The stack of tasks

Florent Lamiraux, Olivier Stasse and Nicolas Mansard

CNRS-LAAS, Toulouse, France

Journex robotex, 3 juillet 2013

The stack of tasks

Outline

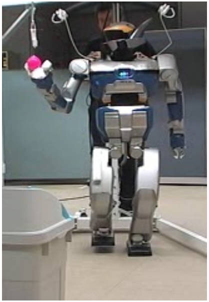
Introduction

A few words about the theory

Software

Introduction

Sensor-based closed loop control of a redundant robot



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 (in simulation).

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 (in simulation).

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 (in simulation).

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 (in simulation).

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 (in simulation).

Outline

Introduction

A few words about the theory

Software

Configuration space

- Robot: set of rigid-bodies linked by joints $\mathcal{B}_0, \cdots \mathcal{B}_m$.
- Configuration: position in space of each body.

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

- Position of $\mathcal{B}_i$ depends on $\mathbf{q}$:

$$M_{\mathcal{B}_i}(\mathbf{q}) \in SE(3)$$

Configuration space

- Robot: set of rigid-bodies linked by joints $\mathcal{B}_0, \cdots \mathcal{B}_m$.
- Configuration: position in space of each body.

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

- Position of $\mathcal{B}_i$ depends on $\mathbf{q}$:

$$M_{\mathcal{B}_i}(\mathbf{q}) \in SE(3)$$

Configuration space

- Robot: set of rigid-bodies linked by joints $\mathcal{B}_0, \cdots \mathcal{B}_m$.
- Configuration: position in space of each body.

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

- Position of $\mathcal{B}_i$ depends on $\mathbf{q}$:

$$M_{\mathcal{B}_i}(\mathbf{q}) \in SE(3)$$

Velocity

- Velocity:

$$\begin{array}{rcl} \dot{\mathbf{q}} & = & (\dot{x}, \dot{y}, \dot{z}, \omega_x, \omega_y, \omega_z, \dot{\theta}_1, \cdots \dot{\theta}_{n-6}) \\ \omega & \in & \mathbb{R}^3 \end{array}$$

- Velocity of $\mathcal{B}_i$

$$\begin{pmatrix} \mathbf{v}_{\mathcal{B}_i} \\ \omega_{\mathcal{B}_i} \end{pmatrix} (\mathbf{q}, \dot{\mathbf{q}}) = J_{\mathcal{B}_i}(\mathbf{q}).\dot{\mathbf{q}} \in \mathbb{R}^6$$

Velocity

- Velocity:

$$\begin{aligned} \dot{\mathbf{q}} &= (\dot{x}, \dot{y}, \dot{z}, \omega_x, \omega_y, \omega_z, \dot{\theta}_1, \cdots \dot{\theta}_{n-6}) \\ \omega &\in \mathbb{R}^3 \end{aligned}$$

- Velocity of $\mathcal{B}_i$

$$\begin{pmatrix} \mathbf{v}_{\mathcal{B}_i} \\ \omega_{\mathcal{B}_i} \end{pmatrix} (\mathbf{q}, \dot{\mathbf{q}}) = J_{\mathcal{B}_i}(\mathbf{q}).\dot{\mathbf{q}} \in \mathbb{R}^6$$

Velocity

- Velocity:

$$\begin{aligned} \dot{\mathbf{q}} &= (\dot{x}, \dot{y}, \dot{z}, \omega_x, \omega_y, \omega_z, \dot{\theta}_1, \cdots \dot{\theta}_{n-6}) \\ \omega &\in \mathbb{R}^3 \end{aligned}$$

- Velocity of $\mathcal{B}_i$

$$\begin{pmatrix} \mathbf{v}_{\mathcal{B}_i} \\ \omega_{\mathcal{B}_i} \end{pmatrix} (\mathbf{q}, \dot{\mathbf{q}}) = J_{\mathcal{B}_i}(\mathbf{q}).\dot{\mathbf{q}} \in \mathbb{R}^6$$

Task

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

  that should converge to 0:

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

- Examples
  - trajectory tracking of an end-effector $\mathcal{B}_{left-hand}$
  - trajectory tracking of the center of mass

Task

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

  that should converge to 0:

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

- Examples
  - trajectory tracking of an end-effector $\mathcal{B}_{left-hand}$
  - trajectory tracking of the center of mass

Task

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

  that should converge to 0:

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

- Examples
  - trajectory tracking of an end-effector $\mathcal{B}_{left-hand}$
  - trajectory tracking of the center of mass

Task

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

  that should converge to 0:

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

- Examples
  - trajectory tracking of an end-effector $\mathcal{B}_{left-hand}$
  - trajectory tracking of the center of mass

Task based control

Given

- a configuration $\mathbf{q}$,
- a task:
  - $T \in C^{\infty}(\mathcal{C} \times \mathbb{R}, \mathbb{R}^m)$,

compute a control vector $\dot{\mathbf{q}}$

- that makes T converge toward 0 and

Task based control

Given

- a configuration $\mathbf{q}$,
- a task:
  - $T \in C^{\infty}(\mathcal{C} \times \mathbb{R}, \mathbb{R}^m)$,

compute a control vector $\dot{\mathbf{q}}$

- that makes T converge toward 0 and

Task based control

Jacobian:

- we denote
  - $J = \frac{\partial T}{\partial \mathbf{q}}$
- then
  - $\dot{T} = J(\mathbf{q}, t)\dot{\mathbf{q}} + \frac{\partial T}{\partial t}(\mathbf{q}, t)$

We try to enforce

- $\dot{T} = -\lambda T \quad \Rightarrow \quad T(t) = e^{-\lambda t} T(0) \rightarrow 0$
- λ is called the gain associated to T.

Task based control

Jacobian:

- we denote
  - $J = \frac{\partial T}{\partial \mathbf{q}}$
- then
  - $\dot{T} = J(\mathbf{q}, t)\dot{\mathbf{q}} + \frac{\partial T}{\partial t}(\mathbf{q}, t)$

We try to enforce

- $\dot{T} = -\lambda T \quad \Rightarrow \quad T(t) = e^{-\lambda t} T(0) \rightarrow 0$
- λ is called the gain associated to T.

Task based control

Jacobian:

- we denote
  - $J = \frac{\partial T}{\partial \mathbf{q}}$
- then
  - $\dot{T} = J(\mathbf{q}, t)\dot{\mathbf{q}} + \frac{\partial T}{\partial t}(\mathbf{q}, t)$

We try to enforce

- $\dot{T} = -\lambda T \quad \Rightarrow \quad T(t) = e^{-\lambda t} T(0) \rightarrow 0$
- λ is called the gain associated to T.

Task based control

Jacobian:

- we denote
  - $J = \frac{\partial T}{\partial \mathbf{q}}$
- then
  - $\dot{T} = J(\mathbf{q}, t)\dot{\mathbf{q}} + \frac{\partial T}{\partial t}(\mathbf{q}, t)$

We try to enforce

- $\dot{T} = -\lambda T \quad \Rightarrow \quad T(t) = e^{-\lambda t} T(0) \rightarrow 0$
- λ is called the gain associated to T.

Task based control

Resolution of the constraint:

$$\dot{T} = J\dot{\mathbf{q}} \quad + \quad \frac{\partial T}{\partial t} = -\lambda T \tag{1}$$

$$J\dot{\mathbf{q}} \quad = \quad -\lambda T - \frac{\partial T}{\partial t} \tag{2}$$

$$\dot{\mathbf{q}} \quad \triangleq \quad -J^{+}(\lambda T + \frac{\partial T}{\partial t}) \tag{3}$$

Where J^{+} is the (Moore Penrose) pseudo-inverse of J.

Task based control

Resolution of the constraint:

$$\dot{T} = J\dot{\mathbf{q}} + \frac{\partial T}{\partial t} = -\lambda T \tag{1}$$

$$J\dot{\mathbf{q}} = -\lambda T - \frac{\partial T}{\partial t} \tag{2}$$

$$\dot{\mathbf{q}} \triangleq -J^{+}(\lambda T + \frac{\partial T}{\partial t}) \tag{3}$$

Where J^{+} is the (Moore Penrose) pseudo-inverse of J.

Hierachical task based control

- Usually, the task requires less dof than available.
- Other tasks can be controlled without affecting $\dot{T}$
- Hence hierarchical task based control

Hierachical task based control

- Usually, the task requires less dof than available.
- Other tasks can be controlled without affecting $\dot{T}$
- Hence hierarchical task based control

Hierachical task based control

- Usually, the task requires less dof than available.
- Other tasks can be controlled without affecting $\dot{T}$
- Hence hierarchical task based control

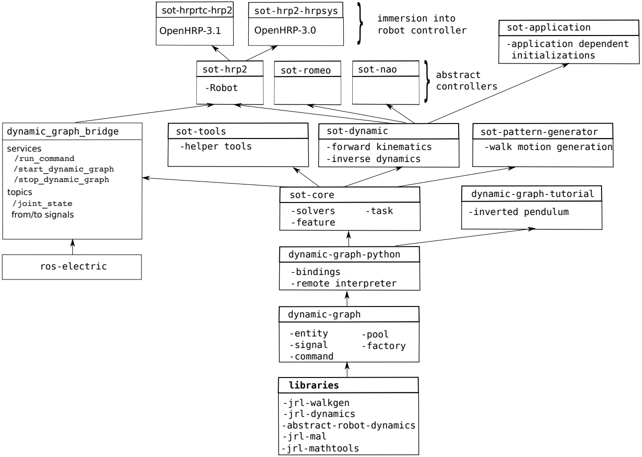
Outline

Introduction

A few words about the theory

Software

Architecture overview



Main features

- 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.

Main features

- 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.

Main features

- 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

Synchronous interface storing a given data type

- output signals:
  - recomputed by a callback function
- input signals:
  - plugged by an output signal

Signal

Synchronous interface storing a given data type

- output signals:
  - recomputed by a callback function
- input signals:
  - plugged by an output signal

Command

Asynchronous interface

- typed input
- trigger an action,
- returns a typed result

Main features

Python bindings

- module `dynamic_graph` linked to C++ compiled code
  - 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
  - class Signal
    - value setter and getter

Main features

Python bindings

- module `dynamic_graph` linked to C++ compiled code
  - 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
  - class Signal
    - value setter and getter

Main features

Python bindings

- module `dynamic_graph` linked to C++ compiled code
  - 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
  - class Signal
    - value setter and getter

Main features

Python bindings

- module `dynamic_graph` linked to C++ compiled code
  - 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
  - class Signal
    - value setter and getter

Main features

Hierarchical task solver

- computes robot joint velocity

Extensions

- `sot-dynamic`: forward kinematics
  - position and Jacobian of end effectors (wrists, ankles),
  - position of center of mass
- `sot-pattern-generator`: walk motion for legged robots,
- `sot-dyninv`: inverse dynamics based control.

Extensions

- `sot-dynamic`: forward kinematics
  - position and Jacobian of end effectors (wrists, ankles),
  - position of center of mass
- `sot-pattern-generator`: walk motion for legged robots,
- `sot-dyninv`: inverse dynamics based control.

Extensions

- `sot-dynamic`: forward kinematics
  - position and Jacobian of end effectors (wrists, ankles),
  - position of center of mass
- `sot-pattern-generator`: walk motion for legged robots,
- `sot-dyninv`: inverse dynamics based control.