RobOptim: an Optimization Framework for Robotics

Thomas Moulard, Florent Lamiraux, Karim Bouyarmane, Eiichi Yoshida

HAL Id: hal-00802083

https://hal.science/hal-00802083v1

Submitted on 19 Mar 2013

RobOptim: an Optimization Framework for Robotics

Thomas Moulard*, Florent Lamiraux†, Karim Bouyarmane‡, Eiichi Yoshida*

*CNRS-AIST, JRL
(Joint Robotics Laboratory),
UMI 3218/CRT,
Intelligent Systems Research
Institute,
AIST Central 2, Umezono 1-1-1,
Tsukuba, Ibaraki 305-8568 Japan
thomas.moulard@gmail.com

†LAAS-CNRS, Université de
Toulouse
7, avenue du Colonel Roche
31077 Toulouse cedex 4, France

‡ATR Computational Neuroscience
Laboratories,
Department of Brain Robot Interface,
2-2-2 Hikaridai, Seika-cho,
Soraku-gun,
Kyoto 619-0288, Japan

Abstract—**Numerical optimization is useful for various areas of robotics. However tackling optimization problems properly requires the use of non-trivial algorithms whose tuning is challenging. RobOptim aims at providing a unified framework for different categories of optimization problems while relying on strong C++ typing to ensure efficient and correct computations. This paper presents this software, demonstrates its genericity and illustrates current use by two full scale robotics examples.**

Index Terms—**Numerical Optimization, Software, Humanoid Robotics**

I. INTRODUCTION

Over the past years, numerical optimization proved itself particularly suited for various robotics applications such as posture or trajectory optimization [1], [2], robot control [3] and more. These applications yield both linear and non-linear optimization problems with equalities and inequalities constraints. Robot control also relies on other types of optimization such as quadratic programming. As robot control algorithms run in real-time, it leads to strong constraints on the implementation efficiency. The design and implementation of a solver is tedious and error-prone. Avoiding numerical precision issues, ensuring that the algorithm behaves properly in all cases even with ill-conditioned problems is challenging, in particular for roboticists who are not necesseraly experts in optimization techniques. Among available optimization toolboxes, the Matlab Optimization Toolbox [4], the Open Optimization library, OPT++, IPOPT [5], SciPy [6] and the GSL (Gnu Scientific Library) [7] all provides several optimization algorithms. Unfortunately, these libraries suffer from several drawbacks: some are difficult to use, some do no provide support for advanced algorithms such as support for constrained optimization or have efficiency issues, etc. They also all lack a unified model expressing optimization problems.

RobOptim solves these limitations by introducing a framework to model any continuous optimization problem, constrained or not. The design is focused on providing a easy-to-use C++ set of safe and efficient libraries which can be used to prototype robotics applications. Let us emphasize here that our framework does not implement any numerical optimization algorithm. Instead, it provides a model for functions, problems and solvers on the one hand and plug-ins wrapping existing solvers as CFSQP or IPOPT on the other hand. The RobOptim computational model will be first introduced in section II and different applications will be detailed in section III. In particular, an extension of RobOptim for a particular category of problem has been implemented recently, demonstrating the ability of RobOptim to support a large variety of problems. The conclusion will detail advantages and limitations of the current approach and summarize the roadmap for the next developments.
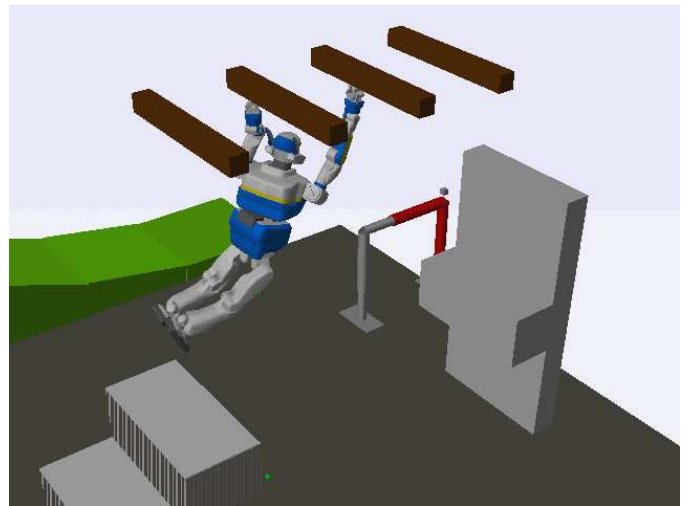

Figure 1. Posture optimization while considering dynamic constraints with RobOptim.

II. ROBOPTIM OVERVIEW

RobOptim is a set of open-source C++ libraries available under the LGPL license. Source code, documentation and examples are provided by the project webpage: http://www.roboptim.net/.

The RobOptim framework is divided into three parts: the solver, core and application layers Figure 2. The core layer provides a computational model expressive enough to formulate different types of optimization problems. The solver layer gathers various optimization algorithms. The application layer consists in several application-dependent packages bundling reusable mathematical functions.

A. Mathematical function representation

Continuous optimization problems can be defined as follows:

$$\min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}) \text{ under the constraint } \mathbf{x} \in \mathbf{X} \qquad (1)$$
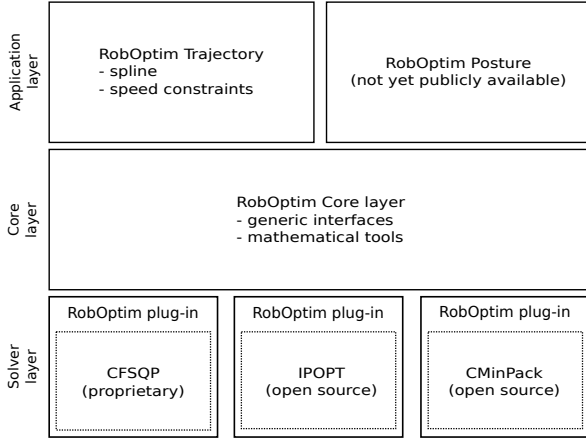
Figure 2. RobOptim three layers architecture: solver (embedding existing state of the Art algorithms), core and applications.

where $f : \mathbb{R}^n \to \mathbb{R}$ is the cost function and $\mathbf{X} \subset \mathbb{R}^n$ is the subspace of the admissible solutions. This space is usually defined by a set of inequality and equality constraints:

$$\mathbf{X} \triangleq \begin{cases} c_i(x) = 0 & i \in \xi \\ c_j(x) \leq 0 & j \in \nu \end{cases} \tag{2}$$

$c_i : \mathbb{R}^n \to \mathbb{R}$, $c_j : \mathbb{R}^n \to \mathbb{R}$ are respectively the set of equality and inequality constraints. i and j are the indices identifying the constraints.

The mathematical functions f and c_k ($k \in \xi \cup \nu$) must provide a way to evaluate their result at any point they are defined. Their associated gradient and hessian can also be provided. Finally, each of these function may be categorized as belonging to a set of functions matching a particular structure which may help the resolution. One example is linear functions. The goal of the RobOptim core layer is to express these features through the C++ typing rules.

Depending on the solving algorithm, it may be necessary to obtain the Jacobian and in some cases even the functions Hessian. Hence, functions providing maximum information about themselves will be compatible with a larger proportion of solvers.

In RobOptim, each mathematical function is represented by a different C++ class. All of these types then inherit the abstract kind of mathematical functions they represent. The following kind of mathematical functions are bundled with RobOptim core: function that can evaluate itself (`Function` type), function that can evaluate itself and its gradient (`DifferentiableFunction`), function that can evaluate itself, its gradient and its hessian (`TwiceDifferentiableFunction`), etc. Let 'A <: B' be the relationship "type B is a subtype (subclass) of A" or "B inherits from A". Then a partial order can be defined where: `Function` <: `DifferentiableFunction` <: `TwiceDifferentiableFunction`.

B. Optimization problem definition and problem resolution

Once the cost function and all constraint functions have been implemented, an optimization problem has to be built. RobOptim core provides a meta-class `Problem` parametrized by two parameters F and C_L. F the cost function type and C_L

is the list of the constraints types. A non-linear problem with constraints then has the following type:

```
Problem<DerivableFunction, vector<LinearFunction,
DerivableFunction > >
```

The constraints can be either linear or non-linear. With this type definition, the constraints will be divided into two categories which will help the solver perform efficiently.

Additionally, bounds can be set on the optimization variables and a starting point for the optimization process may be specified. When the constraints are added to the problem, each one of them is associated with a validity interval. If this interval is reduced to a point, the constraint is an equality constraint. At both compile time and run-time, RobOptim checks that only valid problems are built. For instance if one adds a linear constraint then RobOptim checks at compile time that the constraint is a subtype of the `LinearFunction` type.

A solver that will solve the problem needs to be instantiated. Each solver is parametrized by the same types as optimization problems. Therefore the solver $S < P_1, C_{L1} >$ can solve the problem $P < P_2, C_{L2} >$ if the following relation is true:

$$P_1 <: P_2 \wedge \forall i, C_{L1}(i) <: C_{L2}(i) \tag{3}$$

Basically, the problem can be solved if all types provide enough or more information than necessary. For instance if gradients are required, the function may also provide hessian computation but if gradient is lacking the compile time assertions will fail and prevent the user from building an invalid optimization problem.

By separating problem expression from solver, dynamic changes of the solving algorithm are possible. Each solver is bundled as a plug-in which is loaded at run-time. The interest is to let the user freely change the problem complexity during the design process. Other frameworks would require a different API depending on the kind of optimization at hand, where through RobOptim changes are minimal. One may choose to use a more powerful solver than required at first and then refine the choice or implement later a new plug-in providing the best algorithm for one particular application. These features are provided through meta-programming techniques and come with a near zero cost[1] at runtime and are unique to RobOptim.

C. Cost and constraint toolbox

Unlike others frameworks where computations are tightly linked to one problem and one solver, RobOptim abstraction mechanism allows user to implement toolboxes of reusable functions. This part of RobOptim is dedicated to robotics. The "trajectory" toolbox from the RobOptim application layer is currently providing trajectores defined as B-Splines and associated mathematical functions to realize minimal time optimization for instance.

D. Toward easy and safe problem design

RobOptim is heavily relying on templated meta-programming, a C++ language feature allowing in particular to define parametrized types and execute algorithms at compile time [8]. By defining problems and solvers through parametrized types, the compiler is able to check that the functions used to instantiate the optimization problem are

---

[1] RobOptim core does not realize copy so the additional runtime cost is only due to calls to virtual functions.

compatible with the type of problem under construction. Thus, only solvers supporting this kind of problem can be used. Through meta-programming, these safety checks are evaluated at compile-time and thus do not impact final performances. Regarding ease of use, having a unified representation of all models which matches closely the underlying theory simplifies the understanding of the implementation process. Finally, RobOptim relies on modern tools such as Boost and Eigen to support natural ways of implementing algebra computation. Additional tools such as gradient checks through finite differentiation is also provided to help ensuring functions correctness.

III. Applications and case study

RobOptim has been used to solve several different types of robotics problems. The two scenarii that will be detailed here are footsteps optimization and posture optimization for a humanoid robot. Another important point is the extensibility of the framework. To demonstrate RobOptim capacities to adapt to new types of problems an example of such extension will be given. These experiments aim at generating motion for the humanoid robot HRP-2 [9].

A. Step planning for humanoid robots

Generating a walking motion in an environment cluttered with obstacles is challenging. One commonly used approach is the Rapidly-exploring Random Trees (RRT) method applied to the robot bounding box or to the robot itself sliding on the ground [10]. This probabilistic algorithm will try to create a path between the starting point and the goal point by sampling configurations randomly and is able to find solutions for highly dimensioned problems on a reasonable time. However, the probabilistic nature of these algorithms leads to paths which seem unnatural. One solution to improve these paths is numerical optimization. In this application, RobOptim has been used to optimize a biped robot walking trajectory determined beforehand by a motion planning algorithm.

Let γ be the initial robot waist trajectory defined as a B-Spline from t_{min} to t_{max}. A free time trajectory Γ is defined from T_{min} to $T_{max} = T_{min} + \lambda(t_{max} - t_{min})$ as

$$\Gamma_\lambda(T) = \gamma(t_{min} + \frac{1}{\lambda}(T - T_{min})) \tag{4}$$

Time	control point 1		...	control point n		
λ	x_0	y_0	θ_0	...	x_n	y_n	θ_n

Table I
Optimization variables for walking motion problem

The use of a free time trajectory allows the solver to optimize both trajectory shape and speed by respectively changing the control points and the scale λ. Optimization variables are illustrated by Table I. The free time trajectory derived from the trajectory γ is used as the state of the solver.

Cost function	$f(\mathbf{x}) = \lambda$
Speed constraint	$\forall T, \text{foot}, \quad (\frac{v_{\text{foot}}^x}{v_{\max}^x})^2 + (\frac{v_{\text{foot}}^y}{v_{\max}^y})^2 - 1 \leq 0$
Distance constraint	$\forall T, \quad \text{distance}(\text{obstacle}, \Gamma(T)) > 0$

Table II
Walking optimization problem formulation

The problem cost function is defined as λ. It corresponds to minimizing the time by accelerating the trajectory as much as possible. To preserve a feasible final trajectory while encouraging forward motion, a speed constraint is added which takes separately into account the front speed v^x and the lateral speed v^y of each foot of the robot. Another constraint is preventing the robot from colliding with obstacles. The problem definition is detailed in Table II.
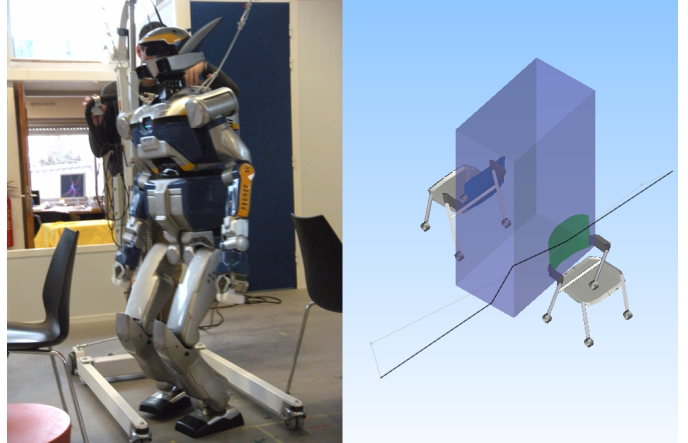

Figure 3. Walking trajectory optimization

This non-linear problem has been successfully solved by RobOptim while relying on the proprietary solver CFSQP [11] internally and result is shown on Figure 3.

B. Posture optimization for humanoid robots

Another problem is posture optimization, which consists in choosing the best configuration such that a system will accomplish the objectives it has been assigned (see 1). In this example, the goal is to find a configuration as close as possible to a goal posture while taking into account various constraints. In this case, robots and environment objects are considered as elements of the optimization problem. Constraints include: robot and objects static equilibrium, Newton's third law, Coulomb friction model, fixed grasp model for bilateral contact forces, joint limits, torque limits and FEM-modeled deformation of the environment [12]. This problem is described extensively in [13], [14] and demonstrated the ability of RobOptim to solve complex large scale non-linear problems.

C. Extending the framework: least square optimization

Initially, non-linear optimization problems with constraints have been solved with RobOptim. By providing a generic computational framework, it is possible to extend RobOptim to support other types of problems such as least square optimization. A least square optimization problem is an unconstrained problem, the cost function of which is defined by

$$f(\mathbf{x}) = \sum_{i=1}^{m} f_i(\mathbf{x})^2 \tag{5}$$

where $f_1, \cdots, f_m$ are m non-linear differentiable functions from $\mathbb{R}^n$ to $\mathbb{R}$.

To build functions matching this definition, another function type called `SumOfC1Squares` has been derived from the

`DifferentiableFunction` class. The constructor of this new type takes as input the function from $\mathbb{R}^n$ to $\mathbb{R}^m$ the coordinates of which are $f_1, \cdots, f_m$. `SumOfC1Squares` then redefines evaluation and gradient of $\sum_{i=1}^{m} f_i(\mathbf{x})^2$.

The CMinPack solver has then been wrapped into a RobOptim solver class which takes as input problem of the type: `Problem<SumOfC1Squares, vector<> >`.

The introduction of this new type of solver did not require any change in the RobOptim infrastructure and proved the genericity of the approach.

IV. CONCLUSION

A new approach to optimization problem representation has been exposed in this paper. By expressing numerical optimization problems through C++ typing, the RobOptim optimization framework provides a unified computational model. Moreover, advanced C++ template meta-programming techniques preserve efficiency while giving the capacity to express high-level mathematical objects such as cost functions and constraints. These features proved useful to solve different types of robotics problems.

However though being generic, the RobOptim core layer suffers from limitations. It still lacks support for optimization in non-scalar spaces such as SO(3). By importing knowledge about the structure of the optimization variables, solvers could realize more efficient computations. For instance, 3D rotations can be represented by homogeneous matrices, quaternions, vectors and angles, etc. Switching from one representation to another may lead to better convergence and a decrease in the number of necessary mathematical operations. One goal would be to both be able to express information regarding the optimization variables structure and to find a method to help solvers rely on these additional information. To achieve this level of expressiveness, modern C++ features are of great help and no optimization framework is providing yet such features when it comes to design an optimization problem.

Therefore, RobOptim is a step forward toward making optimization techniques available for non-expert by providing easy-to-understand and generic model, strong C++ object model ensuring safety and toolboxes bundling robotics oriented functions that may be reused when one builds its own optimization problem.

REFERENCES

[1] S. Miossec, K. Yokoi, and A. Kheddar, "Development of a software for motion optimization of robots - application to the kick motion of the hrp-2 robot," Dec. 2006, pp. 299 –304.

[2] L. Saab, O. Ramos, N. Mansard, P. Souères, and J.-Y. Fourquet, "Dynamic whole-body motion generation under rigid contacts and other unilateral constraints," *IEEE Transaction on Robotics*, November 2012, to appear. [Online]. Available: http://projects.laas.fr/gepetto/index.php/Publications/2012EscandeIjrr

[3] N. Mansard, "A dedicated solver for fast operational-space inverse dynamics," in *IEEE Int. Conf. on Robotics and Automation (ICRA'12)*, St Paul, USA, May 2012.

[4] T. Mathworks. Matlab Optimization Toolbox User's Guide. [Online]. Available: http://www.mathworks.co.uk/access/helpdesk/help/pdf_doc/optim/optim_tb.pdf

[5] L. T. Biegler and V. M. Zavala, "Large-scale nonlinear programming using ipopt: An integrating framework for enterprise-wide dynamic optimization." *Computers & Chemical Engineering*, vol. 33, no. 3, pp. 575–582, 2009. [Online]. Available: http://dblp.uni-trier.de/db/journals/cce/cce33.html#BieglerZ09

[6] E. Jones, T. Oliphant, P. Peterson *et al.*, "SciPy: Open source scientific tools for Python," 2001–. [Online]. Available: http://www.scipy.org/

[7] M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, M. Booth, and F. Rossi, *Gnu Scientific Library: Reference Manual.* Network Theory Ltd., Feb. 2003.

[8] "ISO 14882 Programming Languages – C++," 1998.

[9] K. Kaneko, F. Kanehiro, S. Kajita, H. Hirukawa, T. Kawasaki, M. Hirata, K. Akachi, and T. Isozumi, "Humanoid robot hrp-2," in *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, vol. 2, 26-May 1, 2004, pp. 1083–1090 Vol.2.

[10] S. Dalibard, A. Khoury, F. Lamiraux, M. Taix, and J. Laumond, "Small-space controllability of a walking humanoid robot," in *IEEE International Conference on Humanoid Robots (Humanoids)*, 2011, pp. 739–744.

[11] C. Lawrence, J. L. Zhou, and A. L. Tits, "A C code for solving (large scale) constrained nonlinear (minimax) optimization problems, generating iterates satisfying all inequality constraints," Institute for Systems Research, University of Maryland, Tech. Rep. TR-94-16r1, 1997, http://www.aemdesign.com/download-cfsqp/cfsqp-manual.pdf.

[12] K. Bouyarmane and A. Kheddar, "FEM-based static posture planning for a humanoid robot on deformable contact support," in *2011 11th IEEE-RAS International Conference on Humanoid Robots.* IEEE, Oct. 2011, pp. 487–492. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6100848

[13] ——, "Multi-contact stances planning for multiple agents," in *2011 IEEE International Conference on Robotics and Automation*, no. Section II. IEEE, May 2011, pp. 5246–5253. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5980088

[14] K. Bouyarmane and A. Kheddar, "Static multi-contact inverse problem for multiple humanoid robots and manipulated objects," no. 2. Ieee, Dec. 2010, pp. 8–13. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5686317