Transferring Robotics Software from Research to Industry: the OpenSoT Library

Enrico Mingo Hoffman
Istituto Italiano di Tecnologia, Genoa
enrico.mingo@iit.it

Abstract—Robots are becoming smarter, cheaper and being able to achieve more jobs such as picking and packaging or inspecting products. These constitute a new generation of robots that will guide the next industrial revolution: the Industry 4.0. In the recent years, the robotics research community has developed a large numbers of algorithms and software tools that are suitable for industrial application. In this paper we want to present a software tool, named OpenSoT, that permits to ease the control of highly redundant, floating or fixed base, robots. We think that OpenSoT, and its development, might be of interest for the industrial community, as well as being integrated in well known control frameworks such as MoveIt!.

I. INTRODUCTION

Robots are becoming smarter, faster and cheaper. They are taking new capabilities related to sensing, dexterity, memory and trainability and, as result, they are taking on more jobs such as picking and packaging or inspecting products. A new generation of collaborative robots is rising: they are not anymore closed in cages but they collaborates with human workers who train them through physical demonstration. Thanks to this new capabilities, robots are becoming less specialized in a particular tasks but more and more capable of multiple different tasks. These new capabilities requires a step forward in robotics technology: from the hardware design to the software developing. In particular, the latter, has to be fully reconfigurable, to be adapted to different robotics platforms, quick deployable and easy to use for the non-experts. This new generation of robots will guide the next industrial revolution: the Industry 4.0.

In the recent years, the robotics research community, pushed by international challenges and projects, has developed a large numbers of algorithms and software tools that are suitable for the industrial community. Indeed, some of these software tools, libraries and frameworks, are now used in industry and research and development (R&D) companies. For example, Mujin [1] is a Japanese company that offers solutions for advanced and smart manipulation. Mujin has been created around the OpenRAVE [2] framework that permits to deploy robotics controllers based on many different planning algorithms. OpenRAVE provide an easy to use environment and it is robot independent. ORCOS [3] is a framework devoted to Real-Time (RT) control. It is used, among the many, in PAL Robotics [4], a R&D company located in Barcelona, and NASA JSC [5]. ROS [6], the popular robotics framework, has an industrial branch, namely ROS Industrial [7], which main goal is to further extend and improve industrial robotics applications inside factories which results to be stuck to a defined set of tasks (Figure 1).

In this paper we want to present a software tool, named OpenSoT, developed at the Istituto Italiano di Tecnologia, that permits to ease the control of highly redundant, floating or fixed base, robots. The main goal of this software library is to fast deploy and test different complex controllers in order to easy adapt the same robotic hardware to different high-level tasks. OpenSoT has been designed to be general regarding the controller formulation, modular, robot independent, RT safe and, most important, easy to use.

II. OPENSO T

OpenSoT permits to formulate controllers in the form:

\[ x^* = \arg\min_x \|Ax - b\|_W^2 \]
\[ s.t. \quad A_{eq}x = b_{eq} \]
\[ u \leq x \leq u \]
\[ c \leq Cx \leq c \]

in which we have a cost function to minimize (namely a Task) subject to a set of Constraints.

Many remarkable robotic control problems can be written in the form of (1), for example: inverse kinematics (IK), inverse dynamics (ID), contact force optimization and operational

Fig. 1. Robotics applications in industry are stuck to a limited number of tasks, the challenge of ROS-I is to extend the capabilities of industrial robots

This research was supported by European Union Horizon 2020 robotics program [ICT-23-2014], grant agreement n.644727 CogIMon.

1Image courtesy of ROS Industrial Consortium
space control. In *OpenSoT*, *Tasks* and *Constraints* are atomic entities that can be combined together to formulate controllers. In particular, *Tasks* can be mutually set in *HARD* or *SOFT* priorities while *Constraints*, can be associated to a single *Task* (local Constraint) or to a set of *Tasks* (global Constraint). A library of out-of-the-box *Tasks* and *Constraints*, that covers IK, operational space control and contact force optimization, is provided to the user. New *Tasks* and *Constraints* can be easily written, using the provided interfaces, or can be derived from the existing ones.

*OpenSoT* offers utilities and operators (*Math of Tasks*) to simplify the writing of control problems as the one in (1). This improves the code readability as well as permits to fast prototypes different control solutions. The problem in (1) can be solved by many QP Solvers. *OpenSoT* provides an interface to write brand new Solvers or use the existing ones. The available Solvers are the eHQp (equality Hierarchical QP [8]) and the iHQp (inequality Hierarchical QP [9]).

### III. USE CASE: INVERSE KINEMATICS

As previously mentioned, the IK problem can be written in the form of (1). In particular, considering the iHQp solver, we are going to solve $n$ QP problems (with $n$ the number of priorities or hierarchies) in the form:

$$
\arg\min_{\dot{\mathbf{q}}} \|\mathbf{J}_i\dot{\mathbf{q}}_i - \dot{\mathbf{x}}_i\|_W^2 + \lambda\|\dot{\mathbf{q}}_i\|^2
$$

s.t. $\mathbf{c}_i \leq \mathbf{C}_i\dot{\mathbf{q}}_i \leq \mathbf{\bar{c}}_i$

$\mathbf{\underline{c}} \leq \mathbf{C}_i\dot{\mathbf{q}}_i \leq \mathbf{\bar{c}}$

$\mathbf{\underline{u}} \leq \dot{\mathbf{q}}_i \leq \mathbf{\bar{u}}$

$\mathbf{J}_{i+1}\dot{\mathbf{q}}_{i+1} = \mathbf{J}_{i+1}\dot{\mathbf{q}}_i$

$$
\vdots
$$

$\mathbf{J}_0\dot{\mathbf{q}}_0 = \mathbf{J}_0\dot{\mathbf{q}}_i$

where $\mathbf{J}_i$ and $\dot{\mathbf{x}}_i$ are respectively the Jacobian and the Cartesian velocities of the $i$-th task and $\dot{\mathbf{q}}_i$ are the joint velocities computed at the $i$-th solution step. It is important to notice that the cost function in (2) is slightly changed by a regularization term used to handle singularities. The cost function is minimized under a set of local and global Constraints. In particular, the *local* equality constraints are used to keep the *HARD* priorities between the tasks.

Despite the relatively complex mathematical formulation, the user does not need to write any optimization problem. *OpenSoT* offer a language, the *Math of Tasks* (*MoT*), that can be used to write, for example, IK problems. Figure 3 shows the use of *OpenSoT* to generate motion for the *ABB YuMi* robot. The left end-effector is guided against the right one, the collision between the end-effectors is avoided thanks to the self collision avoidance Constraint. Joint limits and joint velocity limits are also taken into account. Such complex control problem can be written using the *MoT* formalism as:

$$
\begin{align*}
\left(\begin{array}{c}
\mathbf{T}_{\text{World}}\mathbf{T}_{\text{RArm}} + \mathbf{T}_{\text{LArm}}\\
\mathbf{T}_{\text{Posture}}
\end{array}\right) & <= \\
\mathbf{C}_{\text{Joint Limits}} + \mathbf{C}_{\text{Joint Velocity Limits}} + \mathbf{C}_{\text{Self Collision Avoidance}}
\end{align*}
$$

In (3) we are controlling, at the highest priority, the left and right arms w.r.t. the world frame. These two *Tasks* are at the same level (therefore in a *SOFT* relative priority). At the second priority, a *Posture Task*, in joint space, tries to keep a certain joint configuration. These two levels ($n = 2$) of priorities are minimized under the joint limits, joint velocity limits and self collision avoidance Constraints.

Listing 1 represents the C++ code to initialize the Tasks, the Constraints and the Solver for the *ABB YuMi* experiment in Figure 3. It is important to notice that we are considering the self collision only between the 2 end-effector links. Lines 23 to 28 are the code snippet that correspond to (3).

### IV. CONCLUSIONS

*OpenSoT* has been used intensively in many research robotics platforms, such as WALK-MAN (Figure 2), COMAN (Figure 4) and CENTAUR [7] in real-time (RT) scenarios.

RT performances has been obtained with *Cartesian* and *posture Tasks* with joint limits and joint velocity limits Constraints.

Videos are available at [https://www.youtube.com/channel/UCkkZXunCNf6jEwvIcCeK7DrA](https://www.youtube.com/channel/UCkkZXunCNf6jEwvIcCeK7DrA) and *OpenSoT* has been used under Xenomai v2.6.4.
Fig. 3. The ABB YuMi controlled using OpenSoT inside RVIZ. The task consists in controlling left end-effector towards the right one. Thanks to the self collision avoidance Constraint, there are no collisions between the links of the robot. Capsules, around the links, are used to compute collisions and distances. A posture Task, in the null-space of the Cartesian ones, is also used. Two more Constraints are considered: joint limits and joint velocity limits.

```
Listing 1. ABB YuMi Example using iHQP
1 ...  
2 using namespace OpenSoT::tasks::velocity;  
3 using namespace OpenSoT::constraints::velocity;  
4 using namespace OpenSoT::solvers;  
5 model.setJointPosition(q);  
6 model.update();  
7 CoM com(q, model);  
8 Cartesian LArm(q, model, "l_wrist", "world");  
9 Cartesian RArm(q, model, "r_wrist", "world");  
10 Postural Posture(q);  
11 JointLimits joint_limits(q, qmax, qmin);  
12 VelocityLimits vel_limits(M_PI, 0.01, q.size());  
13 list<pair<string,string>> collisions;  
14 collisions.push_back(pair<string,string>("l_wrist","r_wrist");  
15 SelfCollisionAvoidance sca(q, model, "base_link");  
16 sca->setCollisionWhiteList(collisions);  
17 AutoStack auto_stack = (LArm + RArm) / (Posture)<<joint_limits<<vel_limits);  
18 QPOases_sot solver(auto_stack->getStack(), auto_stack->getBounds(), sca, 1e10);  
19 ...  
```

using the iHQP Solver, achieving less than 1 ms control loop.

Many successful applications, based on OpenSoT, are now starting to emerge, and it is the belief and hope of the author that its development might be of interest for the industry community. Finally, the OpenSoT library is suitable to be integrated in well known control framework such as MoveIt!

REFERENCES