir package/build
  cd package/build
  cmake -DCMAKE_INSTALL_PREFIX=<your.prefix> ..
  make install
```

The stack of tasks
Installation

Through github:

- git clone --recursive git://github.com/jrl-umi3218/jrl-mal.git
- git clone --recursive git://github.com/jrl-umi3218/jrl-mathtools.git
- git clone --recursive git://github.com/laas/abstract-robot-dynamics.git
- git clone --recursive git://github.com/jrl-umi3218/jrl-dynamics.git
- git clone --recursive git://github.com/jrl-umi3218/jrl-walkgen.git
- git clone --recursive git://github.com/jrl-umi3218/dynamic-graph.git
- git clone --recursive git://github.com/jrl-umi3218/dynamic-graph-python.git
- git clone --recursive git://github.com/jrl-umi3218/sot-core.git
- git clone --recursive git://github.com/laas/sot-tools.git
- git clone --recursive git://github.com/jrl-umi3218/sot-dynamic.git
- git clone --recursive git://github.com/jrl-umi3218/sot-pattern-generator.git
- git clone --recursive git://github.com/stack-of-tasks/sot-application.git
- git clone --recursive git://github.com/laas/sot-hrp2.git
- git clone --recursive git://github.com/stack-of-tasks/sot-hrprtc-hrp2.git

- for each package,
  mkdir package/build
  cd package/build
  cmake -DCMAKE_INSTALL_PREFIX=<your_prefix> ..

  make install
Installation

Through installation script

```
▶ git clone git://github.com/stack-of-tasks/install-sot.git
   cd install-sot/scripts

   ./install_sot.sh
```
Running the stack of tasks into OpenHRP-3.1

You need to install:

- ros-electric
- OpenHRP-3.1

You will find instructions in [https://wiki.laas.fr/robots/HRP/Software](https://wiki.laas.fr/robots/HRP/Software)

Then follow instructions in [sot-hrprtc/README.md](https://github.com/stack-of-tasks/sot-hrprtc-hrp2)
Running the stack of tasks into OpenHRP-3.0.7

Assumptions

- OpenHRP 3.0.7 is installed
- The Stack of Tasks has been installed thanks to previous slide with `install_sot.sh` in the directory:

  /home/user/devel/ros_unstable

- Your `/opt/grx3.0/HRP2LAAS/bin/config.sh` is well setup.

The golden commands

```
$ roscore
  # Launching HRP2 simulation with OpenHRP
$ rosrun hrp2 bringup openhrp_bridge.launch robot:=hrp2_14
    mode:=dg_with_stabilizer simulation:=true
$ rosservice call /start_dynamic_graph
$ rosrun dynamic_graph_bridge run_command
```
Running the stack of tasks into OpenHRP-3.0.7

Initialize the application: create tracer and solver

```python
[INFO] [WallTime: 1370854858.786392] waiting for service...
Interacting with remote server.
>>> from dynamic_graph.sot.application.velocity.precomputed_tasks import initialize
>>> solver = initialize(robot)
>>> robot.initializeTracer()
```
Running the stack of tasks into OpenHRP-3.0.7

Build the graph including the pattern generator

```
[INFO] [WallTime: 1370854858.786392] waiting for service...
Interacting with remote server.
>>> from dynamic_graph.sot.pattern_generator.walking
   import CreateEverythingForPG, walkFewSteps
With meta selector
```
Running the stack of tasks into OpenHRP-3.0.7

Create the graph

```plaintext
>>> CreateEverythingForPG(robot, solver)
At this stage
('modelDir': '',
   '/home/jaca/ros-unstable/install/share/hrp2-14')
('modelName': 'HRP2JRLmainsmall.wrl')
('specificitiesPath': '',
   'HRP2SpecificitiesSmall.xml')
('jointRankPath': '', 'HRP2LinkJointRankSmall.xml')
After Task for Right and Left Feet
```
Running the stack of tasks into OpenHRP-3.0.7

Switch to the new graph

```python
>>> walkFewSteps(robot)
```
Software structure - Conceptual view

- Robot
- Dyn
- Feature
- WPG
- Desired Feature
- Solver
- Task
- Python
- ROS

Legend:
- SoT Entity
- C++ server
- Process/Task

The stack of tasks
Software structure - Link with Model

The stack of tasks
Software structure - Link with Model

The stack of tasks
Software structure - Link with Model

- Robot
- Dyn
- Feature
- Task
- WPG
- Desired Feature

\[ M(q) \]

\[ M^*(q) \]
Software structure - Link with Model

\[ T(q, t) = \begin{pmatrix} t(M^{-1}(t)M(q)) \\ u_\theta(R^{-1}(t)R(q)) \end{pmatrix} \]

\[ J = \frac{\partial T}{\partial q} \]

The stack of tasks
Software structure - Link with Model

The stack of tasks
Software structure - Link with Model

\[ T(q, t) = \begin{pmatrix} t(M^{-1}(t)M(q)) \\ u_\theta(R^{-1}(t)R(q)) \end{pmatrix} \]

\[ J = \frac{\partial T}{\partial q} \]

\[ \dot{q} \triangleq -J^+ (\lambda T + \frac{\partial T}{\partial t}) \]

\[ \dot{T} = -\lambda T - \frac{\partial T}{\partial t} \]

\[ M^*(q) \]

The stack of tasks
Software structure - Repositories

The stack of tasks