Robot Control for Dummies: Insights and Examples using OpenSoT

Enrico Mingo Hoffman, Alessio Rocchi, Arturo Laurenzi, Nikos Tsagarakis

Robot Control for Dummies: Insights and Examples using OpenSoT

Enrico Mingo Hoffman, Alessio Rocchi, Arturo Laurenzi and Nikos G. Tsagarakis

***Abstract*— In this paper we present *OpenSoT*, an open-source, recently developed software library, that can be used to solve robotics related control problems in a flexible and easy way. OpenSoT includes high-level interfaces to state-of-the-art algorithms for kinematic/dynamic modelling, quadratic programming optimization, cost functions and constraints specification. OpenSoT is implemented in C++ and permits rapid prototyping of controllers for fixed or floating base, highly redundant robots such as (but not limited to) manipulators and humanoids. We discuss the use of OpenSoT from the perspective of the developer and the user, leaving out details on the implementation of the tool. We demonstrate how the software can be used with two examples: control of a redundant humanoid robot through simple inverse kinematics schemes and contact forces optimization.**

I. INTRODUCTION

Recent trends in robotics try to formalize the problem of control as an optimization problem in which a cost function is minimized under a set of constraints. In particular, quadratic programming (QP) optimization has been used as a tool to describe control problems such as, but not limited to, inverse kinematics (IK) [1], footstep planning [2], operational space inverse dynamics and contact force optimization [3]. QP optimization is a powerful tool since it permits to minimize quadratic cost functions under linear constraints and the aforementioned control problems can be easily written in such framework. The fact that the problem formulation becomes general, along with the large number of available solver algorithms, makes it difficult to implement software tools that treat robotic control problems with some generality. While tools do exist that deal with a broad range of robotic control problems, the usage of these tools, in particular for a user who is not directly involved in their actual development, is typically difficult, limited to restricted problem formulations and not extensively tested in real platforms. For example, ControlIt! [4] is constrained to the Operational Space formulation and it does not take into account inequality constraints, OCRA [5] does not provide a user friendly API, RobOptim [6], while offering a lot of flexibility w.r.t. the type of optimization problem to be solved, does not offer robotics-related facilities and finally, the Stack of Tasks [7] relies on a complex and heavy software infrastructure.

The OpenSoT project addresses robotic control problems using a different approach: rather than providing a software with a complex infrastructure, the focus is on a framework that is easy to use and to extend, so that the users can implement their control method of choice, however complex. In OpenSoT, tasks (cost functions) and constraints are atomic entities that can be mixed together in order to setup an optimization problem. A set of out-of-the-box tasks and constraints are already implemented and given to the users to promote reusability and to ease the prototyping of new controllers. Existing tasks and constraints are easy to extend and new ones are easy to write thanks to basic interfaces. The optimization problem can then be solved based on any algorithm strategy (e.g. the equality hierachical QP (`eHQP`) [8], the inequality hierachical QP (`iHQP`) [9] or the hierarchical complete orthogonal decomposition (`HCOD`) [1]) and optimization library (e.g. qpOASES [10], QuadProg++ [11]). Furthermore, OpenSoT does not depend on any specific kinematic/dynamic model library but permits to use any model library of choice (such as RBDL [12] or KDL [13]) through the use of abstract interfaces. Finally, OpenSoT has been extensively used, for real-time control, in many robotics platforms including, but not limited to, COMAN [14], [15], WALK-MAN [16] and CENTAURO [17].

The main contribution of this paper is the presentation of OpenSoT in a form that makes it accessible and inquisitive to use from the average experienced users. We consider QP formalization of classical robotic control problems (IK and contact force optimization) and how, thanks to OpenSoT, it is possible to formalize and solve them using few lines of code.

II. BACKGROUND

A standard formulation for a QP problem is:

$$\begin{aligned} \mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmin}} \; & \|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2_{\mathbf{W}} \\ s.t. \; & \mathbf{A}_{\text{eq}}\mathbf{x} = \mathbf{b}_{\text{eq}} \\ & \underline{\mathbf{u}} \leq \mathbf{x} \leq \overline{\mathbf{u}} \\ & \underline{\mathbf{c}} \leq \mathbf{C}\mathbf{x} \leq \overline{\mathbf{c}} \end{aligned} \tag{1}$$

where $\|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2_{\mathbf{W}} = \mathbf{x}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\mathbf{W}\mathbf{A}\mathbf{x} - 2(\mathbf{A}\mathbf{x})^{\mathrm{T}}\mathbf{W}\mathbf{b} + \mathbf{b}^{\mathrm{T}}\mathbf{W}\mathbf{b}$ is the quadratic cost function that is minimized, $\mathbf{A}_{\text{eq}}\mathbf{x} = \mathbf{b}_{\text{eq}}$ is a linear equality constraint, $\underline{\mathbf{u}} \leq \mathbf{x} \leq \overline{\mathbf{u}}$ is a bound and $\underline{\mathbf{c}} \leq \mathbf{C}\mathbf{x} \leq \overline{\mathbf{c}}$ is a general linear constraint. If the problem (1) consists only in minimizing the cost function under equality constraints, a simple close form solution is known. Considering inequalities, there exist basically two big families of methods to solve the problem in (1): *active set* and *interior point* methods. *Active set* methods consider the constraints as equalities only when they are active, so that the problem (1) is transformed in an equality-constrained QP. At each iteration, the new active set is computed by checking for the violated constraints. *Interior point* methods use barrier

Advance Robotics Department, Istituto Italiano di Tecnologia, Genoa, Italy {enrico.mingo, alessio.rocchi, arturo.laurenzi, nikos.tsagarakis}@iit.it

functions (most of the time *log* functions) to penalize the cost function in the region where the bounds are violated and the solution is found by iterative relaxation of the barrier function.

A large number of remarkable control problems in robotics can be written in the form of (1) [18].

III. STRUCTURE OF OPENSOT

OpenSoT is designed to follow the *Hierachical Paradigm* [19]: at each control loop, the robot makes use of proprioceptive and exteroceptive data to update an internal model, the updated model is used to update tasks and constraints, and finally the next control action is computed by solving a QP problem as shown in Fig. 1. In particular, OpenSoT addresses the last two steps: it

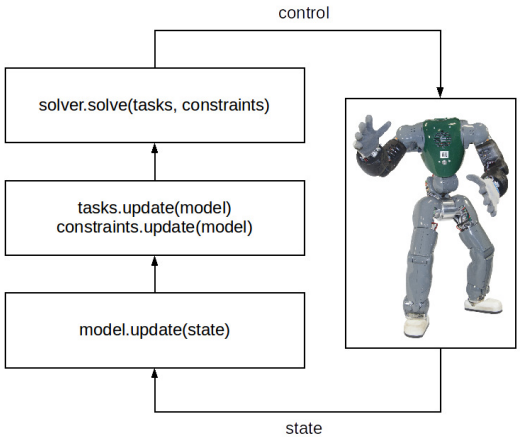

Fig. 1. The *Hierarchical Paradigm*: sense, plan and act

provides tools and interfaces to easily write and use tasks, constraints and solvers. For what concerns the first step, robot and world modeling, a large number of existing C++ libraries provide standard functionalities such as forward kinematics, Jacobians and inverse dynamics computation. Tasks and constraints use quantities computed inside the model library, to build matrices and vectors used inside the solver.

A. OpenSoT Interfaces

The basic objects to describe the problem (1) are the tasks $\mathcal{T}$ and the constraints $\mathcal{C}$. Tasks and constraints use an internal reference to the model $\mathcal{M}$. To be independent from any model library, we use an interface called *ModelInterface*, which is developed inside the *XBotCore* project [20]. The *ModelInterface* provides a clean and standardized way to access kinematic and dynamic quantities, regardless of the specific algorithms that are used under the hood. An implementation based on RBDL [21] is available at `https://github.com/ADVRHumanoids/ModelInterfaceRBDL`.

Fig. 2 shows the main pure virtual methods that the user has to implement to write new tasks, constraints and solvers in OpenSoT as well as to implement the *ModelInterface*. If we consider a new task, the user will implement just four

Task	Constraint	Solver	Model
`getA()`	`getLowerBound()`	`solve()`	`getJacobian()`
`getb()`	`getUpperBound()`		`getPose()`
`getWeight()`	`getAeq()`		`getMass()`
`update()`	`getbeq()`		`getCoM()`
	`getAineq()`		`getTwist()`
	`getbLowerBound()`		`getCoMVelocity()`
	`getbUpperBound()`		`getLinkIndex()`
	`update()`		`...`

Fig. 2. Pure virtual methods to implements new tasks, constraints, solvers and models in OpenSoT

methods: three of them are used to compute the cost function in (1) (`getA()`, `getb()` and `getWeight()`) while the `update()` is used to update the task in the control loop using quantities stored in the model. A similar interface is provided by the constraint base class. Solvers has to provide mainly a `solve()` method that is called at every control loop.

B. Library of Tasks, Constraints and Solvers

In Fig. 3, the library of implemented tasks, constraints and solvers, at the moment in which this paper is written, are shown. The user can write complex control problems using these entities and combining them. In particular, all the

Tasks	Constraints	Solvers
Cartesian	CartesianPose	eHQP [4]
CoM	CartesianVel	iHQP [5]
Admittance	CoMVel	
Manipulability	StaticStability	
MinAcceleration [17]	TorqueLimits [15]	
MinEffort	SelfCollision [16]	
MinVelocity	JointLimits	
Postural	JointVelLimits	
CentrodialDyn	FrictionCones	
Wrench	WrenchLimits	

Fig. 3. Library of out-of-the-box tasks, constraints and solvers. In blue tasks and constraints used for velocity-based IK, in red for contact force optimization

tasks and constraints that are used for velocity-based IK are represented in blue, while the ones used for contact force optimization considering the centroidal dynamics of a floating base robot are in red. Further tasks and constraints regarding Cartesian impedance control, inverse dynamics, force control and acceleration-based IK are under development.

The available solvers at the moment are the `eHQP` and the `iHQP`. The `eHQP` solver is based on classical SVD-based pseudo-inverse and implemented using the *Eigen* library. The `eHQP` solver permits to solve control problems with only equality constraints. The `iHQP` solver is based on *Eigen* and *qpOASES* for the solution of QP with equality and inequality constraints (in particular, the *active set* strategy is used). Both the solvers permits to handle *HARD* and *SOFT* priorities between the tasks. More details on these two solvers are given in the next section.

C. Math of Tasks (MoT)

Inside the OpenSoT library, a series of *operations* between tasks and constraints are defined to ease the formulation of

complex problems. Before introducing these operations, we recall the concept of stack $\mathcal{S}$: a stack is a set of tasks $\mathcal{T}$ and constraints $\mathcal{C}$ at different priority levels [1]. If a constraint is present only at a certain priority level, it is called *local* constraint, while, if it is present at all the priority levels, it is called *global* constraint.

Two (or more) tasks can be in a *SOFT* relative priority using the *Augment* operation:

$$\mathcal{T}_3 = \mathcal{T}_1 + \mathcal{T}_2 \tag{2}$$

basically the resultant task $\mathcal{T}_3$ is the weighted norm of the two tasks $\mathcal{T}_1$ and $\mathcal{T}_2$.

If two tasks are in a *HARD* relative priority, then a new stack is generated:

$$\mathcal{S} = \mathcal{T}_1 / \mathcal{T}_2 \tag{3}$$

in particular, this means that the $\mathcal{T}_2$ is solved keeping the optimality of $\mathcal{T}_1$.

A constraint can be associated to a task with the *insert* operation:

$$\mathcal{T} << \mathcal{C} \tag{4}$$

in this case the constraint is *local* and applied just at the level in which the task is specified. A constraint can also be inserted in a stack

$$\mathcal{S} << \mathcal{C} \tag{5}$$

in this case the constraint is *global* and applied to all the levels of the stack. The `taskToConstraints` operation permits to transform a task into a constraint.

Two constraints can be merged into one with the *Augment* operation:

$$\mathcal{C}_3 = \mathcal{C}_1 + \mathcal{C}_2 \tag{6}$$

The operations defined from (2) to (6) constitute a simple, yet effective, set of math operations that we can use to describe complex problems, even in C++ code, keeping a clean readability on how the control problem is formalized. For example, we can consider the control problem to make a humanoid robot walk using the floating-base: at the first priority level we control the left and right soles w.r.t. the *world* frame. At the second priority level we control the Center of Mass (CoM) of the robot w.r.t. the *world* frame and subsequently, at the third level, we control the left and right hands w.r.t. the *Waist* frame. Finally we consider a postural task in joint space. To all the priorities we apply joint limits and joint velocity limits. This control problem can be written using the Math of Task (*MoT*) formalism as:

$$\begin{pmatrix} \left({}^{\text{World}}\mathcal{T}_{\text{RFoot}} + {}^{\text{World}}\mathcal{T}_{\text{LFoot}}\right) / \\ {}^{\text{World}}\mathcal{T}_{\text{CoM}} / \\ \left({}^{\text{Waist}}\mathcal{T}_{\text{RWrist}} + {}^{\text{Waist}}\mathcal{T}_{\text{LWrist}}\right) / \\ \mathcal{T}_{\text{Posture}} \end{pmatrix} << \left(\mathcal{C}_{\substack{\text{Joint} \\ \text{Limits}}} + \mathcal{C}_{\substack{\text{Joint Velocity} \\ \text{Limits}}}\right) \tag{7}$$

that in C++ code will be the Listing 1. Here, in the first part we update the model with the actual joint space configuration

Listing 1. C++ Example using iHQP

```
using namespace OpenSoT::tasks::velocity;
using namespace OpenSoT::constraints::
    velocity;
using namespace OpenSoT::solvers;

model.setJointPosition(q);
model.update()

CoM com(q, model);
Cartesian LFoot(q,model,"l_sole","world");
Cartesian Rfoot(q,model,"r_sole","world");
Cartesian LWrist(q,model,
    "l_wrist","Waist");
Cartesian RWrist(q,model,
    "r_wrist","Waist");
Postural Posture(q);
JointLimits joint_limits(q,qmax,qmin);
VelocityLimits vel_limits(M_PI,0.01,
    q.size());

AutoStack auto_stack =(
 (LFoot + RFoot)/
 (com)/
 (LWrist + RWrist)/
 (Posture)<<joint_limits<<vel_limits);

QPOases_sot solver(auto_stack->getStack(),
    auto_stack->getBounds(),1e10);

while(1){
    model.setJointPosition(q);
    model.update();
    stack.update(q);

    if(solver.solve(dq))
        q += dq;}
```

sensed from the robot, then we create the needed tasks and constraints passing the model reference to them. The `stack` contains the definition of the control problem (7) and it is passed to the solver. In the control loop, we update the model, update the stack and then we ask for a solution from the solver. The resulting code is simple and readable as its *MoT* formulation.

Furthermore, we have defined three other important operations that can be applied to tasks. The `subTask` operation permits to select *rows* of the task while the `activeJointMask` permits to select columns of the task (setting to zero the disabled one). For example, considering the IK problem, if we are interested to control just the position (not the orientation) of left wrist w.r.t. the torso frame, without using the joints of the torso, we can use the `subTask` to select just the position task and with the `activeJointMask` we can disable the joints of the torso. Finally, the `setActive` method permits to activate and deactivate a task or a stack during the execution.

IV. USE CASES AND EXAMPLES

In this section we introduce the use of the **OpenSoT** library in two realistic robot control examples exploring the *Velocity-based IK* and the *Contact Force Optimization*.

Listing 2. C++ Example using eHQP

```
1  ...
2
3  DampedPseudoInverse solver(
4      auto_stack->getStack()));
5
6  ...
```

A. Velocity-based IK

In *Velocity based IK* we want to find the joint velocities $\dot{\mathbf{q}}$ that realize a certain Cartesian velocity $\dot{\mathbf{x}}$ of the *end-effector*. Depending on the type of solver one may decide to use, there may be the possibility to set inequality constraints to the IK problem. We are going to consider an IK problem stated as in (7) (or in Listing 1, lines 22 to 25).

If we use the `eHQP` solver we can consider task hierarchies and equality constraint and it is implemented using the iterative algorithm in [8]:

$$\begin{aligned} \dot{\mathbf{q}}_0 &= \mathbf{0} \\ \dot{\mathbf{q}}_i &= \dot{\mathbf{q}}_{i-1} + (\mathbf{J}_i\mathbf{P}_{i-1})^\dagger (\dot{\mathbf{x}}_i - \mathbf{J}_i\dot{\mathbf{q}}_{i-1}) \\ \mathbf{P}_0 &= \mathbf{I} \\ \mathbf{P}_i &= \mathbf{P}_{i-1} - (\mathbf{J}_i\mathbf{P}_{i-1})^\dagger \mathbf{J}_i\mathbf{P}_{i-1} \end{aligned} \tag{8}$$

where $\mathbf{J}_i$ is the task Jacobian and $(\cdot)^\dagger$ is the Moore-Penrose pseudo-inverse. The algorithm in (8) is a QP solver that can handle only equality constraints. To use this solver, we are going to change lines 27 and 28 in Listing 1 with the ones in Listing 2. in which we just use the defined stacks.

If one decides to use the `iHQP` solver, we can consider task hierarchies and equality/inequality constraints. We will set up n QP problems (with n equal to the number of level of priorities) of the form:

$$\begin{aligned} \underset{\dot{\mathbf{q}}}{\text{argmin}} \ & \|\mathbf{J}_i\dot{\mathbf{q}}_i - \dot{\mathbf{x}}_i\|^2 + \lambda\|\dot{\mathbf{q}}_i\|^2 \\ s.t. \ & \underline{\mathbf{c}}_i \leq \mathbf{C}_i\dot{\mathbf{q}}_i \leq \overline{\mathbf{c}}_i \\ & \underline{\mathbf{c}} \leq \mathbf{C}\dot{\mathbf{q}}_i \leq \overline{\mathbf{c}} \\ & \underline{\mathbf{u}} \leq \dot{\mathbf{q}}_i \leq \overline{\mathbf{u}} \\ & \mathbf{J}_{i-1}\dot{\mathbf{q}}_{i-1} = \mathbf{J}_{i-1}\dot{\mathbf{q}}_i \\ & \vdots \\ & \mathbf{J}_0\dot{\mathbf{q}}_0 = \mathbf{J}_0\dot{\mathbf{q}}_i \end{aligned} \tag{9}$$

in which we use the *active set* approach implemented in the *qpOASES* library to solve it[1].

The tasks consist of doing some steps (CoM and feet tracking) and a squat motion moving the arm (CoM and arm tracking). During the squat motion, the CoM is commanded to move down of 0.2 $[m]$. Such movement is not doable by the robot without passing the joint limits of the knees.

Fig. 4 shows the joint trajectory produced by the `eHQP` and the `iHQP` solvers. The first one can not handle joint limits, therefore the produced trajectory is not inside the given constraint while, the latter keeps the joint inside the given constraint. The possibility to handle joint limits is paid back in terms of CoM tracking as shown in Fig. 5. Finally

[1]Both the implemented `eHQP` and `iHQP` solvers uses a damping term to avoid kinematic singularities.

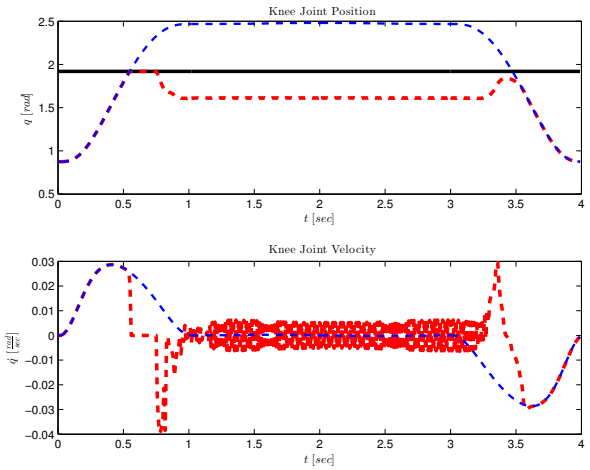

Fig. 4. Knee joint position and velocity during squat motion. The blue and red dashed lines represent the `eHQP` and `iHQP` solver respectively. The black line in the first picture represents the upper limit of the knee joint

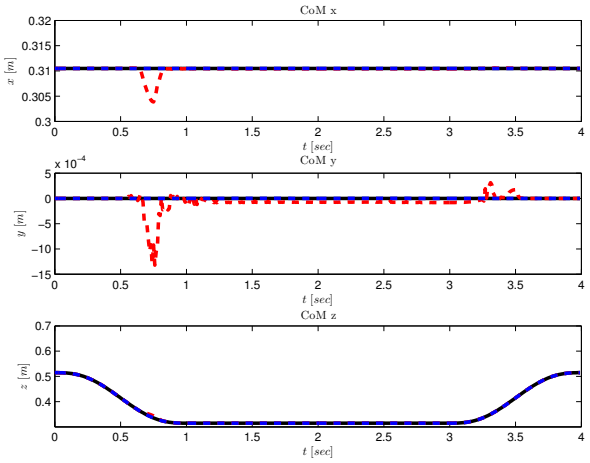

Fig. 5. CoM tracking during squat motion. The blue and red dashed lines represent the `eHQP` and `iHQP` solver respectively, the black line is the reference

Fig. 6 shows the motion of the whole robot.

B. Contact Force Optimization

An important control challenge in humanoid robotics is the Contact Force Optimization given the Centroidal Dynamics of the robot:

$$\begin{aligned} m\ddot{\mathbf{x}} &= \sum_i \mathbf{F}_{c,i} + m\mathbf{g} \\ \dot{\mathbf{h}} &= \sum_i (\mathbf{c}_i - \mathbf{x}) \times \mathbf{F}_{c,i} + \boldsymbol{\tau}_i \end{aligned} \tag{10}$$

where $\ddot{\mathbf{x}}$ is the acceleration of the CoM, $\dot{\mathbf{h}}$ is the variation of centroidal angular momentum, $\mathbf{x}$ is the position of the CoM, m is the total mass of the robot, $\mathbf{g}$ is the gravity vector, $\mathbf{F}_{c,i}$ and $\boldsymbol{\tau}_i$ are respectively the forces and the torques at the *i-th* contact at position $\mathbf{c}_i$ expressed in world frame [22]. Furthermore, it is practice to associate to (10) the *linearized friction cones* constraints in order to generate wrenches at the

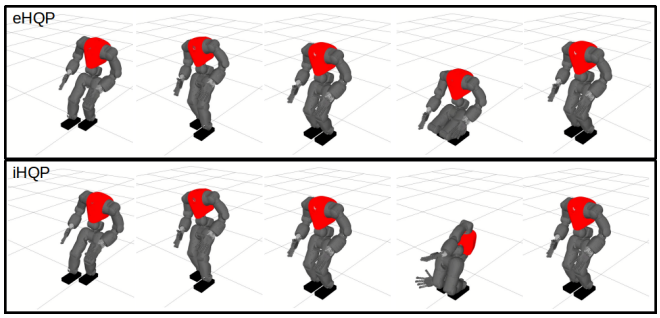

Fig. 6. The upper sequence shows the resulting motion from the `eHQP` solver and the the lower presents the resulting motion of the `iHQP` solver. In the lower sequence we can see how the presence of the joint limits constraint change the whole body motion, trying to fulfill the CoM task

contacts that do not cause slipping. The *linearized friction cones* are written in the form:

$$\begin{aligned} |\mathbf{F}_{\mathrm{t}}| &\leq \frac{\sqrt{2}\mu}{2}\mathbf{F}_{\mathrm{n}} \\ \mathbf{F}_{\mathrm{n}} &\geq 0 \end{aligned} \tag{11}$$

where $\mathbf{F}_{\mathrm{t}}$ and $\mathbf{F}_{\mathrm{n}}$ are respectively tangential and normal forces at the contacts, in the contact frame, and μ is the friction coefficient. The problem stated in (10), constrained by (11), can be expressed in the form of (1) and can be used to compute the contact wrenches that realize a certain motion of the CoM and a certain variation of the centroidal angular momentum, without making the robot slide. Using the *Math of Tasks* formulation, we can write the contact force optimization problem as[2]:

$$\left(\mathcal{T}_{\mathrm{CoM}}\right) << \left(\mathcal{C}_{\substack{\mathrm{Friction}\\\mathrm{Cones}}} + \mathcal{C}_{\substack{\mathrm{Wrench}\\\mathrm{Limits}}}\right) \tag{12}$$

where we added an extra wrench limits constraint that further bounds the problem. In C++ code this will be written as in Listing 3. As example, we consider the task of compensating the weight of the robot ($31.4639\ Kg$) without generating any variation of centroidal angular momentum taking into account the friction cones. We consider the configuration in Fig. 7, where the robot has its feet on two flat rocks and the ankle roll joints rotated of $45\ deg$. The considered friction coefficient is $\mu = 0.5$. The computed wrenches, for the two contacts, expressed in world frame are:

$${}^{\mathrm{world}}\mathbf{w}_{\mathrm{l\_sole}} = \begin{bmatrix} 0 \\ -51.4 \\ 154.3 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad {}^{\mathrm{world}}\mathbf{w}_{\mathrm{r\_sole}} = \begin{bmatrix} 0 \\ 51.4 \\ 154.3 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \begin{bmatrix} N \\ Nm \end{bmatrix} \tag{13}$$

[2]Note that the $\mathcal{T}_{\mathrm{CoM}}$ defined in (7) and the one in (12) are different: in the first one we are referring to the CoM task defined under the namespace `OpenSoT::tasks::velocity` while in the second one we are referring to the centroidal dynamics task defined under the namespace `OpenSoT::tasks::forces`.

Listing 3. C++ Example contact force optimization

```
using namespace OpenSoT::tasks::force;
using namespace OpenSoT::constraints::
    force;
using namespace OpenSoT::solvers;

model.setJointPosition(q);
model.update()

vector<string> links_in_contact;
links_in_contact.push_back("r_sole");
links_in_contact.push_back("l_sole");

CoM com(wrench,links_in_contact,model);
WrenchLimits wrenchLims(300.,
    6*links_in_contact.size())

vector<pair<string,double>> mu;
mu.push_back(pair<string,double>(
    links_in_contact[0], 0.5));
mu.push_back(pair<string,double>(
    links_in_contact[1], 0.5));
FrictionCones friction(wrench,model,mu);

AutoStack auto_stack =(
com<<wrenchLims<<friction);
QPOases_sot solver(auto_stack->getStack(),
    auto_stack->getBounds());

solver.solve(wrench);
```

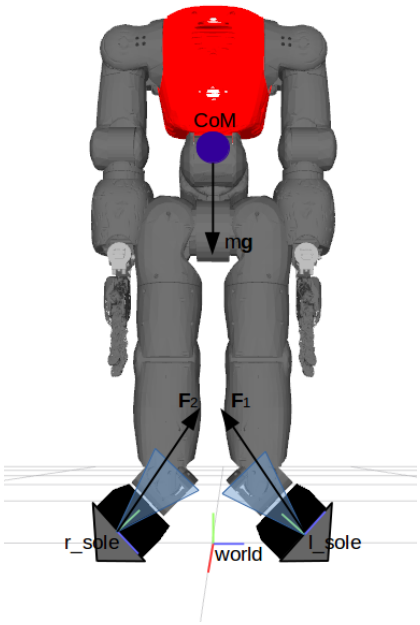

Fig. 7. The robot has its feet on two flat rocks that makes the ankles turn of $45\ deg$

If we consider the forces in (13) expressed at the local frame, we obtain:

$${}^{\mathrm{l\_sole}}\mathbf{F}_{\mathrm{l\_sole}} = \begin{bmatrix} 0 \\ 72.7 \\ 145.5 \end{bmatrix} \quad {}^{\mathrm{r\_sole}}\mathbf{F}_{\mathrm{r\_sole}} = \begin{bmatrix} 0 \\ -72.7 \\ 145.5 \end{bmatrix} \quad [N] \tag{14}$$

that fulfill the given friction cones constraints as it can

be easily checked. If we consider the same optimization computed using the `eHQP` solver, therefore without including the friction cone constraints, the obtained wrenches in world frame are:

$$ {}^{\text{world}}\mathbf{w}_{\text{l\_sole}} = \begin{bmatrix} 0 \\ 0 \\ 154.3 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad {}^{\text{world}}\mathbf{w}_{\text{r\_sole}} = \begin{bmatrix} 0 \\ 0 \\ 154.3 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \begin{bmatrix} N \\ Nm \end{bmatrix} \tag{15} $$

where the forces computed in the local frame are:

$$ {}^{\text{l\_sole}}\mathbf{F}_{\text{l\_sole}} = \begin{bmatrix} 0 \\ 109.1 \\ 109.1 \end{bmatrix} \quad {}^{\text{r\_sole}}\mathbf{F}_{\text{r\_sole}} = \begin{bmatrix} 0 \\ -109.1 \\ 109.1 \end{bmatrix} \quad \begin{bmatrix} N \end{bmatrix} \tag{16} $$

which are clearly outside the specified friction cones.

The two given use cases are part of the tests inside the OpenSoT library that can be found at the url `https://github.com/robotology/OpenSoT`.

V. CONCLUSIONS

We have presented the recently developed, free and open-source control library OpenSoT, which is a software aimed at giving to users an efficient, yet transparent approach to shortening the development time for robotics control related problems. It provides to developers and users a flexible software framework that competes favorably with other ones, but with an easy and more readable syntax and usage, developed for solving control problems for complex robotic systems. We demonstrated how the library can be used to implement different control problems and reported on how the package was used successfully in a real-world application on many different robotic platforms with real-time constraints. More and more successful applications of OpenSoT are now starting to emerge, and it is the belief and hope of the authors of this paper that robotic researchers, working in the area of motion generation and control of complex multi-dof robotic systems, will experiment and validate the proposed software for application-oriented tasks, research and development as well as teaching.

ACKNOWLEDGMENT

The research leading to these results has received funding from the European Union Horizon 2020 robotics program [ICT-23-2014], grant agreement n.644727 CogIMon and the Seventh Framework Programme [ICT-2013-10], grant agreements n.611832 WALK-MAN. The authors want to acknowledge Luca Muratore for the help given to test the `iHQP` solver in the Xenomai real-time environment.

REFERENCES

[1] A. Escande, N. Mansard, and P.-B. Wieber, "Hierarchical quadratic programming: Fast online humanoid-robot motion generation," *The International Journal of Robotics Research*, vol. 33, no. 7, pp. 1006–1028, 2014.

[2] A. Herdt, H. Diedam, P.-B. Wieber, D. Dimitrov, K. Mombaur, and M. Diehl, "Online walking motion generation with automatic footstep placement," *Advanced Robotics*, vol. 24, no. 5-6, pp. 719–737, 2010.

[3] L. Righetti and S. Schaal, "Quadratic programming for inverse dynamics with optimal distribution of contact forces," in *Humanoid Robots (Humanoids), 2012 12th IEEE-RAS International Conference on*. IEEE, 2012, pp. 538–543.

[4] C.-L. Fok, G. Johnson, L. Sentis, A. Mok, and J. D. Yamokoski, "ControlIt! — a software framework for Whole-Body operational space control," *International Journal of Humanoid Robotics*, vol. 0, no. 0, p. 1550040, 9 Oct. 2015.

[5] ISIR. (2014) Optimization-based control for robotics applications: Ocra. [Online]. Available: https://github.com/ocra-recipes

[6] T. Moulard, F. Lamiraux, K. Bouyarmane, and E. Yoshida, "RobOptim: an Optimization Framework for Robotics," in *Robomec*, Tsukuba, Japan, May 2013, p. 4p.

[7] "Stack of tasks development team." [Online]. Available: http://stack-of-tasks.github.io/index.html

[8] F. Flacco and A. De Luca, "Discrete-time redundancy resolution at the velocity level with acceleration/torque optimization properties," *Robotics and Autonomous Systems*, vol. 70, pp. 191–201, 2015.

[9] 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.

[10] H. J. Ferreau, C. Kirches, A. Potschka, H. G. Bock, and M. Diehl, "qpOASES: a parametric active-set algorithm for quadratic programming." *Math. Program. Comput.*, vol. 6, no. 4, pp. 327–363, 2014.

[11] L. D. Gaspero. (2007) Quadprog++. [Online]. Available: https://github.com/liuq/QuadProgpp

[12] M. Felis. (2011) Rigid body dynamics library: Rbdl. [Online]. Available: https://rbdl.bitbucket.io/

[13] R. Smits, "KDL: Kinematics and Dynamics Library." [Online]. Available: http://www.orocos.org/kdl

[14] A. Rocchi, E. M. Hoffman, D. G. Caldwell, and N. G. Tsagarakis, "Opensot: a whole-body control library for the compliant humanoid robot coman," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 1093–1099.

[15] N. Tsagarakis, S. Morfey, G. Cerda, L. Zhibin, and D. Caldwell, "Compliant humanoid coman: Optimal joint stiffness tuning for modal frequency control," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, May 2013, pp. 673–678.

[16] N. G. Tsagarakis, D. G. Caldwell, F. Negrello, W. Choi, L. Baccelliere, V. G. Loc, J. Noorden, L. Muratore, A. Margan, A. Cardellino, L. Natale, E. M. Hoffman, H. Dallali, N. Kashiri, J. Malzahn, J. Lee, P. Kryczka, D. Kanoulas, M. Garabini, M. G. Catalano, M. Ferrati, V. Varricchio, L. Pallottino, C. Pavan, A. Bicchi, A. Settimi, A. Rocchi, and A. Ajoudani, "Walk-man: A high-performance humanoid platform for realistic environments," *Journal of Field Robotics*, vol. 34, pp. 1 – 34, 06/2017 2017.

[17] L. Baccelliere, N. Kashiri, L. Muratore, A. Laurenzi, M. Kamedula, A. Margan, J. Cordasco, S. Malzahn, and N. Tsagarakis, "Development of a human size and strength compliant bi-manual platform for realistic heavy manipulation tasks," in *Intelligent Robots and Systems (IROS), 2017 IEEE International Conference on*, September 2017.

[18] Y. Nakamura, *Advanced Robotics: Redundancy and Optimization*, 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1990.

[19] R. C. Arkin, *Behavior-Based Robotics*. MIT Press, 1998.

[20] L. Muratore, A. Laurenzi, E. M. Hoffman, A. Rocchi, D. G. Caldwell, and N. G. Tsagarakis, "XBotCore: A Real-Time Cross-Robot Software Platform," in *IEEE International Conference on Robotic Computing, IRC17*, 2017.

[21] M. L. Felis, "Rbdl: an efficient rigid-body dynamics library using recursive algorithms," *Autonomous Robots*, vol. 41, no. 2, pp. 495–511, 2017.

[22] H. Dai, A. Valenzuela, and R. Tedrake, "Whole-body motion planning with centroidal dynamics and full kinematics," in *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on*. IEEE, 2014, pp. 295–302.

[23] E. Mingo Hoffman, A. Rocchi, N. G. Tsagarakis, and D. G. Caldwell, "Robot dynamics constraint for inverse kinematics," in *International Conference on Advances in Robot Kinematics, ARK 2016*. IFToMM, 2016, pp. 280–286.

[24] C. Fang, A. Rocchi, E. M. Hoffman, N. G. Tsagarakis, and D. G. Caldwell, "Efficient self-collision avoidance based on focus of interest for humanoid robots." in *Humanoids*. IEEE, 2015, pp. 1060–1066.