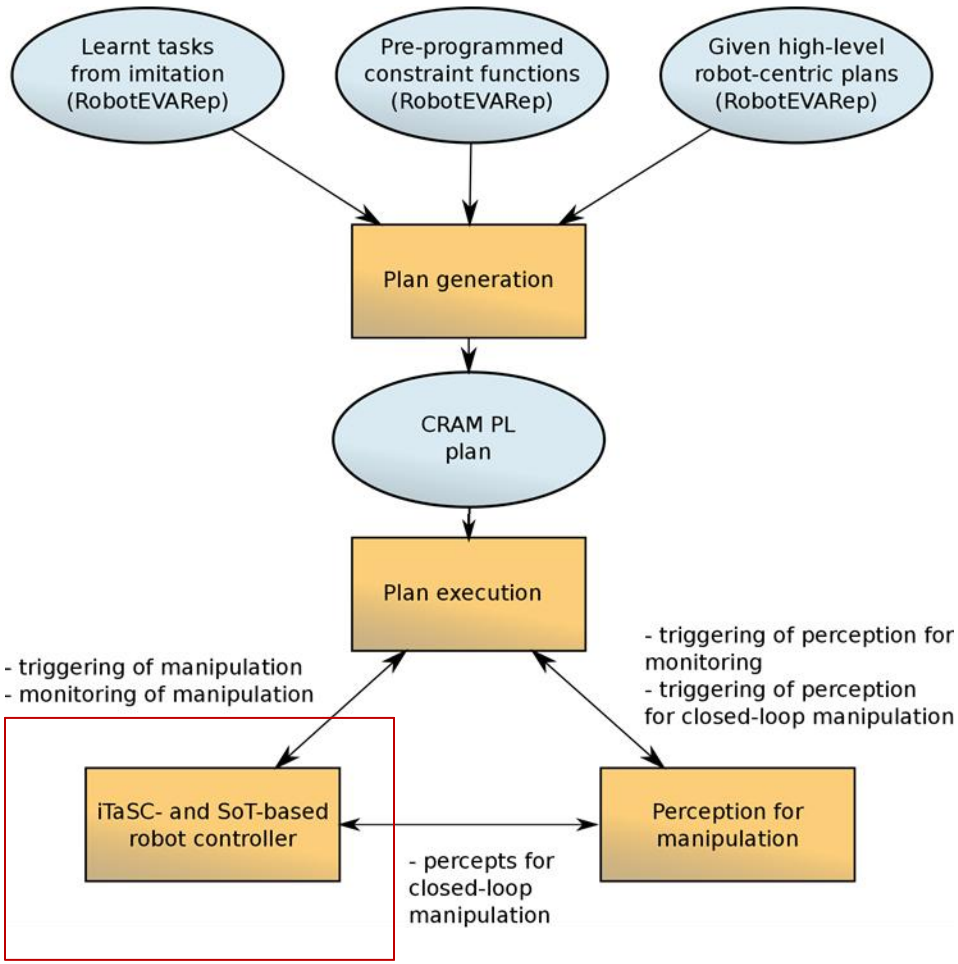
Stack of tasks tutorial

KEITH Francois
CNRS Lirmm

Position in robohow

- End of the process
- All symbolic data have been resolved and replaced by geometrical data.
- Task plan has been defined



Outline

- Theoretical background
- Structure of the software
- Hands in manipulation

Stack of Tasks

Simple task hierarchy

- Resolution of a problem of inverse kinematic Jq = e
  - A task space $\mathbf{e} = \mathbf{s} - \mathbf{s}^*$
    (error between current and desired sensor values)
  - A reference behavior of the error $\dot{\mathbf{e}}^* = -\lambda \mathbf{e}$
  - A Jacobian $\mathbf{J} = \frac{\partial \mathbf{e}}{\partial \mathbf{q}}$
- The problem we want to solve can be written as:
  - Minimization problem $\min_{\dot{\mathbf{q}}} \|\mathbf{J}\dot{\mathbf{q}} - \dot{\mathbf{e}}^*\|$
  - Pseudo inverse $\dot{\mathbf{q}} = \mathbf{J}^+ \dot{\mathbf{e}}^*$

Stack of Tasks

Simple task hierarchy

- Take advantage of the redundancy of the robot to realize several tasks simultaneously
- Task weighting (slacked hierarchy)

$$\min_{\dot{\mathbf{q}}} \left(\sum_{i=1}^{n} \left(\|\mathbf{J_i}\dot{\mathbf{q}} - \dot{\mathbf{e}}_\mathbf{i}^*\|^2 \alpha_i \right) \right)$$

- Strict hierarchy (Stack of tasks)

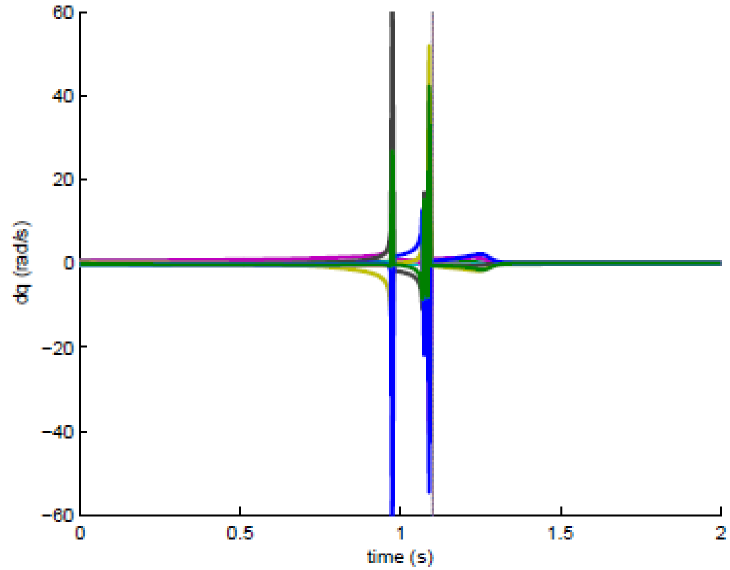
$$\alpha_i \ll \alpha_{i-1}$$

$$\dot{\mathbf{q}}_\mathbf{i} = \dot{\mathbf{q}}_\mathbf{i-1} + (\mathbf{J_i P_{i-1}})^+ (\dot{\mathbf{e}}_\mathbf{i}^* - \mathbf{J_i}\dot{\mathbf{q}}_\mathbf{i-1})$$

Stack of Tasks

Damping

- Pseudo-inverse approach not suited
  - Discontinuities near singularities
- Damped-inverse



$$\min_{\dot{\mathbf{q}}_\mathbf{i} \in S_i} \|\mathbf{J}_\mathbf{i}\dot{\mathbf{q}}_\mathbf{i} - \dot{\mathbf{e}}^*_\mathbf{i}\|^2 + \delta\|\dot{\mathbf{q}}_\mathbf{i}\|^2$$

$$\mathbf{M}^\dagger = (\mathbf{M} + \delta\mathbf{I})^+$$

\+ Continuous control law

\+ Prevent excessive values in the control when close to singularity.

\- Weakens the hierarchy between the tasks.

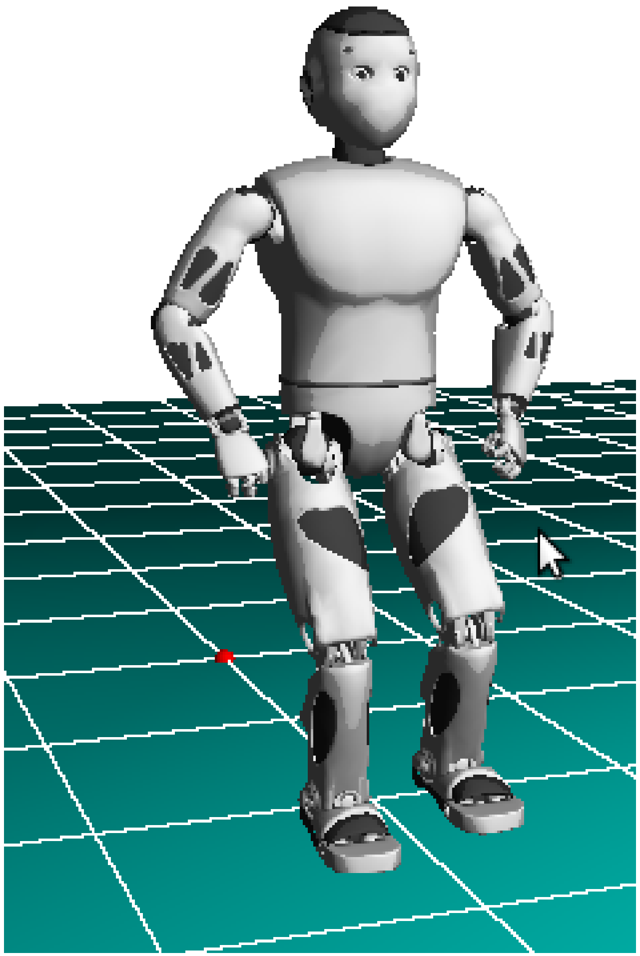
Stack of Tasks

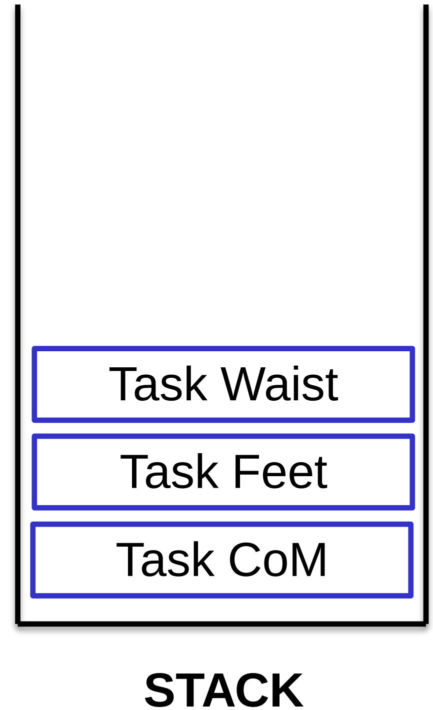
Inverse kinematic

- References $J_i \dot{q} = \dot{e}_i^*$
- Variables
  - Joint velocity $\dot{q}$
- Constraints
  - Joint limits $\begin{cases} \underline{\dot{q}} < \dot{q} < \overline{\dot{q}} \\ \underline{q} < q < \overline{q} \end{cases}$

Stack of Tasks

Example of kinematic simulation





Stack of Tasks

Hierarchical QP

- First stage

$$\min \| A_1 x - b_1 \|^2$$

$$\text{Let } w_1^* = A_1 x^* - b_1$$

- Second stage
  - If $w_1=0$

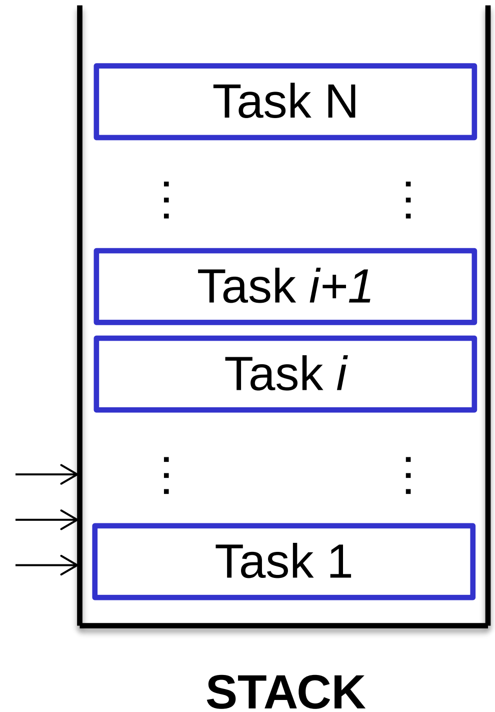
$$\min \| A_2 x - b_2 \|^2$$

$$\text{s.t.} \quad A_1 x = b_1$$

  - If $w_1 > 0$

$$\min \| A_2 x - b_2 \|^2$$

$$\text{s.t. } A_1 x = b_1 + w_1^*$$



Stack of Tasks

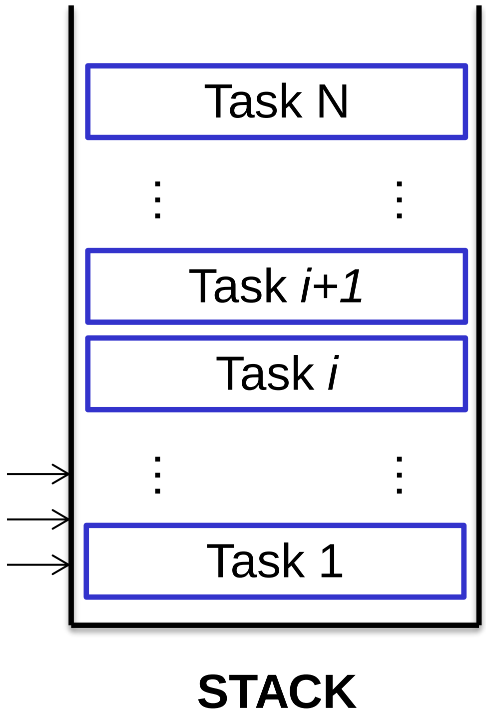
Hierarchical QP

- Rewriting with slack variables
- First stage

$$\begin{aligned} &\min \| w_1 \|^2 \\ &\text{s.t.} \quad A_1 x - b_1 = w_1 \end{aligned}$$

- Second stage

$$\begin{aligned} &\min \| w_2 \|^2 \\ &\text{s.t.} \quad A_1 x = b_1 + w_1^* \\ &\qquad\;\; A_2 x = b_2 + w_2 \end{aligned}$$



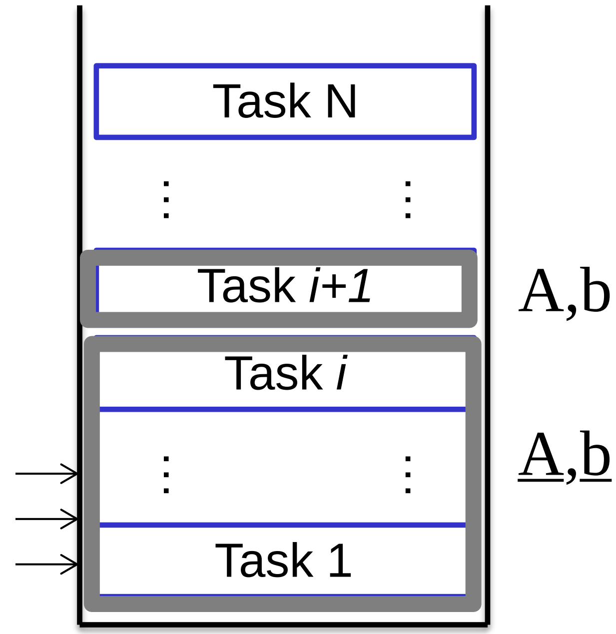
Stack of Tasks

Hierarchical QP

- Generic Formulation

Hierarchical QP

$$\min_{x,w} \| w \|^2$$
$$\text{s.t.} \quad A x - b = w$$
$$\underline{A}\, x - \underline{b} = \underline{w}^*$$



STACK

$$\underline{\bullet}_{k+1} = \begin{bmatrix} \bullet_{k+1} \\ \underline{\bullet}_{k} \end{bmatrix}$$

Least Square Equality (LSE)

Stack of Tasks

Hierarchical QP

- Generic Formulation

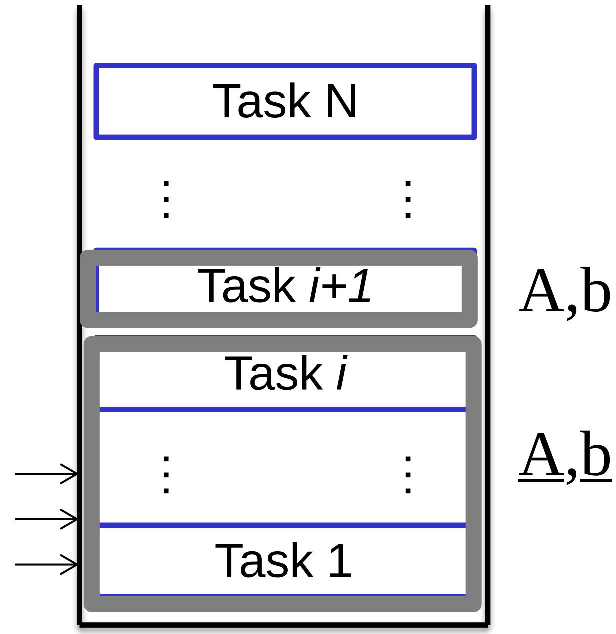
Hierarchical QP

$$\min_{x,w} \| w \|^2$$

$$\text{s.t.} \quad A\,x - b \leq w$$

$$\underline{A}\,x - \underline{b} \leq \underline{w}^*$$

$$\underline{\bullet}_{k+1} = \begin{bmatrix} \bullet_{k+1} \\ \underline{\bullet}_k \end{bmatrix}$$



STACK

Least Square Inequality (LSI)

Stack of Tasks

LSI Algorithm

- Primal Active search

$A := A_0$

$x, w := LSE(A)$

if $\exists\, i \notin A,\ \ w_i > 0$

$\quad A \mathrel{+}= i$

if $\exists\, i \in A,\ \ w_i < 0$

$\quad A \mathrel{-}= I$

return x

Decide an active set A

Solve the LSE reduced to A

If any unactive constraint is unsatisfied

Active the largest violation

If any Lagrange multiplier is negative

Unactive the smallest multiplier

The LSI optimum is the LSE optimum
for the *good* active set

Stack of Tasks

Example using the HCOD solver

KEITH Francois
CNRS Lirmm

robohow

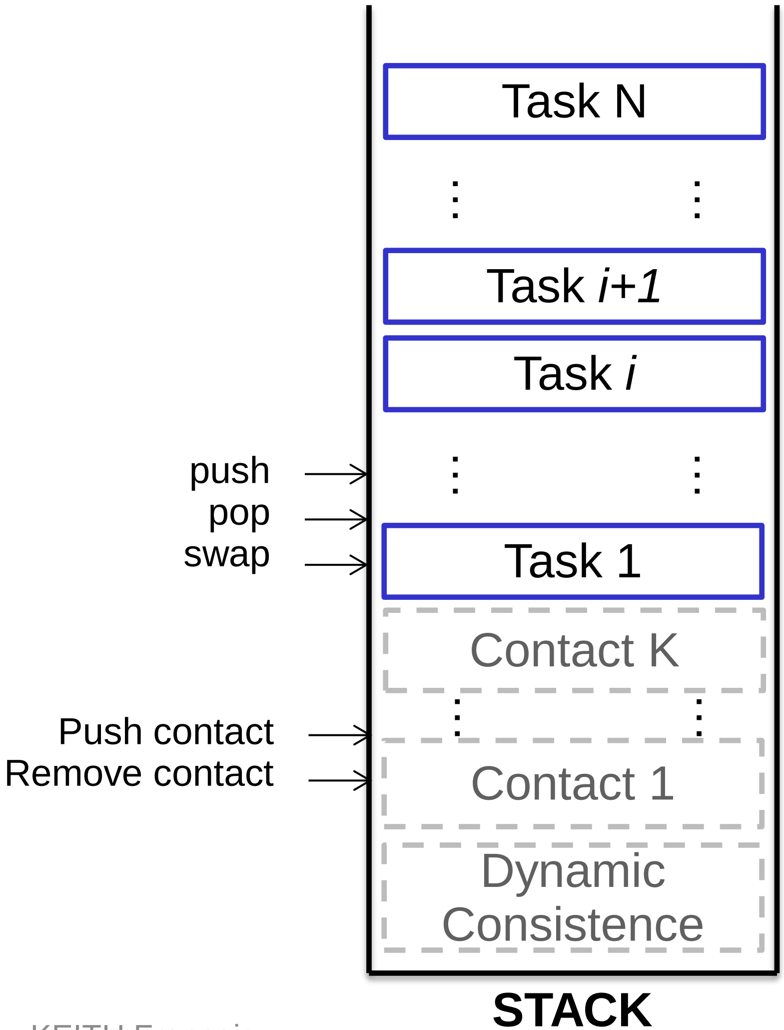
Stack of Tasks

Inverse dynamic

- References $J_i\ddot{q} + \dot{J}_i\dot{q} = \ddot{e}_i^*$
- Variables
  - Joint torques τ
  - Joint acceleration $\ddot{q}$
  - External forces f_c
- Constraints
  - Dynamic equation $M\ddot{q} + b(q,\dot{q}) = \tau + J_c^T f_c$
  - Contact constraint $\begin{cases} J_c\ddot{q} + \dot{J}_c\dot{q} = 0 \\ 0 \leq f_c^{\perp} \end{cases}$
  - Joint limits $\underline{\tau} < \tau < \overline{\tau}$

Stack of Tasks

Inverse dynamic



$$\ddot{e}_N + \mu_N = J_N \ddot{q}$$

$$\vdots$$

$$\ddot{e}_1 + \mu_1 = J_1 \ddot{q}$$

$$J_c^T \ddot{q} + \dot{J}_c \dot{q} = 0$$
$$f_\perp > 0$$

$$M\ddot{q} + b(q, \dot{q}) = \tau + J_c^T f_c$$

Stack of Tasks

Example of dynamic control

Presentation of the framework

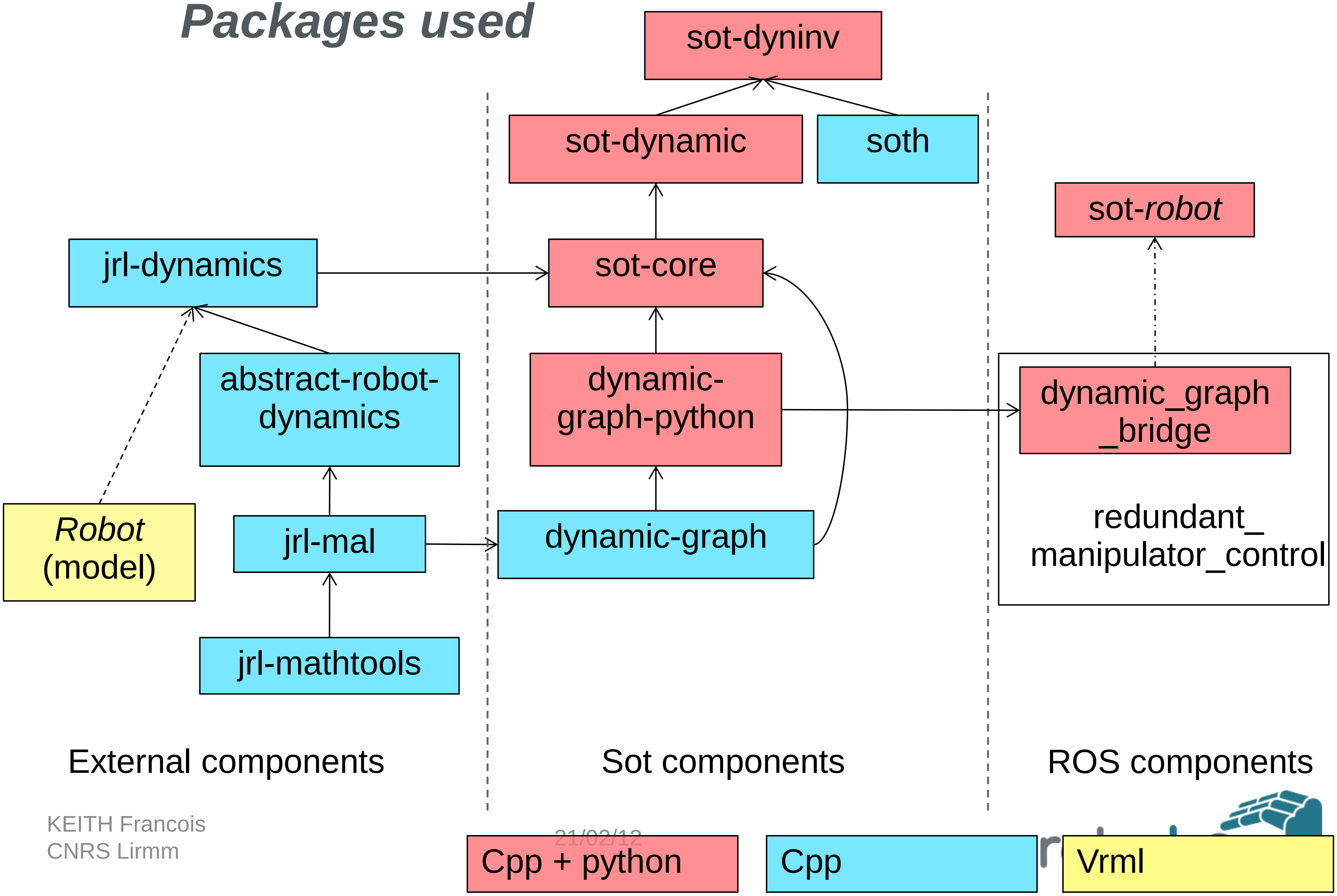
General description of the software

- Generic controler for robots: tested on humanoid robots (hrp2, romeo, nao) and wheeled robot (pr2)
- Prototyping + execution on the real robot
- C++ / python
- LGPL License

- Not a dynamic simulator:
  (It estimates the dynamic, but is not able to handle contact / impact with other moving objects).
- No high level (sequencer …)

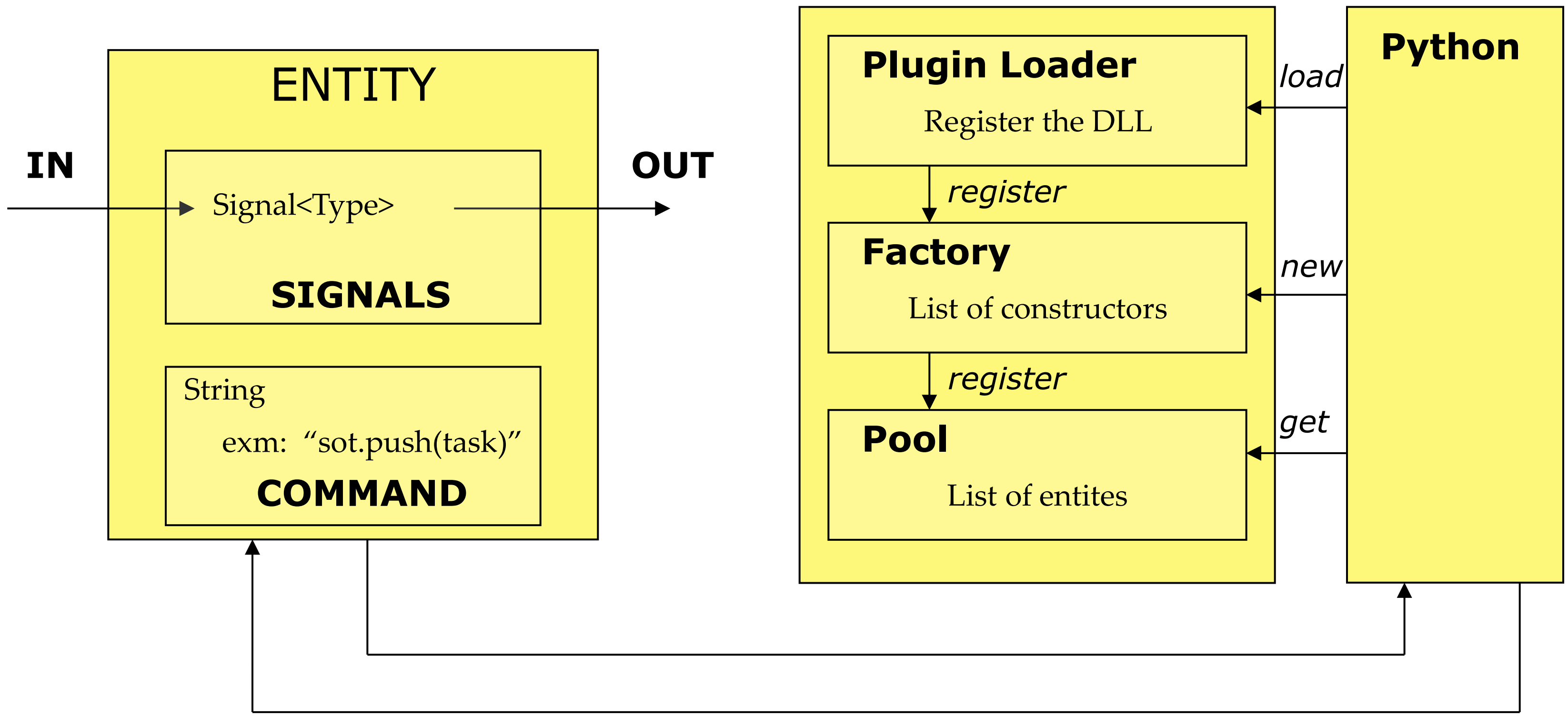
Framework architecture

Packages used



KEITH Francois
CNRS Lirmm

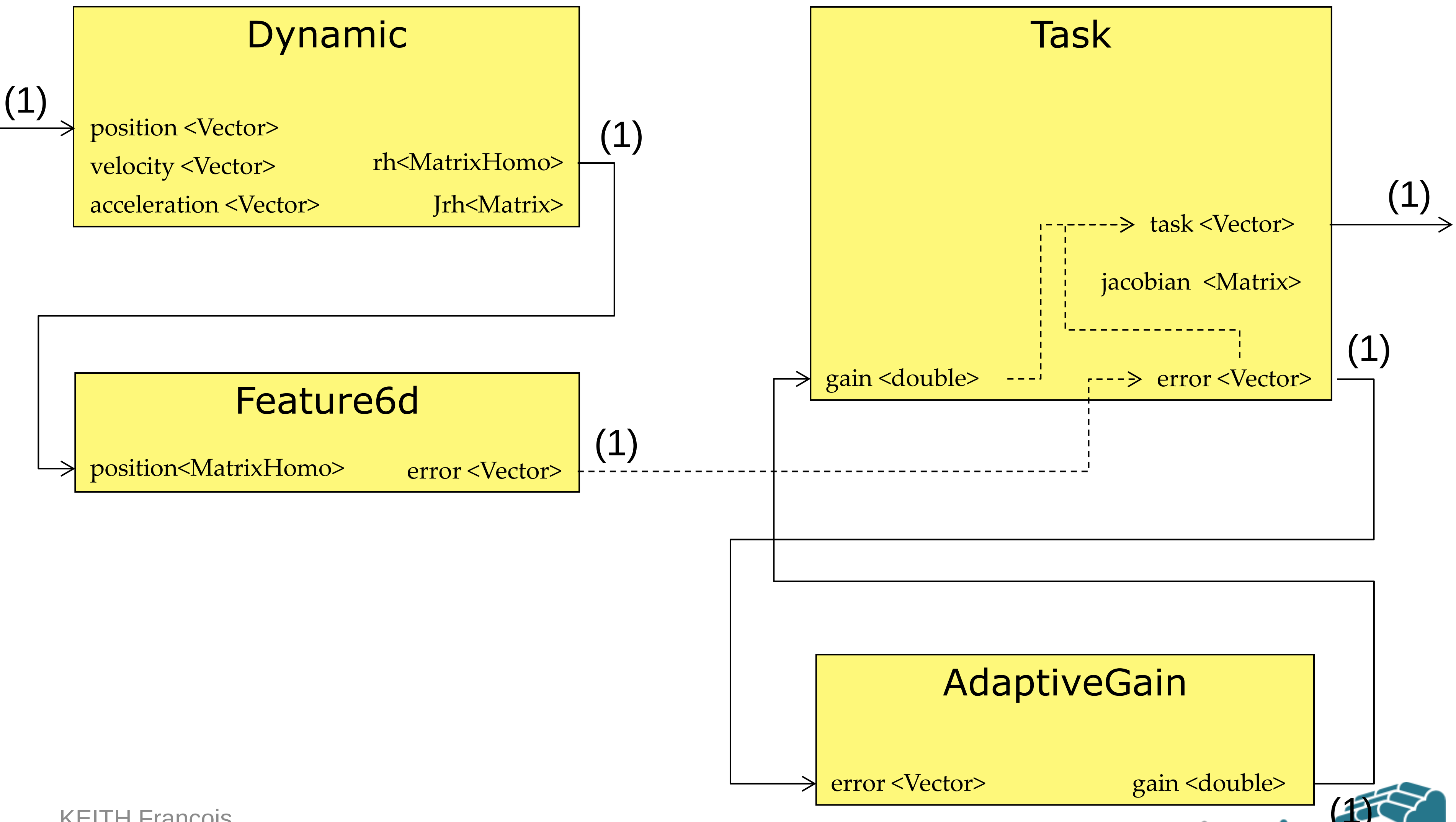
Dynamic-graph architecture



Type of signal handled: bool, int, double, vector, matrix, homogeneous matrix

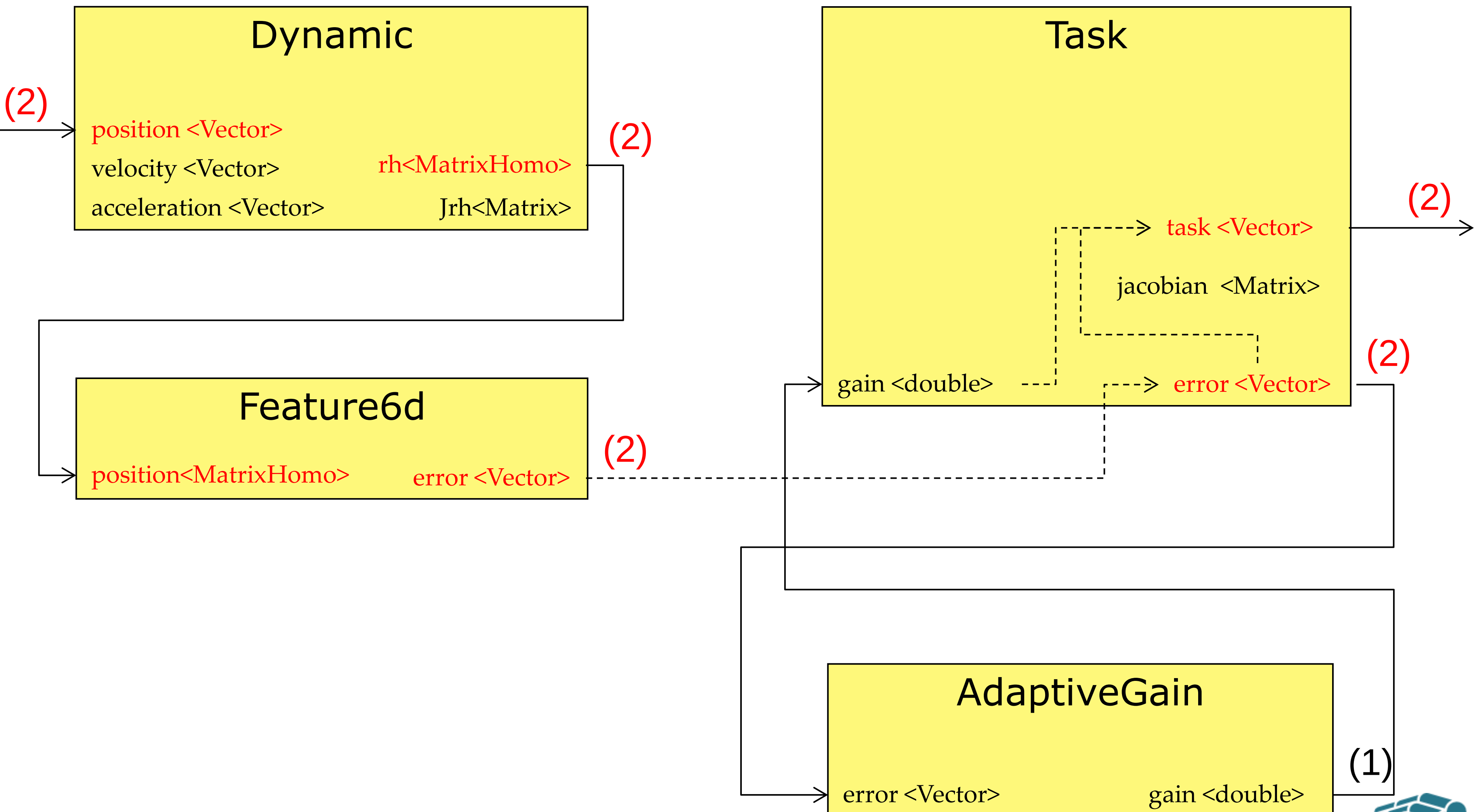
Dynamic-graph architecture

Graph example



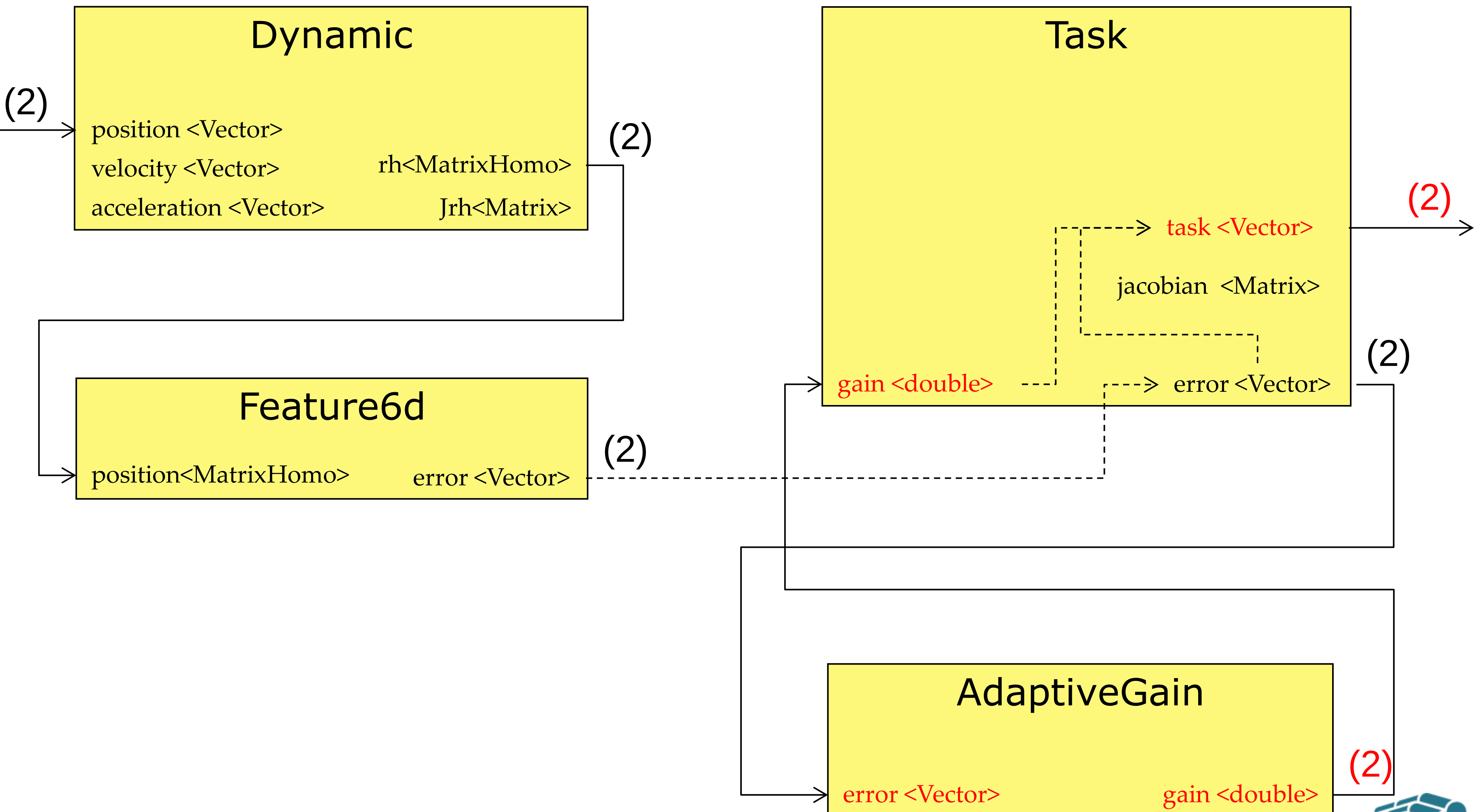
Dynamic-graph architecture

Signal update (step 1/2)



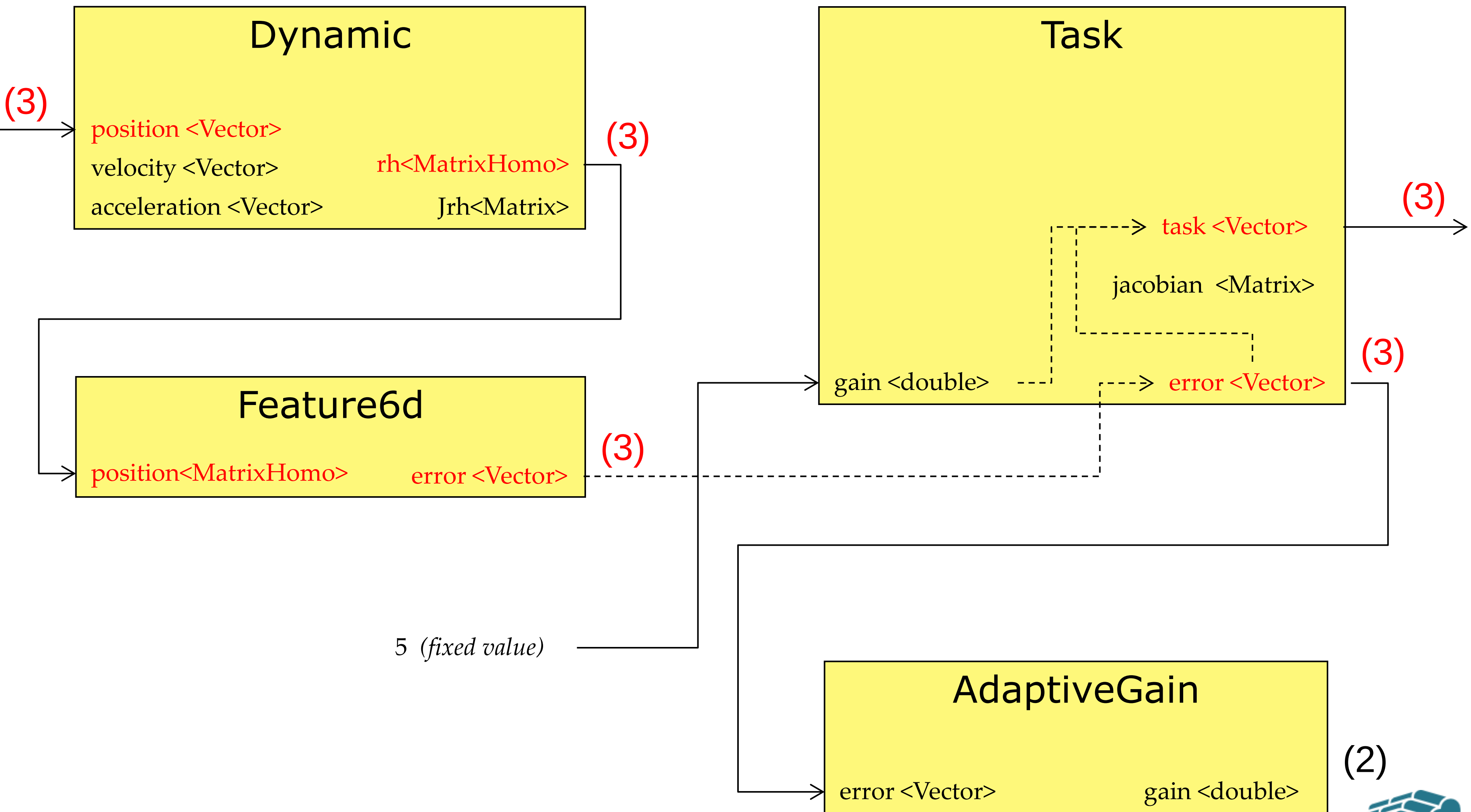
Dynamic-graph architecture

Signal update (step 2/2)



Dynamic-graph architecture

Limited computation



Dynamic-graph architecture

Signal update

- The signals are recomputed only if needed
- No history is kept for the signal values

- No recomputation if the given time is smaller than the current one.
- When using hard coded values, you may need to manually trigger the time of the signal so as to force the signal recomputation.

sot-core

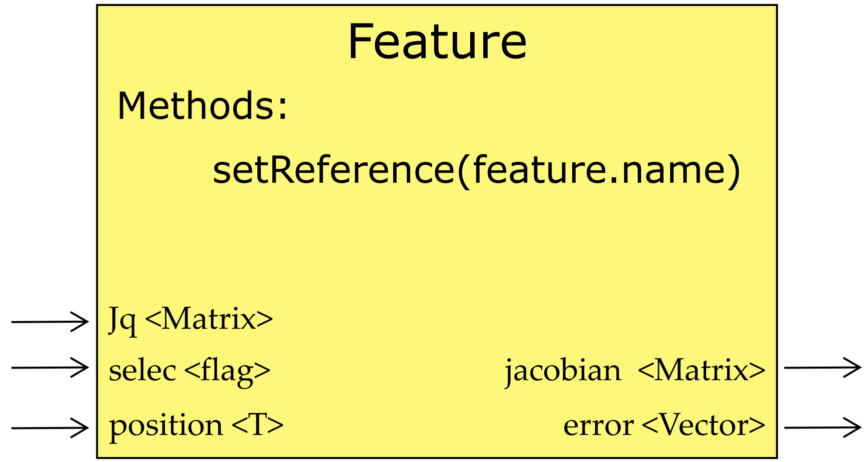
Main components

- Feature
- Task (equality / inequality)
- Solver (basic)
- Gain
- RobotSimu

sot-core

Main components

- **Feature**
- Task
- Solver (basic)
- Gain
- RobotSimu



Usually two features by task: one linked to the OP, one to the desired value.

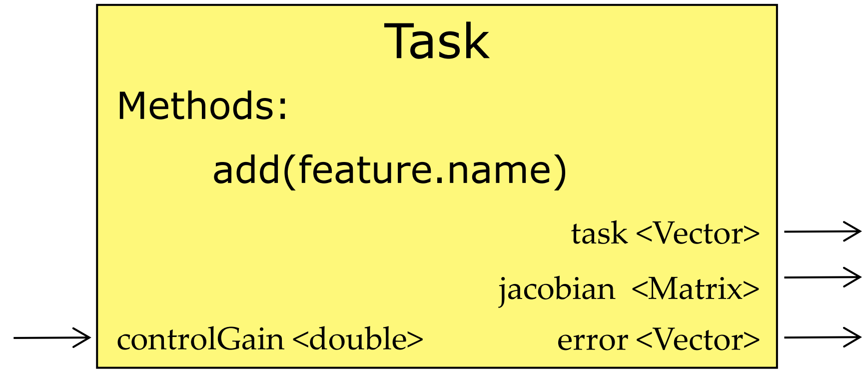
The signal *selec* allow to select the dof controlled.

Outputs the error between the signals $\mathbf{e} = \mathbf{s} - \mathbf{s}^*$

sot-core

Main components

- Feature
- Task (equality / inequality)
- Solver (basic)
- Gain
- RobotSimu

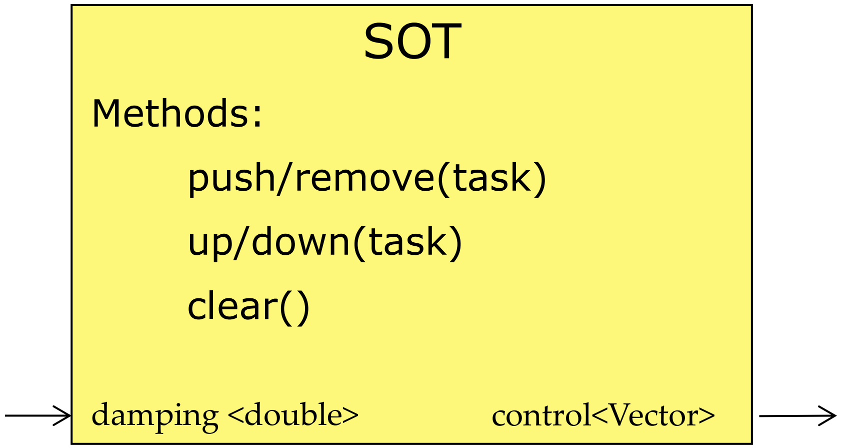


Computes the reference behavior $\dot{\mathbf{e}}^* = -\lambda \mathbf{e}$

sot-core

Main components

- Feature
- Task
- Solver (basic)
- Gain
- RobotSimu

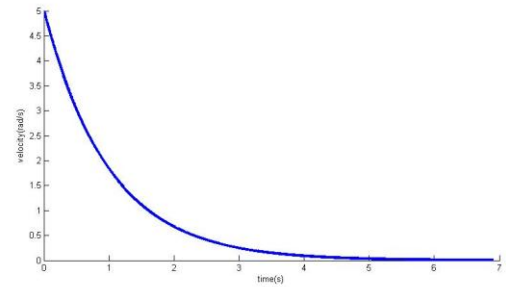


Computes the control law

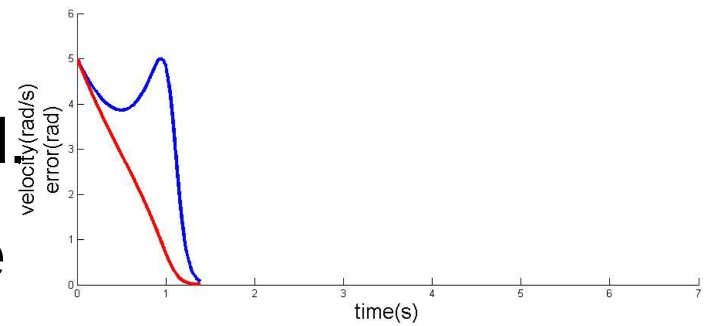
sot-core

Main components

- Feature
- Task
- Solver (basic)
- **Gain**
- RobotSimu



Fixed gain



Adaptive gain

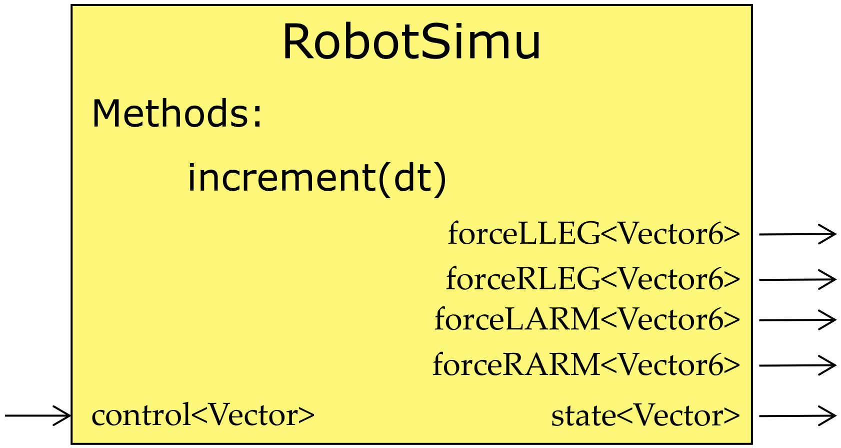
Define *how* the task will be realized.

The trajectory followed remains the same, but the velocity changes

sot-core

Main components

- Task
- Feature
- Solver (basic)
- Gain
- RobotSimu



Simulates the behavior of the robot:

- integrates joint control
- provides forces for the sensors

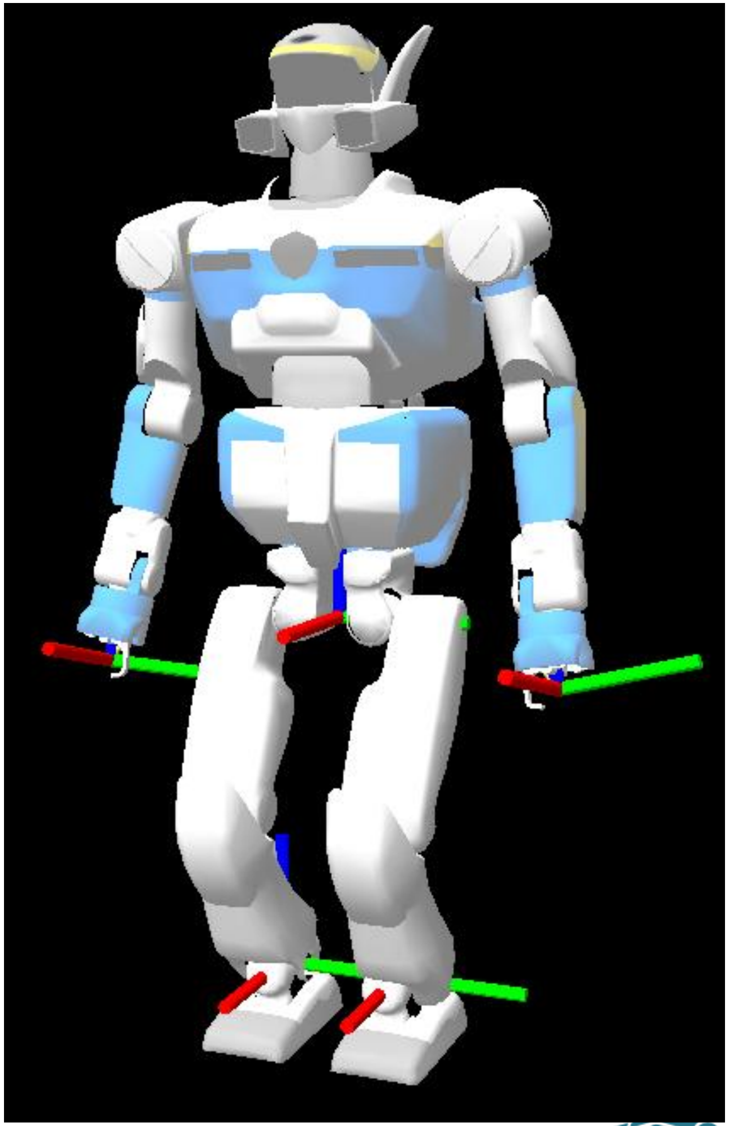
sot-dynamic

- Binds with the RNEA module.
- Implementation of the Entity Dynamic.
  - Inputs: state, velocity
  - Output:
    - position and Jacobian for the operational points left-ankle, right-ankle, left-hand, right-hand, toes, gaze
    - COM, Inertia, non linear effects.
- Binds with optimized implementation

redundant_manipulator_control

Binding with ROS

- ROS stack
- Package dynamics_graph_bridge
  - JointState automatically published
  - Import/Export SoT signals to topics
- Package dynamics_graph_action
  - Starts the interpretor on the remote computer



Building a simulation

- Choose the robot
- Choose the solver
- Define the tasks / contacts
- Choose the displayer (either ROS/robot-viewer)
- Start the simulation

- Minimal example file: ros-kineromeo.py

Building a simulation

Create the robot

- Load of specific data is automatically realized by the import of the appropriated header.

```
from dynamic_graph.sot.romeo.romeo import *
robot = Robot('robot')
```

- This creates the dynamic and a device for kinematic simulation. Also, it plugs their signals altogether.
- Generic dynamic model is loaded from a vrml / urdf file.

Building a simulation

Create the kinematic solver

- 3 possible ways to create the kinematic solver
  - Basic solver:

```
from dynamic_graph.sot.core import *
sot = SOT('sot')
```

  - Solver wrapper

```
from dynamic_graph.sot.dynamics.solver import Solver
solver = Solver(robot)
```

  - Kinematic solver with HCOD

```
sot = SolverKine('sot')
sot.setSize(robot.dimension)
```

- Output of the solver: velocity (sot.control)
- Basic methods: push, pop, remove, add, *addContact, removeContact*

Building a simulation

Define the tasks / contacts

- Tasks handled

Task, TaskInequality, TaskPD

- Choose / create the operational point

'left-wrist', 'right-wrist','left-ankle', 'right-ankle', 'gaze‘, dyn.state…

- Choose the feature (and desired feature if needed)

Feature6d (an homogeneous matrix), Feature3d,
FeatureGeneric, FeatureJointLimits, FeaturePosture

- Choose the gain

Entities: GainAdaptive, GainHyperbolic or constant gain.

Building a simulation

Define the tasks / contacts

- Plug signals altogether.

```
plug (signalOUT, signalIN)
```

- Attach the desired feature to the main feature

```
feature.setReference(featureDes.name)
```

- Attach the feature to the task

```
task.add(feature.name)
```

- Optional: limit the dofs used

```
feature.selec.value = ‘000011’ # constrains X,Y for a 6 dofs constraint
(X,Y,Z,Phi,T,Psi): # /!\ this is reverse polish notation
```

- Fill the stack

```
sot.push(‘task.name’)
```

Building a simulation

Define the tasks / contacts

- There are macros to make the construction of those elements easier
- The macro MetaTaskKine6d gathers the following elements:
    - task’
    - ‘feature’
    - ‘featureDes’
    - ‘gain’
- Realize automatically the signal pluging.

Building a simulation

Run the simulation

- Simulation can be run manually step by step

```
device.increment(timestep)
```

- Shortcuts: *next* (do one increment) / *go* (infinite loop)
- Hand-made sequencing

```
attime(T ,(lambda : command [,description]) )
attime(2 ,(lambda : sot.push(taskCom.task.name),"Add COM") )
```

- Python script started in interactive mode.

```
python –i file.py
```

Building a simulation

Other useful things

- Trace entity
  - Saves registered data in files during the given period of time.
- ROS-independant-viewer: robot-viewer
  - Installation script in sot-script/robot-viewer
  - Based on python-opengl
  - Use the VRML model of the robot.

Hands-in session

Hands-in session

Installation/Compilation

- Get the script from
  git clone https://github.com/francois-keith/sot-script.git
- Electric / Fuerte. (rendering issues for romeo in Groovy?)
- Full instructions for installation/execution in the README
  - Scripts for external dependencies.
  - Compilation from source (./cs_sot.sh)

```
./cs_sot.sh [pull] [build] [rmcache]
```

- 2 packages ROS: *romeo* and *dynamic_graph_bridge*

/!\ Installation process of *dynamic_graph_bridge* is special!

```
rosmake dynamic_graph_bridge; roscd dynamic_graph_bridge/build;
cmake –DCMAKE_INSTALL_PREFIX=SOT_ROOT ..;
make –s install
```

Hands-in session

Execution

- Make sure those environment variable are set

```
export PYTHONPATH=$PYTHONPATH:$SOT_ROOT/lib/python2.7/site-packages
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$SOT_ROOT/lib/
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$SOT_ROOT/lib/plugin
```

- Start roscore

```
roscore
```

- Start the display of the robot with rviz

```
roslaunch romeo sot-display.launch
```

- (or) start the display of the robot with robot-viewer

```
export PYTHONPATH=$PYTHONPATH:$RVIEW_PATH/lib/python2.7/site-packages
alias rview='~/devel/robotviewer/bin/robotviewer –s XML-RPC'
'rview'
```

- Run the python script

```
cd SOT_SOURCE/sot-dyninv/python/ros
python –i file.py
```

Hands-in session

Python commands: cheat sheet

- List basic commands to manipulate entities and signals

entity.help(), *entity*.commands(), *entity*.displaySignals(), entity.signal

signal(time), *signal*.value, *signal*.recompute(T)

plug (*signalOUT*, *signalIN*), *signal*.unplug()

- List main methods of basic entities

Hands-in session

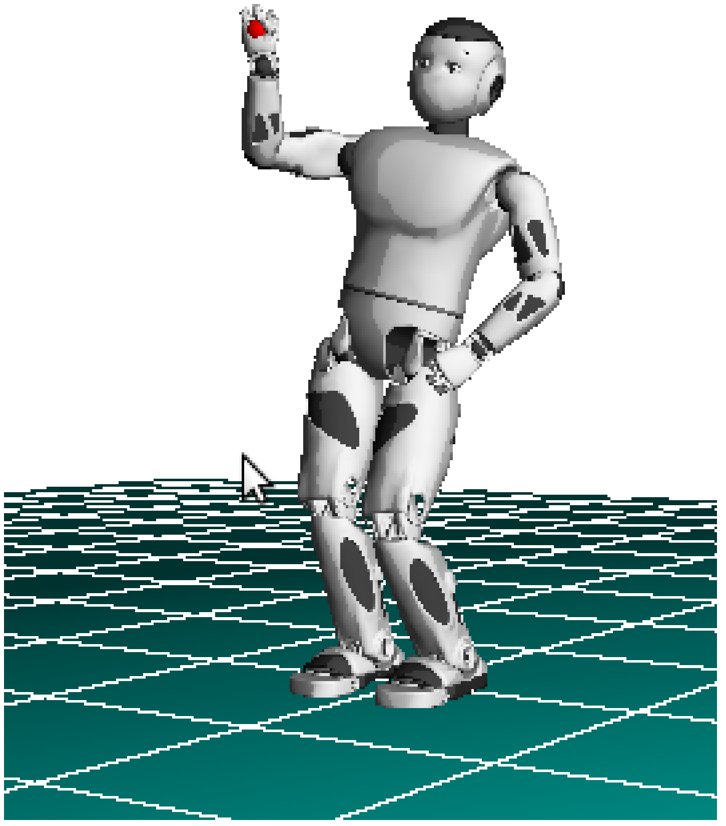
Examples proposed

- Kinematic solver
  - sot-concept.py
    testing the capabilities of the sot
  - ros-kinesimple.py
    Minimalist example using the SoT.
  - ros-walkromeo.py
    Kinematic walk with the romeo robot.
- Dynamic solver
  - ros-planche.py
  - ros-dynromeo.py
    Simple example using the SoT dynamic

Hands-in session

Kinematic example: sot-concept.py

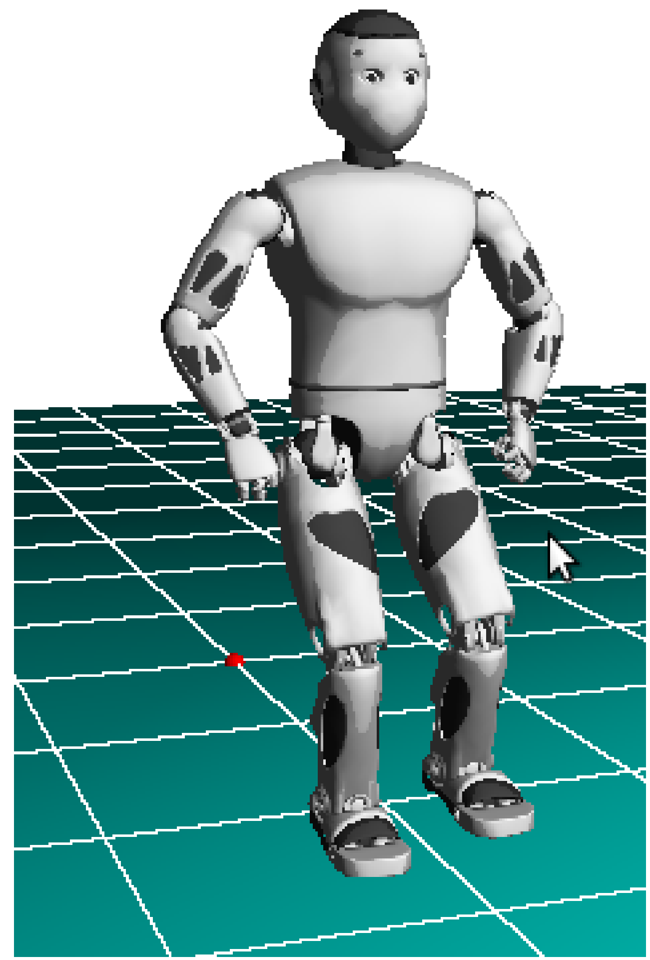
- The robot has to grasp a ball with its right hand. The free flyer is relaxed.
- 7 configurations tested (cf lines 332 to 386) with different constraints for the robot:
- Under constrained robot
- Rupture of balance
- Singular configuration with/without damping.



Hands-in session
Kinematic walk

- Use a pattern generator to define trajectories to follow for the feet and the CoM
- waist remains in the same plan
- 17 dofs (/ 39) controlled

```
pg.pg.setvelocity(x,y, theta)
# -0,2 < x,y < 0,2
```

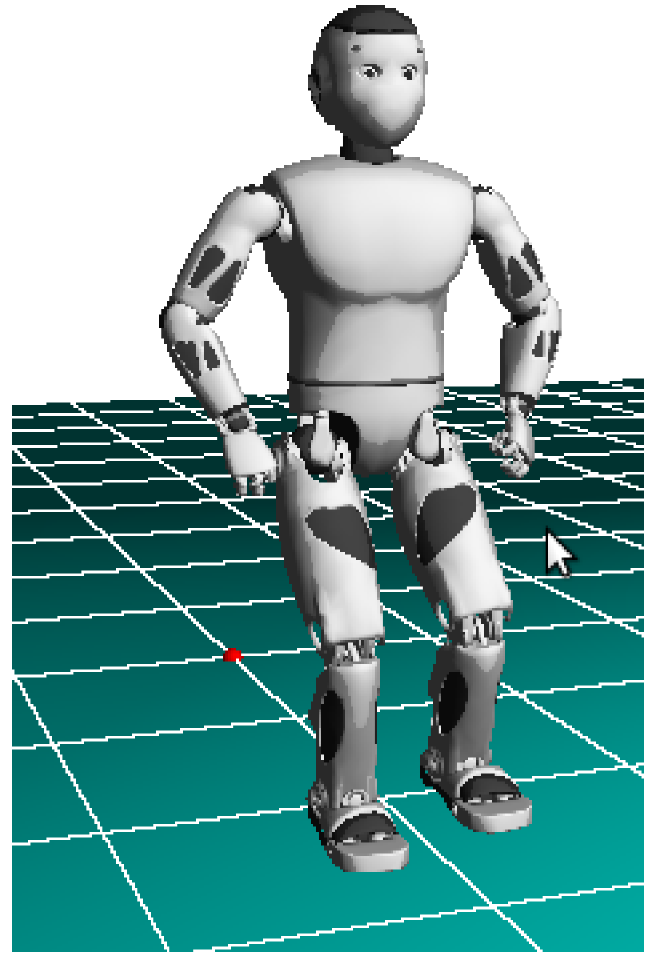


Task CoM	2 dofs
Task Feet	12 dofs
Task Waist	3 dofs

STACK

Hands-in session
Kinematic walk

- Two visible problems
- Romeo use its head and its arms to compensate the COM error



Task CoM

Task Feet

Task Waist

STACK

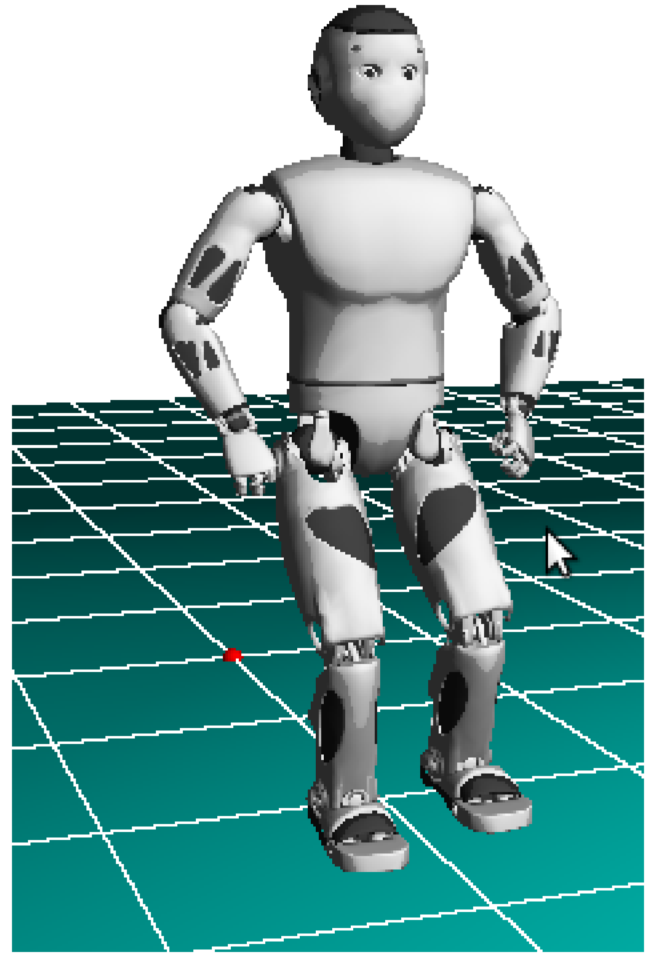
Hands-in session

Kinematic walk

- Define the two tasks in order to correct this issue
- Add an orientation task to control the rotation of the head (follow the orientation of the right foot).
- Prevent the arms from moving backward.

Task Head	14 dofs
Task WaistOr	1 dofs
Task CoM	2 dofs
Task Feet	12 dofs
Task Waist	3 dofs

STACK



Building a dynamic simulation

- Robot:

```
from dynamic_graph.sot.dynamics.solver import Solver
from dynamic_graph.sot.dyninv import *
robot = Robot('robot', device=RobotDynSimu('robot'))
```

- Defines the dynamic solver used

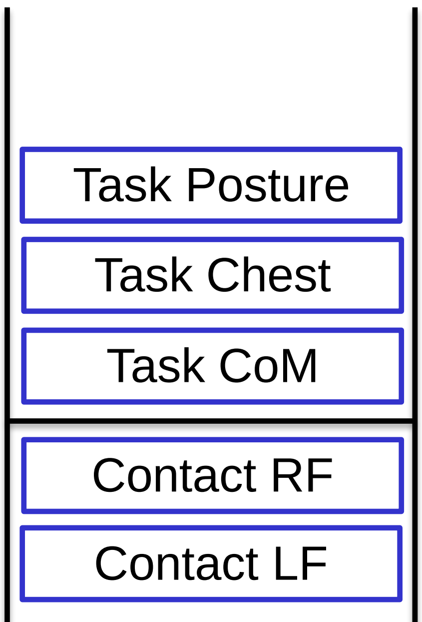
```
sot = SolverDyn ('sot')
sot.setSize(robot.dimension)
```

- Dyn: input: position,velocity
  output: acceleration
- Ensure that the motion realized is dynamically feasible

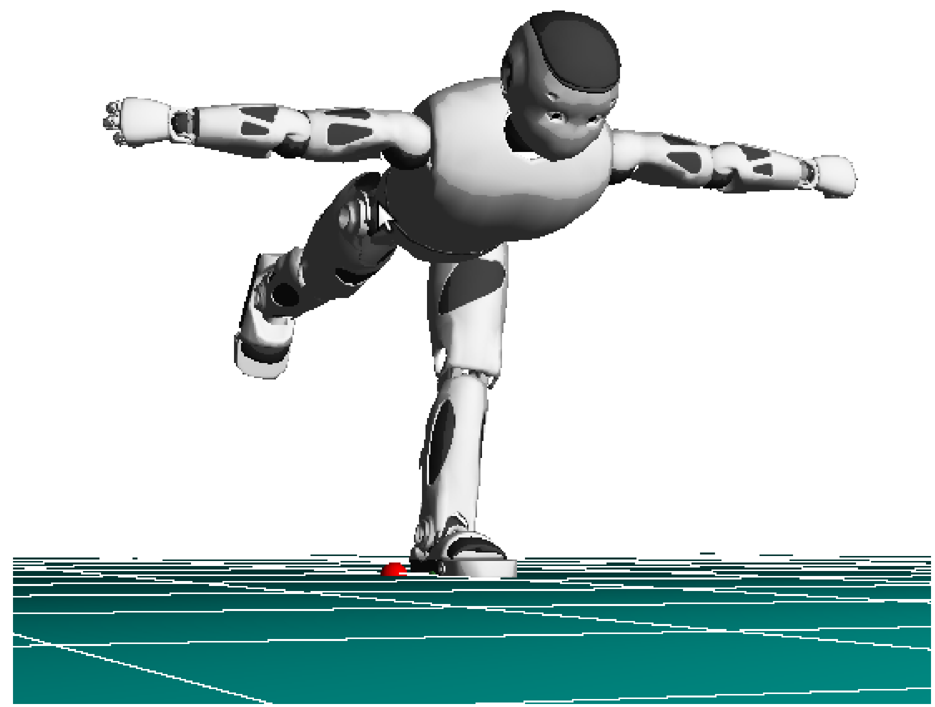
Hands-in session

Complex example using dynamic solver

- Hand-made sequencer
  (lines 225 - 285)



STACK



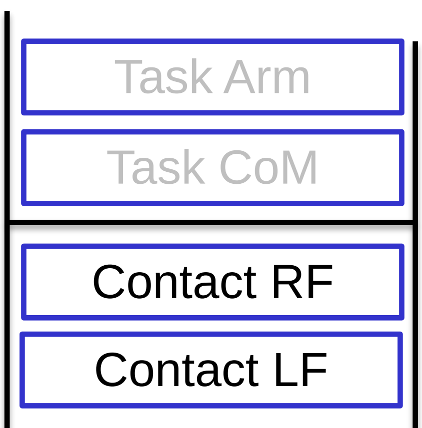
Hands-in session

Basic dynamic solver example

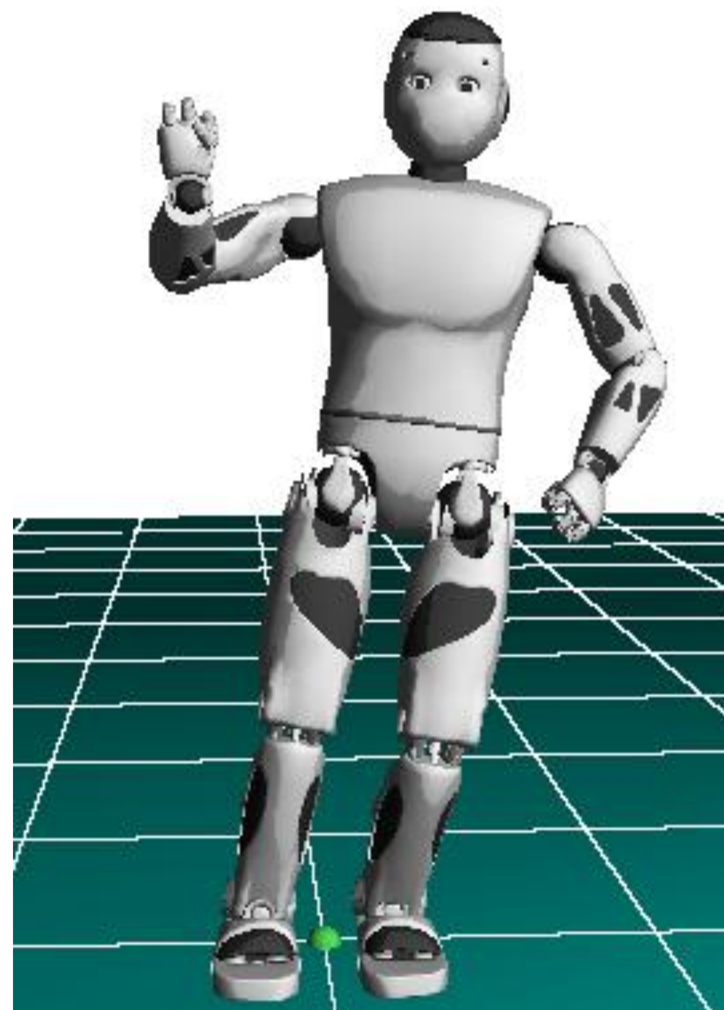
- Change the contacts.
- Lift the right leg: remove the contact and add the lifting task: what happens?

```
sot.rmContact('LF')
```

- Redo it after adding the com task



STACK



Additional references

- 2013 (ijrr): Hierarchical Quadratic Programming
  http://projects.laas.fr/gepetto/index.php/Publications/2012EscandeIjrr
- 2012 (icra): A Dedicated Solver for Fast Operational-Space Inverse Dynamics
- 2012 (itro) : Dynamic Whole-Body Motion Generation under Rigid Contacts and other Unilateral Constraints
- 2009 (icar): A Versatile Generalized Inverted Kinematics Implementation for Collaborative Humanoid Robots: The Stack of Tasks

Thank you for your attention