How to reconcile path planning and visual servoing through manipulation tasks

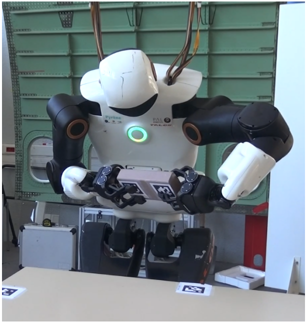
Florent Lamiraux

CNRS-LAAS, Toulouse, France

Position of the problem

We want to robustly perform manipulation tasks with a robot

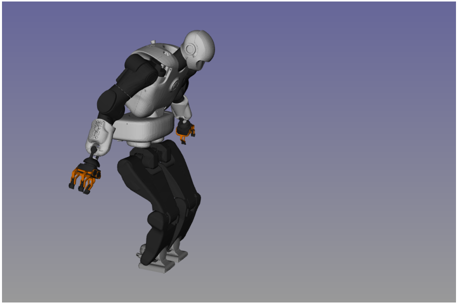
- with many degrees-of-freedom,
- moving in a cluttered environment,
- subject to kinematic and dynamic contraints.



Position of the problem

We have models of

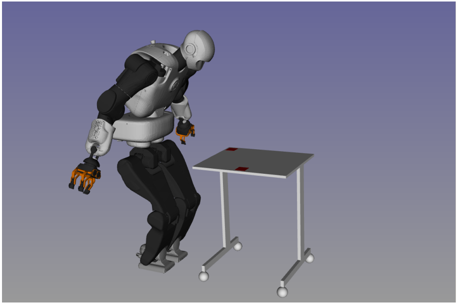
- the robot,
-
-



Position of the problem

We have models of

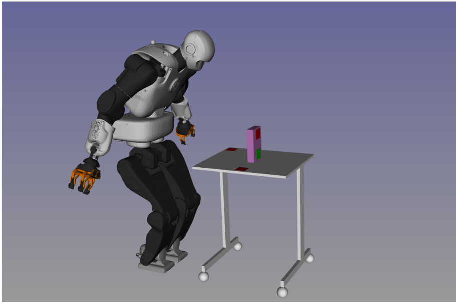
- the robot,
- the environment,
-



Position of the problem

We have models of

- the robot,
- the environment,
- the objects.



Position of the problem

We want to robustly perform manipulation tasks with a robot

- with many degrees-of-freedom,
- moving in a cluttered environment,
- subject to kinematic and dynamic contraints.

-
-

Position of the problem

We want to robustly perform manipulation tasks with a robot

- with many degrees-of-freedom,
- moving in a cluttered environment,
- subject to kinematic and dynamic contraints.

- Visual servoing,
-

Position of the problem

We want to robustly perform manipulation tasks with a robot

- with many degrees-of-freedom,
- moving in a cluttered environment,
- subject to kinematic and dynamic contraints.

- Visual servoing,
- motion planning (random sampling)

Position of the problem

We want to robustly perform manipulation tasks with a robot

- with many degrees-of-freedom,
- moving in a cluttered environment,
- subject to kinematic and dynamic contraints.

- Visual servoing,
- motion planning (random sampling)

Little work combine motion planning and visual servoing.

Contribution

- An original methodology to integrate visual servoing and motion planning together.
- going beyond previous work :
  - Y. Mezouar and F. Chaumette, "Path planning for robust image-based control," IEEE TRO, 2002

Contribution

- An original methodology to integrate visual servoing and motion planning together.
- going beyond previous work :
  - Y. Mezouar and F. Chaumette, “Path planning for robust image-based control,” IEEE TRO, 2002

Manipulation planning

Motion control

Implementation

Definitions

A manipulation motion is a motion

- implying
  - one or several robots
  - one or several objects
- in such a way that each object
  - either is in a stable pose,
  - or is moved by a robot.

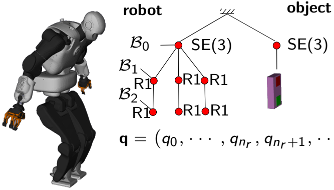
Definitions

A manipulation motion is a motion

- implying
  - one or several robots
  - one or several objects
- in such a way that each object
  - either is in a stable pose,
  - or is moved by a robot.

Composite robot

Kinematic chain composed of each robot and each object



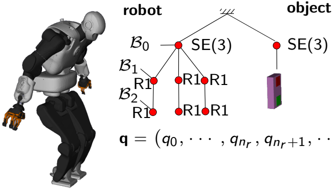
$$\mathbf{q} = (q_0, \cdots, q_{n_r}, q_{n_r+1}, \cdots, q_{n_r+7})$$

The configuration space of the composite robot is the Cartesian product of the configuration spaces of the robots and objects.

$$\mathcal{C} = \mathcal{C}_{r1} \times \mathcal{C}_{r_{nb\ robots}} \times SE(3)^{nb\ objets}$$

Composite robot

Kinematic chain composed of each robot and each object



$$\mathbf{q} = (q_0, \cdots, q_{n_r}, q_{n_r+1}, \cdots, q_{n_r+7})$$

The configuration space of the composite robot is the Cartesian product of the configuration spaces of the robots and objects.

$$\mathcal{C} = \mathcal{C}_{r1} \times \mathcal{C}_{r_{nb\ \ robots}} \times SE(3)^{nb\ \ objets}$$

Numerical constraints

Manipulation constraints can be expressed numerically.

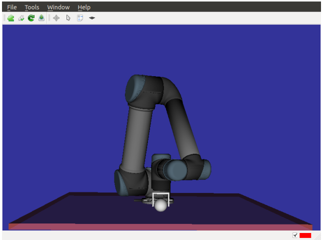
- Numerical constraints :

$$f(\mathbf{q}) = 0, \qquad \begin{array}{l} m \in \mathbb{N}, \\ f \in C^1(\mathcal{C}, \mathbb{R}^m) \end{array}$$

- Parameterizable numerical constraints :

$$f(\mathbf{q}) = f_0, \qquad \begin{array}{l} m \in \mathbb{N}, \\ f \in C^1(\mathcal{C}, \mathbb{R}^m) \\ f_0 \in \mathbb{R}^m \end{array}$$

Example : a robot manipulating a ball



$$\mathcal{C} = [-\pi, \pi]^6 \times SE(3) \quad (1)$$

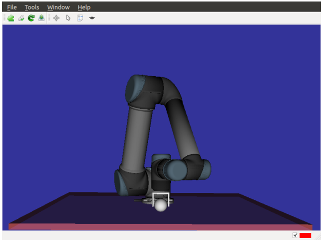
$$\mathbf{q} = (q_0, q_1, q_2, q_3, q_4, q_5, \quad (2)$$

$$x_b, y_b, z_b, X_b, Y_b, Z_b, W_b) \quad (3)$$

Two *states* :

- `placement` : the ball lies on the table,
- `grasp` : the ball is grasped by the gripper.

Example : a robot manipulating a ball



Each state is defined by a numerical constraint :

- `placement`

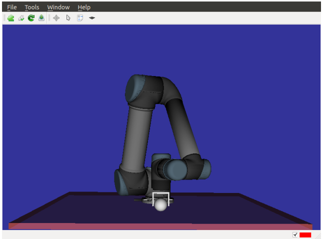
$$z_b = 0$$

- `grasp`

$$\mathbf{x}_{gripper}(q_0, \cdots, q_5) - \begin{pmatrix} x_b \\ y_b \\ z_b \end{pmatrix} = 0$$

Each state is a sub-manifold of the composite configuration space.

Example : a robot manipulating a ball



Each state is defined by a numerical constraint :

- `placement`

$$z_b = 0$$

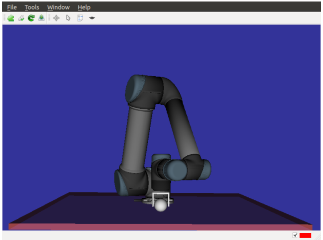
- `grasp`

$$\mathbf{x}_{gripper}(q_0, \cdots, q_5) - \begin{pmatrix} x_b \\ y_b \\ z_b \end{pmatrix} = 0$$

Each state is a sub-manifold of the composite configuration space.

Example : a robot manipulating a ball

Numerical motion constraints

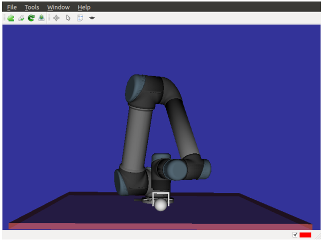


Two *types of motions* :

- `transit` : the ball is lying **still** on the table,
- `transfer` : the balle moves with the gripper.

Example : a robot manipulating a ball

Numerical motion constraints



- `transit`

$$\left.\begin{array}{rcl} x_b & = & x_0 \\ y_b & = & y_0 \end{array}\right\} \text{ parameterizable}$$

$$\left. z_b = 0 \right\} \text{ simple}$$

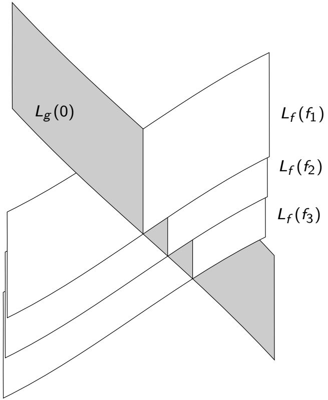
- `transfer`

$$\mathbf{x}_{gripper}(q_0, \cdots, q_5) - \begin{pmatrix} x_b \\ y_b \\ z_b \end{pmatrix} = 0$$

$$R_{gripper}^{-1} R_b = R_0$$

Foliation

Motion constraints define foliations of the admissible configuration space (`grasp` ∪ `placement`).



- f : position of the ball

$$L_f(f_1) = \{\mathbf{q} \in \mathcal{C}, f(\mathbf{q}) = f_1\}$$

- g : grasp of the ball

$$L_g(0) = \{\mathbf{q} \in \mathcal{C}, g(\mathbf{q}) = 0\}$$

General case

In a manipulation planning problem,

- the state of the system is subject to
  - numerical constraints,
- the trajectories of the system are subject to
  - parameterizable numerical constraints, the dimension of the parameter may be smaller than the dimension of the constraints,
  - the parameter value is constant along trajectories

General case

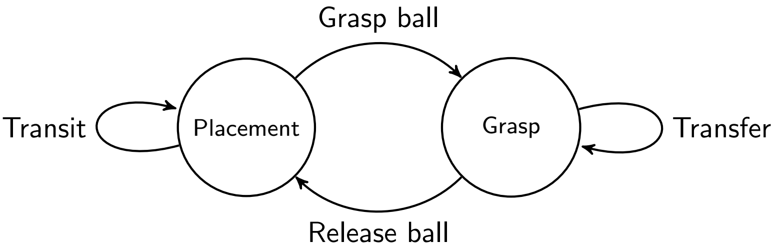
In a manipulation planning problem,

- the state of the system is subject to
  - numerical constraints,
- the trajectories of the system are subject to
  - parameterizable numerical constraints, the dimension of the parameter may be smaller than the dimension of the constraints,
  - the parameter value is constant along trajectories

Constraint graph

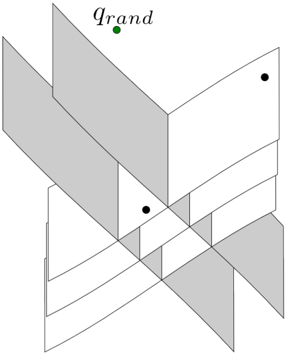
A manipulation problem can be represented by a *manipulation graph*.

- **The nodes** or *states* contain numerical constraints.
- **The edges** or *transitions* contain parameterizable constraints.



Algorithms

Manipulation RRT



Manipulation RRT

$q_{rand} = \text{shoot\_random\_config}()$

$q_{near} = \text{nearest\_neighbour}(q_{rand}, tree)$

$f_e = \text{select\_next\_state}(q_{near})$

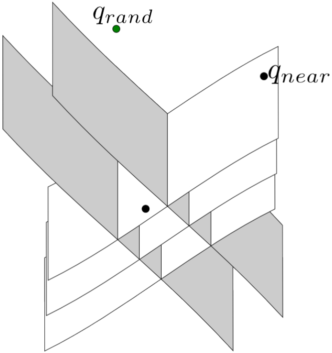
$q_{proj} = \text{project}(q_{rand}, f_e)$

$q_{new} = \text{extend}(q_{near}, q_{proj})$

$tree.\text{insert\_node}(q_{near}, q_{new})$

Algorithms

Manipulation RRT



Manipulation RRT

$q_{rand} = \text{shoot\_random\_config}()$

$q_{near} = \text{nearest\_neighbour}(q_{rand}, \textit{tree})$

$f_e = \text{select\_next\_state}(q_{near})$

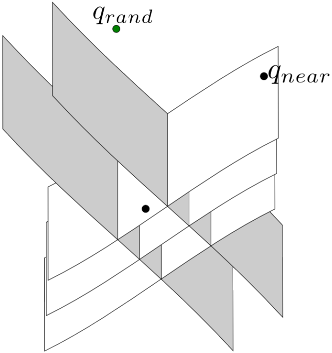
$q_{proj} = \text{project}(q_{rand}, f_e)$

$q_{new} = \text{extend}(q_{near}, q_{proj})$

$\textit{tree}.\text{insert\_node}(q_{near}, q_{new})$

Algorithms

Manipulation RRT



Manipulation RRT

q_{rand} = shoot_random_config()

q_{near} = nearest_neighbour(q_{rand}, *tree*)

f_e = select_next_state(q_{near})

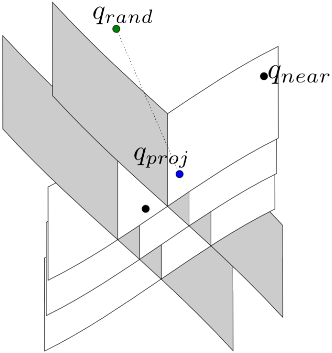
q_{proj} = project(q_{rand}, f_e)

q_{new} = extend(q_{near}, q_{proj})

tree.insert_node(q_{near}, q_{new})

Algorithms

Manipulation RRT



Manipulation RRT

$q_{rand} = \text{shoot\_random\_config}()$

$q_{near} = \text{nearest\_neighbour}(q_{rand}, tree)$

$f_e = \text{select\_next\_state}(q_{near})$

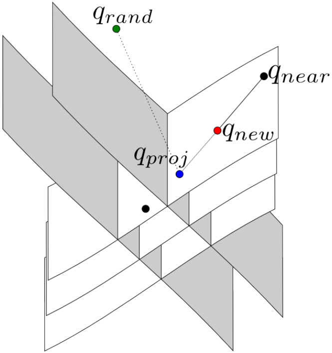
$q_{proj} = \text{project}(q_{rand}, f_e)$

$q_{new} = \text{extend}(q_{near}, q_{proj})$

$tree.\text{insert\_node}(q_{near}, q_{new})$

Algorithms

Manipulation RRT



Manipulation RRT

q_{rand} = shoot_random_config()

q_{near} = nearest_neighbour(q_{rand}, $tree$)

f_e = select_next_state(q_{near})

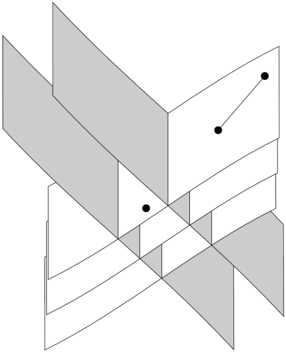
q_{proj} = project(q_{rand}, f_e)

q_{new} = extend(q_{near}, q_{proj})

$tree$.insert_node(q_{near}, q_{new})

Algorithms

Manipulation RRT



Manipulation RRT

$q_{rand} = \text{shoot\_random\_config}()$

$q_{near} = \text{nearest\_neighbour}(q_{rand}, tree)$

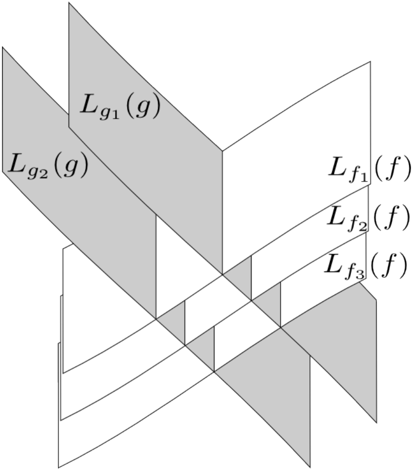
$f_e = \text{select\_next\_state}(q_{near})$

$q_{proj} = \text{project}(q_{rand}, f_e)$

$q_{new} = \text{extend}(q_{near}, q_{proj})$

$tree.\text{insert\_node}(q_{near}, q_{new})$

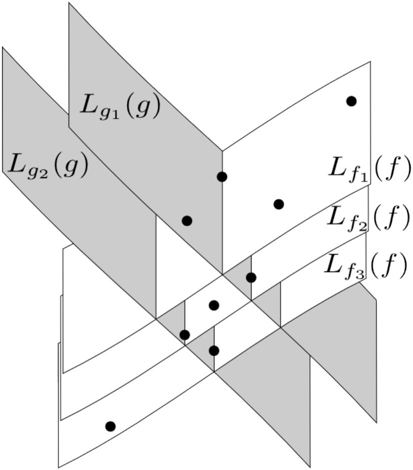
Constraint graph and configuration space



2 constraints on motion

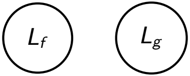
- f : position of the object.
- g : grasp of the object.

Constraint graph and configuration space

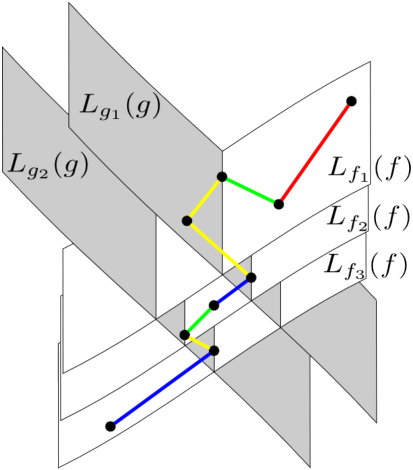


2 constraints on motion

- f : position of the object.
- g : grasp of the object.

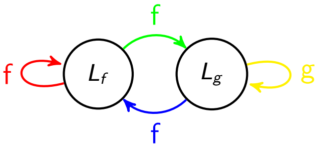


Constraint graph and configuration space

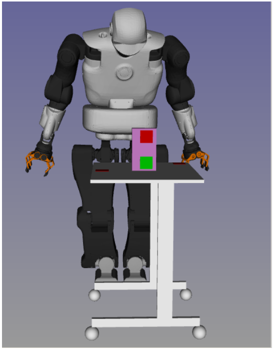


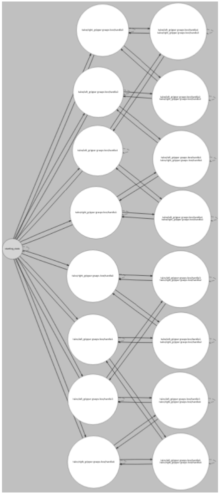
2 constraints on motion

- f : position of the object.
- g : grasp of the object.



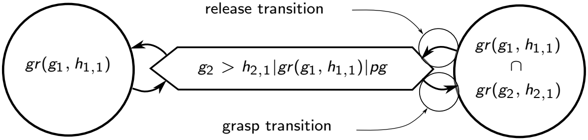
Example

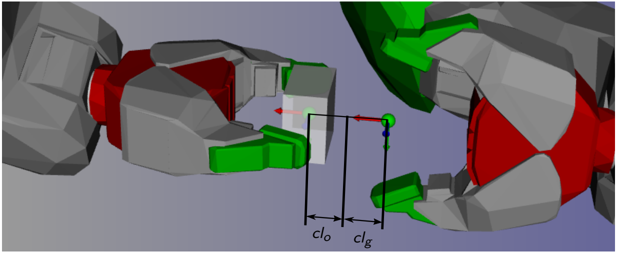




Waypoints states

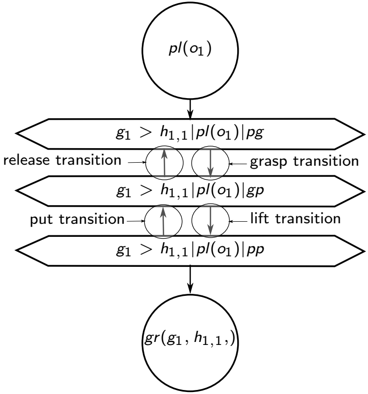
Intermediate states in the constraint graph

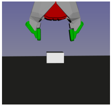


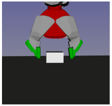


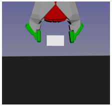
Waypoints states

Intermediate states in the constraint graph









Hierarchical task based controller

Task

- model : mapping T from $\mathcal{C}_{rob}$ to $\mathbb{T}$ (vector space or SE(3)),
- reference : smooth mapping T^* from $I \subset \mathbb{R}$ to $\mathbb{T}$, desired trajectory of the task,

Hierarchical task based controller

Task

- model : mapping T from $\mathcal{C}_{rob}$ to $\mathbb{T}$ (vector space or SE(3)),
- reference : smooth mapping T^* from $I \subset \mathbb{R}$ to $\mathbb{T}$, desired trajectory of the task,

Hierarchical task based controller

If the model of the task is perfect and $\mathbb{T} = \mathbb{R}^{n_1}$

$$\varepsilon(\mathbf{q}, t) \triangleq T(\mathbf{q}) - T^*(t)$$

$$\dot{\varepsilon} = \frac{\partial T}{\partial \mathbf{q}}\dot{\mathbf{q}} - \dot{T}^*$$

We wish to achieve $\dot{\varepsilon} = -\lambda\varepsilon$:

$$\dot{\mathbf{q}} = \frac{\partial T}{\partial \mathbf{q}}^{+}(\dot{T}^* - \lambda\varepsilon)$$

Hierarchical task based controller

If the model of the task is perfect and $\mathbb{T} = \mathbb{R}^{n_1}$

$$\varepsilon(\mathbf{q}, t) \triangleq T(\mathbf{q}) - T^*(t)$$

$$\dot{\varepsilon} = \frac{\partial T}{\partial \mathbf{q}} \dot{\mathbf{q}} - \dot{T}^*$$

We wish to achieve $\dot{\varepsilon} = -\lambda \varepsilon$:

$$\dot{\mathbf{q}} = \frac{\partial T}{\partial \mathbf{q}}^+ (\dot{T}^* - \lambda \varepsilon)$$

Hierarchical task based controller

If the model of the task is perfect and $\mathbb{T} = \mathbb{R}^{n_1}$

$$\varepsilon(\mathbf{q}, t) \triangleq T(\mathbf{q}) - T^*(t)$$

$$\dot{\varepsilon} = \frac{\partial T}{\partial \mathbf{q}}\dot{\mathbf{q}} - \dot{T}^*$$

We wish to achieve $\dot{\varepsilon} = -\lambda\varepsilon$:

$$\begin{aligned} \dot{\mathbf{q}} &= \frac{\partial T}{\partial \mathbf{q}}^{+}(\dot{T}^* - \lambda\varepsilon) + N\mathbf{u} \\ \mathbf{u} &\in \mathbb{R}^{n_2} \quad n_2 < n_1 \end{aligned}$$

Hierarchical task based controller

If the model of the task is perfect and $\mathbb{T} = \mathbb{R}^{n_1}$

$$\varepsilon(\mathbf{q}, t) \triangleq T(\mathbf{q}) - T^*(t)$$

If $\hat{T}$ is a measure of T through sensors,

$$\varepsilon(t) \triangleq \hat{T}(t) - T^*(t)$$

$$\dot{\mathbf{q}} = \frac{\partial T}{\partial \mathbf{q}}^{+}(\dot{T}^* - \lambda\varepsilon) + N\mathbf{u}$$

Hierarchical task based controller

If the model of the task is perfect and $\mathbb{T} = \mathbb{R}^{n_1}$

$$\varepsilon(\mathbf{q}, t) \triangleq T(\mathbf{q}) - T^*(t)$$

If $\hat{T}$ is a measure of T through sensors,

$$\varepsilon(t) \triangleq \hat{T}(t) - T^*(t)$$

$$\dot{\mathbf{q}} = \frac{\partial T}{\partial \mathbf{q}}^+ (\dot{T}^* - \lambda\varepsilon) + N\mathbf{u}$$

Hierarchical task based controller

If the model of the task is perfect and $\mathbb{T} = \mathbb{R}^{n_1}$

$$\varepsilon(\mathbf{q}, t) \triangleq T(\mathbf{q}) - T^*(t)$$

If $\hat{T}$ is a measure of T through sensors,

$$\varepsilon(t) \triangleq \hat{T}(t) - T^*(t)$$

$$\dot{\mathbf{q}} = \frac{\partial T}{\partial \mathbf{q}}^+ (\dot{T}^* - \lambda\varepsilon) + N\mathbf{u}$$

Hierarchical task based controller

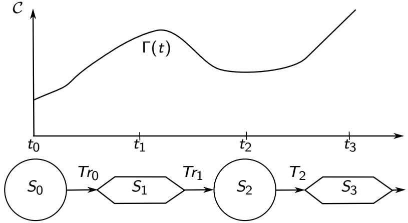
- Given tasks $T_1, T_2, \cdots$ in decreasing order of priority,
- a hierarchical task based controller iteratively computes $\dot{\mathbf{q}}$ in order to
  - achieve at best the decreasing ratio of the current task,
  - without affecting the decreasing ratio of the previous tasks.

Hierarchical task based controller

- Given tasks $T_1, T_2, \cdots$ in decreasing order of priority,
- a hierarchical task based controller iteratively computes $\dot{\mathbf{q}}$ in order to
  - achieve at best the decreasing ratio of the current task,
  - without affecting the decreasing ratio of the previous tasks.

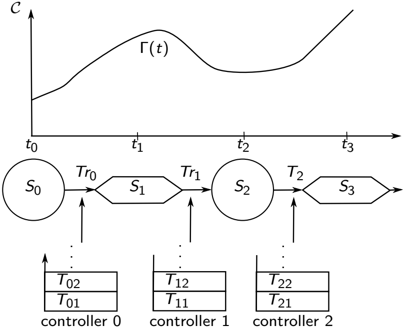
Output of manipulation planning

A sequence of paths along transitions linking states (including waypoints)



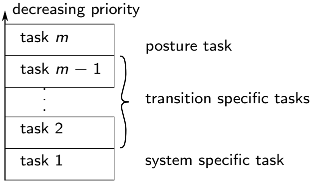
Output of manipulation planning

A sequence of paths along transitions linking states (including waypoints)



To each transition, we associate a hierarchical controller.

Hierarchical task based controller



- system specific task :
  - quasi-static equilibrium,
  - position of the base
- posture task : $\varepsilon = \mathbf{q} - \mathbf{q}^*(t)$
  - $\mathbf{q}^*(t)$: planned trajectory

Transition specific tasks

transition	**specific task**	$\mathbb{T}$
grasp or release	object/gripper	$SE(3)$
put or lift	object/contact surface	$SE(3)$
dual grap	gripper 1 / gripper 2	$SE(3)$

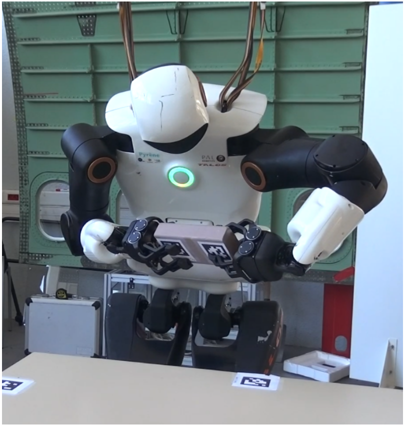
- $\hat{T}$: as much as possible, the above relative positions are measured through visual features,
- visibility of these features can be enforced through constraints at the manipulation planning step.

Manipulation planning

Motion control

Implementation

Humanoid robot manipulating a box



- Humanoid Path Planner
- Agimus
- Stack of Tasks
- ROS

Humanoid Path Planner

Open-source software platform for motion planning

- manipulation planning
    - Automatic construction of constraint graph from a set of grasps.
- acyclic contact planning for multiped robots (hpp-rbprm)

https://humanoid-path-planner.github.io/hpp-doc

Stack of Tasks

Open-source software platform for redundant robot motion control

- hierarchical task based controllers

https://stack-of-tasks.github.io

Work in progress

Implement drilling operations in a factory environment.

ROB4FAM

acknowledgements