OpenSoT: a Software Tool for Advanced Whole-Body Control

Enrico Mingo Hoffman

OpenSoT: a Software Tool for Advanced Whole-Body Control

Enrico Mingo Hoffman
Inria Nancy Grand-Est

2024 IEEE 20th International Conference on Automation Science and Engineering
Workshop on Human Movement Understanding, Whole-Body Control, and Human-Robot Interfaces

Inria

iit HHCM ISTITUTO ITALIANO DI TECNOLOGIA
HUMANOIDS AND HUMAN CENTERED MECHATRONICS

OpenSoT

Whole-Body Control

- Defines a set of low-dimensional rules, e.g. equilibrium, self collision avoidance, ...
- Guarantees the correct execution of single task and/or simultaneous multiple tasks, e.g., reaching different objects simultaneously with different end-effectors
- Exploits the full capabilities of the entire body of redundant, floating-based robots in compliant multi-contact interaction with the environment

IEEE-RAS Technical Committee for Whole-Body Control

ieeewbc.org

Open Stack of Tasks (OpenSoT)

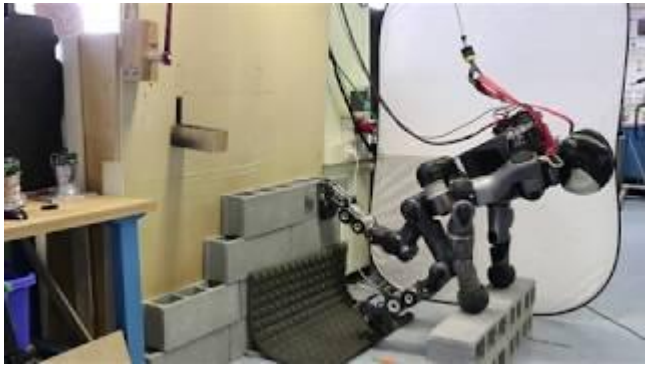
OpenSoT is a C++ library designed to streamline the process of writing and solving QP and LP tailored to whole-body control and robotic applications:

- focus on instantaneous (reactive) whole-body control problems
- modular, scalable, and easy-to-use
- designed for research, educational, and industrial applications

https://github.com/ADVRHumanoids/OpenSoT

OpenSoT

Hierarchical Cartesian Impedance Control



Whole-Body Inverse Kinematics + Statics

Whole-Body Inverse Kinematics

Whole-Body Inverse Kinematics

Features

- **Robot-Agnostic:** support to generic fixed-/floating-base systems
- **Efficient:** based on Eigen for fast and real-time computation
- **Ready & Easy to Use:** out-of-the-box library of *Tasks* and *Constraints* to create complex control problems, efficiently solvable using dedicated *Solvers*
- **Easy to Extend:** C++ API to implement new *Tasks*, *Constraints*, and *Solvers*

OpenSoT Development

OpenSoT v1.0

Development started with the **WALK-MAN** EU Project to participate to the **DARPA Robotics Challenge (DRC)**. Mainly IK engine w/ priorities.

2014-2015

2016-2018

OpenSoT v2.0

Porting to Eigen, ID formulation, new solvers and tasks. Development under the **CENTAURO** and **COGIMON** EU Projects.

OpenSoT v3.0

Closed sourced, development continued under national & private projects, and collaborations.

2019-2023

2024

OpenSoT v4.0

Major refactor, Python bindings, and open-sourced again. Development under the **euROBIN** EU Project.

Actual community:

Introduction

OpenSoT provides an API to setup and solve generic Least-Squares like optimization problems in the form:

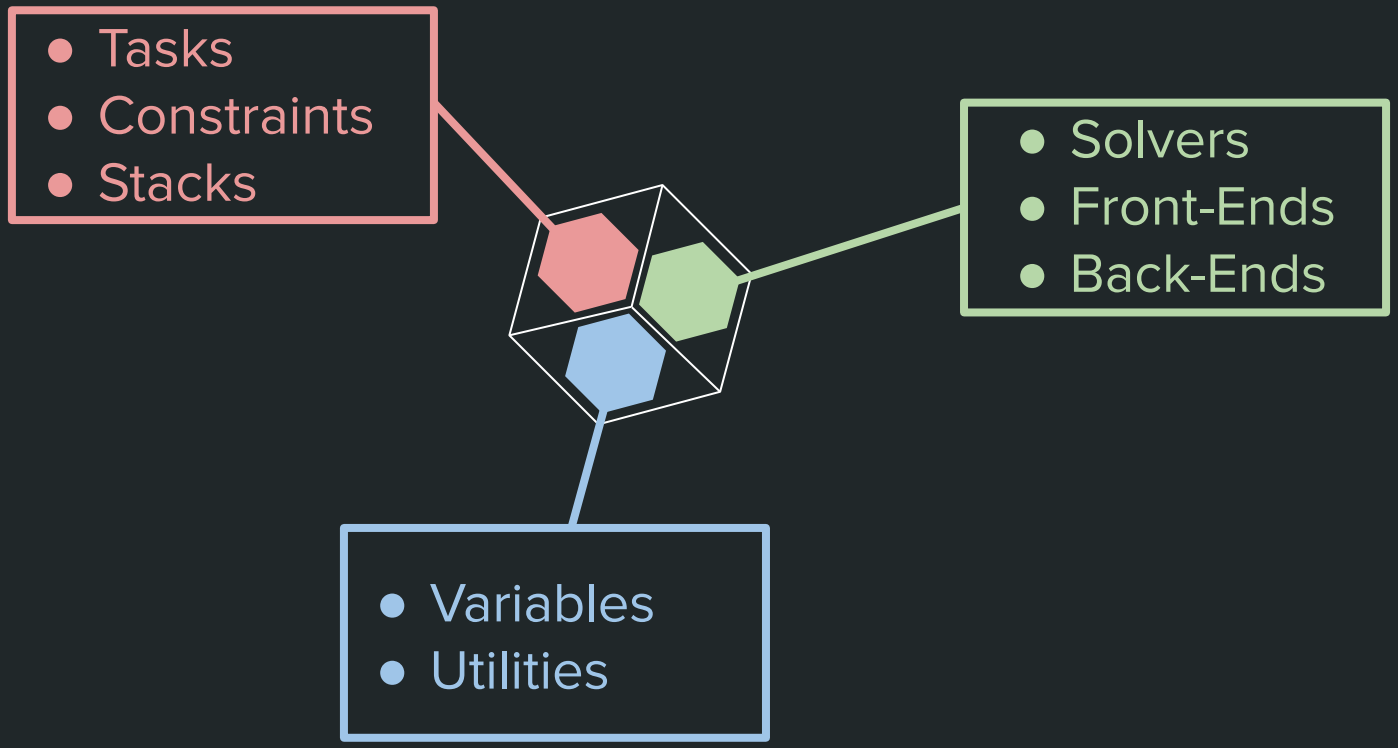
$$\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_{\mathbf{W}}^2 + \mathbf{c}^T\mathbf{x} + \epsilon\|\mathbf{x}\|^2$$

$$s.t. \quad \mathbf{d}_m \leq \mathbf{C}\mathbf{x} \leq \mathbf{d}_M$$

Tasks to be solved

Constraints to fulfill

OpenSoT Concepts



Tasks

A **Task** is an atomic element denoted by:

$$\mathcal{T}_1 = \{\mathbf{A}_1 \in \mathbb{R}^{m \times n}, \mathbf{W}_1 \in \mathbb{R}^{m \times m}, \mathbf{b}_1 \in \mathbb{R}^m, \mathbf{c}_1 \in \mathbb{R}^n\}$$

Task Matrix

Task Weight

Task Vector

Linear Term

which defines the scalar cost function:

$$\mathcal{F}_1 = \|\mathbf{A}_1\mathbf{x} - \mathbf{b}_1\|^2_{\mathbf{W}_1} + \mathbf{c}_1^T\mathbf{x}$$

Sum of Tasks

Multiple tasks can be **summed** together to form complex cost functions, for instance:

$$\mathcal{T}_3 = \mathcal{T}_1 + \mathcal{T}_2 = \left\{ \begin{bmatrix} \mathbf{A}_1 \\ \mathbf{A}_2 \end{bmatrix}, \begin{bmatrix} \mathbf{W}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{W}_2 \end{bmatrix}, \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix}, \mathbf{c}_1 + \mathbf{c}_2 \right\}$$

which defines the scalar cost function:

$$\mathcal{F}_3 = \sum_{i=1}^{2} \left[\left\| \mathbf{A}_i \mathbf{x} - \mathbf{b}_i \right\|_{\mathbf{W}_i}^2 + \mathbf{c}_i^T \mathbf{x} \right]$$

a weighted sum of cost functions associated to each task.

Constraints

A **Constraint** is an atomic element denoted by:

$$\mathcal{C}_1 = \left\{ \mathbf{C}_1 \in \mathbb{R}^{l \times n}, \mathbf{l}_1 \in \mathbb{R}^{l}, \mathbf{u}_1 \in \mathbb{R}^{l} \right\}$$

Constraint Matrix

Lower Bounds

Upper Bounds

which defines the inequality constraint: $\mathbf{l}_1 \leq \mathbf{C}_1\mathbf{x} \leq \mathbf{u}_1$,

or the equality constraint: $\mathbf{C}_1\mathbf{x} = \mathbf{u}_1 = \mathbf{l}_1$

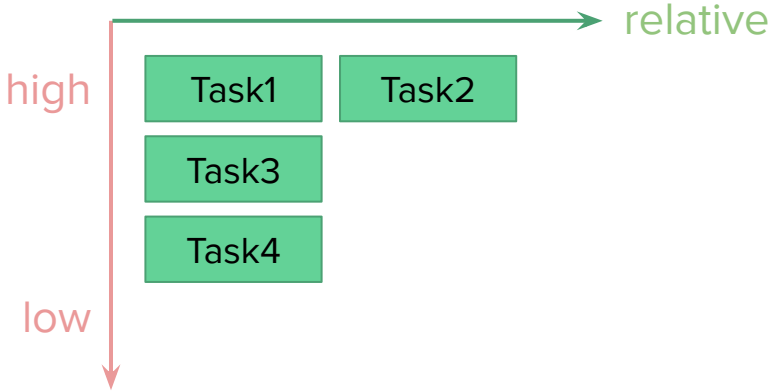
Stacks

A **Stack** consists in one or more tasks and constraints, and their relations.

Tasks can be in **soft** or **hard** priority relations:

In hard priorities, low level tasks can not change optimality of high level tasks.

Implemented through solver strategies.



In soft priorities, tasks are at same level, optimality can change according to relative weights.

Implemented through sum of tasks.

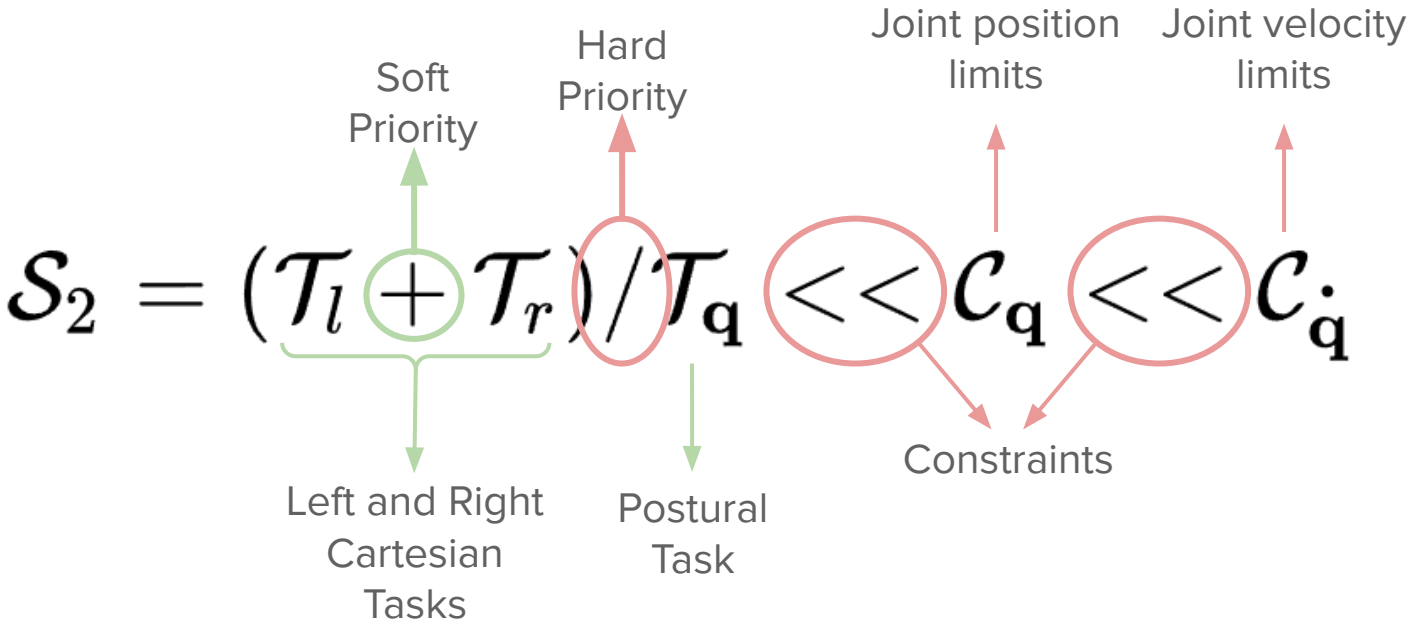
Stacks

Example of Stack:

$$\mathcal{S}_2 = (\mathcal{T}_l + \mathcal{T}_r)/\mathcal{T}_\mathbf{q} << \mathcal{C}_\mathbf{q} << \mathcal{C}_\mathbf{q}^{\cdot}$$

Stacks

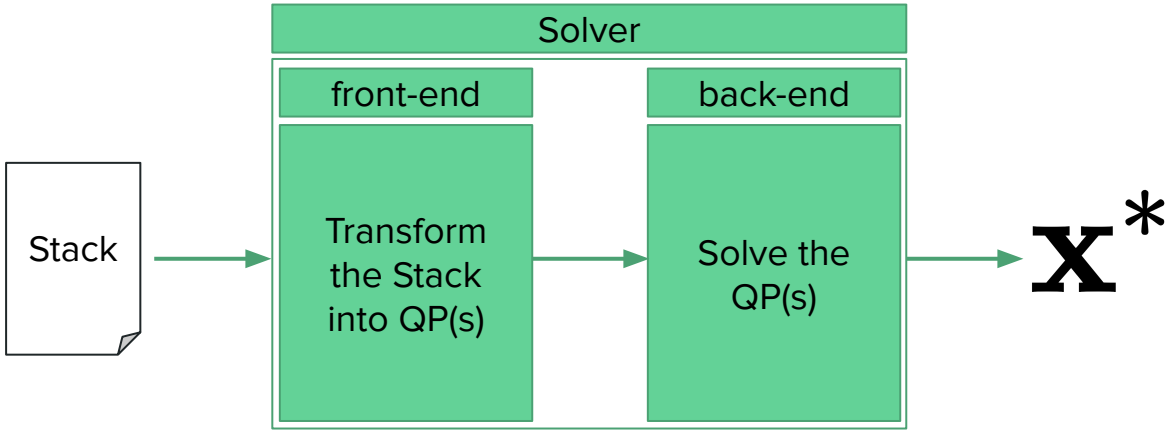
Example of Stack:



Solvers

A **solver** implements a mathematical method to resolve a Stack, i.e. one or more QP/LP problems.

IMPORTANT: Solvers implements hard priorities between tasks.



Solvers

Example: inequality Hierarchical QP (iHQP) [1]

$$(\mathcal{T}_l + \mathcal{T}_r)/\mathcal{T}_\mathbf{q} << \mathcal{C}_\mathbf{q} << \mathcal{C}_{\dot{\mathbf{q}}}$$

$$\dot{\mathbf{q}}_1^* = \underset{\dot{\mathbf{q}}}{\operatorname{argmin}} \|\mathbf{J}_{l+r}\dot{\mathbf{q}} - \mathbf{v}_{d,l+r}\|^2_{\mathbf{W}_{l+r}} + \epsilon\|\dot{\mathbf{q}}\|^2 \quad s.t. \quad \mathbf{l}_{\mathbf{q}+\dot{\mathbf{q}}} \leq \mathbf{C}_{\mathbf{q}+\dot{\mathbf{q}}} \leq \mathbf{u}_{\mathbf{q}+\dot{\mathbf{q}}}$$

1st priority is resolved in 1st QP

$$\dot{\mathbf{q}}_0^* = \underset{\dot{\mathbf{q}}}{\operatorname{argmin}} \|\dot{\mathbf{q}} - \dot{\mathbf{q}}_d\|^2 \quad s.t. \quad \mathbf{l}_{\mathbf{q}+\dot{\mathbf{q}}} \leq \mathbf{C}_{\mathbf{q}+\dot{\mathbf{q}}} \leq \mathbf{u}_{\mathbf{q}+\dot{\mathbf{q}}} \quad \mathbf{J}_{l+r}\dot{\mathbf{q}}_1^* = \mathbf{J}_{l+r}\dot{\mathbf{q}}$$

2nd priority is resolved in 2nd QP, constrained by solution of 1st QP

$$\dot{\mathbf{q}}_0^*$$

solution

[1] O. Kanoun, F. Lamiraux, and P.-B. Wieber, “Kinematic control of redundant manipulators: Generalizing the task-priority framework to inequality task,” IEEE Transactions on Robotics, vol. 27, no. 4, pp. 785–792, 2011.

Variables

An affine variable is defined as:

$$\mathbf{y} = \mathbf{M}\mathbf{x} + \mathbf{p}$$

which can be used to:

- select variables in complex problems: $\mathbf{f} = \underbrace{\begin{bmatrix}\mathbf{0} & \mathbf{I}\end{bmatrix}}_{\mathbf{M}} \underbrace{\begin{bmatrix}\dot{\boldsymbol{\nu}} \\ \mathbf{f}\end{bmatrix}}_{\mathbf{x}}$

- derive complex variables: $\ddot{\mathbf{x}} = \underbrace{\mathbf{J}\ddot{\mathbf{q}}}_{\mathbf{M}\mathbf{x}} + \underbrace{\dot{\mathbf{J}}\dot{\mathbf{q}}}_{\mathbf{p}}$

API (C++)

- **Tasks/Constraints:**
  - base classes to write new components
  - library of out-of-the-box components (velocity/acceleration/force/wrench formulations)
- **Variables:**
  - API to simplify the writing of complex problems, i.e. floating-base ID
- **Operators:**
  - creation of stacks using mathematical operators applied to tasks and constraints
- **Solvers:**
  - base classes to write new solvers and front-/back-ends
  - library of out-of-the-box front-ends and back-ends with SoA QP/LP solvers

Library of Tasks, Constraints, and Solvers

Task	Formulation
Cartesian	Velocity/Acceleration
Postural	Velocity/Acceleration
CoM	Velocity/Acceleration
Angular Momentum	Velocity/Acceleration
Gaze	Velocity
Pure Rolling*	Velocity
Floating-Base Dynamics*	Acceleration + Force
Min Effort	Velocity
Manipulability	Velocity

Constraint	Formulation	Type
Joint Pos./Vel./Acc. Limits	Velocity/Acceleration	Inequality
Torque Limits	Acceleration + Force	Inequality
CoP Limits	Wrench	Inequality
Friction Cone Limits	Force	Inequality
Normal Torque Limits	Wrench	Inequality
Collision Avoidance	Velocity	Inequality
OmniWheels	Velocity	Equality

Solver	Constraints Type	Use Back-end
eHQP	equality only	×
iHQP	equality/inequality	✓
nHQP	equality/inequality	✓
HCOD	equality/inequality	×

Solver	Type
qpOASES	QP/LP
OSQP	QP/LP
proxQP	QP/LP
qpSWIFT	QP
eiQuadProg	QP
GLPK	LP/MIP

* Mostly used as equality constraints

Operators (Math of Tasks)

Operator	Symbol	Stack	Task	Constraint	Properties
soft priority	$+$		$\mathcal{T}_c = \mathcal{T}_a + \mathcal{T}_b$		*commutative* $\mathcal{T}_a + \mathcal{T}_b = \mathcal{T}_b + \mathcal{T}_a$ *associative* $\mathcal{T}_a + (\mathcal{T}_b + \mathcal{T}_c) = (\mathcal{T}_a + \mathcal{T}_b) + \mathcal{T}_c$
sub-task	$\%$		$\mathcal{T}^{[i,j,k]} = \mathcal{T}\%[i,j,k]$		*(indices) commutative* $\mathcal{T}^{[i,j,k]} = \mathcal{T}^{[j,k,i]}$ *NOT distributive* $(\mathcal{T}_a + \mathcal{T}_b)^{[1,2,3]} \neq \mathcal{T}_a^{[1,2,3]} + \mathcal{T}_b^{[1,2,3]}$
multiplication	$\cdot$		$a\mathcal{T} \rightarrow \mathbf{W} = \mathrm{diag}(a), \quad a > 0$ $\mathbf{E}\mathcal{T} \rightarrow \mathbf{W} = \mathbf{E}, \quad \mathbf{x}^T\mathbf{E}\mathbf{x} > 0, \ \forall \mathbf{x}$		*(scalar) distributive* $a\left(\mathcal{T}_a + \mathcal{T}_b\right) = a\mathcal{T}_a + a\mathcal{T}_b$
hard priority	$/$	$\mathcal{S}_{i+j} = \mathcal{S}_i / \mathcal{S}_j$	$\mathcal{S}_{i+1} = \mathcal{S}_i / \mathcal{T}$		*NOT commutative* $\mathcal{T}_a / \mathcal{T}_b \neq \mathcal{T}_b / \mathcal{T}_a$ *associative* $\mathcal{T}_a / \left(\mathcal{T}_b / \mathcal{T}_c\right) = \left(\mathcal{T}_a / \mathcal{T}_b\right) / \mathcal{T}_c$
local constraint	$\ll$	$\mathcal{S}_a / \mathcal{T}_b \ll \mathcal{C} \neq \left(\mathcal{S}_a / \mathcal{T}_b\right) \ll \mathcal{C}$	$\mathcal{T} \ll \mathcal{C}$	$\mathcal{C}_c = \mathcal{C}_a \ll \mathcal{C}_b$	*distributive* $\mathcal{T}_a \ll \mathcal{C} + \mathcal{T}_b \ll \mathcal{C} = \left(\mathcal{T}_a + \mathcal{T}_b\right) \ll \mathcal{C}$ *minor precedence* $\mathcal{T}_a + \mathcal{T}_b \ll \mathcal{C} = \left(\mathcal{T}_a + \mathcal{T}_b\right) \ll \mathcal{C}$
global constraint	$\ll$	$\mathcal{S} \ll \mathcal{C}$	$\mathcal{S}_a \ll \mathcal{C} / \mathcal{T}_b = \left(\mathcal{S}_a / \mathcal{T}_b\right) \ll \mathcal{C}$	$\mathcal{C}_c = \mathcal{C}_a \ll \mathcal{C}_b$	*distributive* $\mathcal{S}_a \ll \mathcal{C} / \mathcal{S}_b \ll \mathcal{C} = \left(\mathcal{S}_a / \mathcal{S}_b\right) \ll \mathcal{C}$ *minor precedence* $\mathcal{S}_a / \mathcal{S}_b \ll \mathcal{C} = \left(\mathcal{S}_a / \mathcal{S}_b\right) \ll \mathcal{C}$

Example: Whole-Body Inverse Dynamics (Python)

- Imports, model creation and update

```
from xbot2_interface import pyxbot2_interface as xbi
from pyopensot.tasks.acceleration import Cartesian, CoM, DynamicFeasibility, Postural
from pyopensot.constraints.acceleration import JointLimits, VelocityLimits
from pyopensot.constraints.force import FrictionCone
import pyopensot as pysot

...
model = xbi.ModelInterface2(urdf)
...
model.setJointPosition(q)
model.setJointVelocity(dq)
model.update()
```

Example: Whole-Body Inverse Dynamics (Python)

- Variables creation

```
contact_frames = ["left_foot_upper_right",  "left_foot_lower_right",
                  "left_foot_upper_left",   "left_foot_lower_left",
                  "right_foot_upper_right", "right_foot_lower_right",
                  "right_foot_upper_left",  "right_foot_lower_left"]
variables_vec = dict()
variables_vec["qddot"] = model.nv
for contact_frame in contact_frames:
    variables_vec[contact_frame] = 3
variables = pysot.OptvarHelper(variables_vec)
```

Example: Whole-Body Inverse Dynamics (Python)

- Tasks & Constraints creation

```
# Tasks
com = CoM(model, variables.getVariable("qddot"))
...
base = Cartesian("base", model, "world", "base_link", variables.getVariable("qddot"))
...

# Constraints
force_variables = list()
for c in contact_frames:
    force_variables.append(variables.getVariable(c))
fb = DynamicFeasibility("floating_base_dynamics", model, variables.getVariable("qddot"),
force_variables, contact_frames)
...
jlims = JointLimits(model, variables.getVariable("qddot"), qmax, qmin, dqmax, dt)
...
```

Example: Whole-Body Inverse Dynamics (Python)

- Stack & Solver creation

```
# Creates 1st priority level
lv1 = (0.1*com + 0.1*(base%[3, 4, 5]))
for i in range(len(cartesian_contact_tasks_frames)):
    lv1 = lv1 + 10.*contact_tasks[i]

# Creates Stack
stack = (lv1/posture) << fb << jlims << VelocityLimits(model,
variables.getVariable("qddot"), dqmax, dt)
...

# Creates Solver
solver = pysot.iHQP(stack, be_solver=OpenSoT::solvers::solver_back_ends::qpOASES)
```

Example: Whole-Body Inverse Dynamics (Python)

- Control loop

```
while ok():
     model.setJointPosition(q)
     model.setJointVelocity(dq)
     model.update()

     com.setReference(com_ref)
     ...
     stack.update()

     x = solver.solve()
     qddot= variables.getVariable("qddot").getValue(x)
     ...
```

Example: Whole-Body Inverse Dynamics (Python)

Tele-Operation w/ Self-Collision Avoidance

Conclusions

- OpenSoT is a mature library for Whole-Body Control based on optimization
- Add-ons:
  - CartesI/O for high-level interfaces based on ROS (ROS2 is coming...)
  - Visual Servoing based on Visp
  - ...
- Try it on Docker!

https://github.com/hucebot/opensot_docker

Thank You!

2024 IEEE 20th International Conference on Automation Science and Engineering
Workshop on Human Movement Understanding, Whole-Body Control, and Human-Robot Interfaces

enrico.mingo-hoffman@inria.fr