Optimization of Industrial Applications with Hardware in the Loop

Matthieu Guilbert, Pierre-Brice Wieber, Luc Joly

Optimization of Industrial Applications with Hardware in the Loop

Matthieu Guilbert
Stäubli Robotics
Faverges, France 74210
Email: matthieu.guilbert@inrialpes.fr

Pierre-Brice Wieber
INRIA Rhône Alpes
Montbonnot, 38330
Email: pierre-brice.wieber@inrialpes.fr

Luc Joly
Stäubli Robotics
Faverges, France 74210
Email:l.joly@staubli.com

Abstract

This paper deals with optimizing the task cycle time of industrial robots integrated in complex robot cells. Trajectory optimizers are usually based on models and can't properly deal with uncertainties due to interactions between the robot and its environment. We propose here a trajectory optimizer with hardware in the loop which can take into account constraints such as maximum authorized temperature and maximum authorized torque. Our approach is based on unconstrained optimization algorithms without derivatives and penalty methods. Experiments on real industrial applications showed good robustness properties of this algorithm even with a high number of parameters and with changes of the robot task.

Index Terms

Industrial robotics, Trajectory optimization, Derivative free optimization, penalty methods.

Optimization of Industrial Applications with Hardware in the Loop

I. Introduction

To reduce production costs, industrial robots must work as fast as possible. But when the speed of a robot is increased, problems such as actuator saturation or overheating may arise what can eventually damage the robot. Due to the complexity of robots and manufacturing systems, only highly qualified operators can reach a high level of efficiency. A better exploitation of the performances of robots integrated in manufacturing systems can only be achieved then by using computer aided optimization methods. Works on robot trajectory optimization usually focus on off-line procedures [1] [2] [3] [4] [5] which find the optimal geometric path or the optimal velocity profile along a specified geometric path. Off-line procedures need a perfect knowledge of both the robot task and the manufacturing system (robot cell) what can be out of reach when the robot cell is formed of several machine tools, conveyors, pallets and scheduled by a programmable logic controller. We propose therefore here a procedure to optimize robot trajectories along a specified geometric path by modifying on-line the parameters defining this trajectory (these parameters can be for example maximum acceleration and maximum velocity on parts of the geometric path). This amounts to minimizing a criterion such as cycle time, subject to limitations of the robot such as maximum authorized temperature and maximum authorized torque, what can be translated into inequality constraints:

$$\begin{aligned} &\min_{p\in\mathbb{R}^n} L(p) \\ &c_i(p) \leq 0,\ 1 \leq i \leq m \end{aligned} \tag{1}$$

with p the parameters of the trajectory, n the number of parameters, $L(p)$ the criterion to minimize, $c_i(p)$ the constraints representing the robot limitations, and m the number of these constraints.

We want to develop a procedure which minimizes the effort of modelling from the operator. We propose therefore a solution with "hardware in the loop":

(i) initialize the set of parameters
(ii) execute the trajectory on the real robot integrated in its manufacturing environment
(iii) record data from the robot's sensors
(iv) find a better set of parameters which improves a given criterion without exceeding robot limitations
(v) go to (ii) until an optimal set of parameters is found.

A similar scheme can be found in iterative learning control methods. Actually, in typical learning control applications, the machine under control repeatedly attempts to execute a prescribed task while an adaptation algorithm successively improves the control system's performance from one trial to the next by updating the control input based on the error signals from previous trials [6]. The method of iterative learning control can be applied to problems with criteria and constraints that must be verified along the whole trajectory such as the problem of regulating the tracking error to zero. But it can't be easily applied to problems with global criteria and constraints, such as cycle time or energetic criteria and constraints, when the calculation is done through an integration over the whole trajectory. We must develop therefore a new optimization method.

Since the criterion and the constraints are evaluated with respect to data measured on the robot, this procedure optimizes the real manufacturing application and not a model of this application, what implies a minimal effort of modelling from the operator. The difficulty in this scheme is the optimization step (iv). Since we use measured data, the gradients of the criterion and the constraints don't exist or can't be obtained easily and efficiently, we must use therefore optimization methods without derivatives. This paper presents in a first section how to minimize a function without constraints and without derivatives, a second part discusses how to take into account constraints when the derivatives aren't available, finally the last section shows practical results on real robot applications.

II. Unconstrained optimization without derivatives

Since we focus on optimization methods without derivatives, direct search methods as discussed in [7] and [8] are to be looked for. The Nelder-Mead simplex method is one of the most frequently used algorithm in optimization without derivatives, but it doesn't converge in some cases and suffer from inefficiency when the dimension of the problem is too large. Other methods such as simulated annealing or genetic algorithms suffer from similar limitations [7].

A real improvement in direct search methods has been obtained when Powell described a method for solving nonlinear unconstrained minimization problems based on the use of conjugate directions [9]. The main idea of this proposal is that the minimum of a positive definite quadratic form can be found by performing at most n successive linear searches along mutually conjugate directions. Then he proposed (independently of [10]) to use the available objective values for building a quadratic model. This model is assumed to be valid in a neighborhood of the current iterate, which is described as a trust region, whose radius is iteratively adjusted. The model is then minimized within this trust region, hopefully yielding a point with a low function value. The difficulty in this kind of algorithms is that the set of interpolation points must have certain geometric properties. Powell proposed therefore an algorithm where the set of interpolation points is updated

in a way that preserves its geometric properties in the sense that the differences between points of this set are guaranted to remain sufficiently linearly independent.

We can find now two efficient algorithms for optimization without derivatives: Derivative Free Optimization (DFO) [8] from Conn, Scheinberg and Toint and NEWUOA from Powell [11]. They are both based on the identification of a quadratic model of a function $\Phi : \mathbb{R}^n \rightarrow \mathbb{R}$ at each iteration of a numerical algorithm. For the k^{th} iteration the model is:

$$m_k(x_k + s) = \Phi(x_k) + < g_k, s > + \frac{1}{2} < s, H_k s > \qquad (2)$$

with $< x, y >$ the standard euclidian scalar product, g_k a real vector and H_k a symmetric matrix. g_k and H_k are estimated by interpolating a set Y of points:

$$\forall\, y \in Y,\ m_k(y) = \Phi(y). \qquad (3)$$

Note that the number of elements of Y must be:

$$p = \frac{1}{2}(n+2)(n+1)$$

in order to estimate all the parameters of the model. The interpolation set Y is updated at each iteration in order to preserve its geometric properties. The number of elements of Y is an important point of difference between DFO and NEWUOA, since NEWUOA actually needs a limited number p of elements in Y:

$$(n+2) \leq p \leq \frac{1}{2}(n+1)(n+2).$$

The important point here is that the evaluation of the original function Φ can be a costly process, especially in our case where a whole application cycle is required each time: the NEWUOA algorithm allows then to reduce strongly the total number of such evaluations with respect to the DFO algorithm, reducing therefore strongly the total cost of the optimization. The number of elements of Y is decreased since NEWUOA identifies the quadratic model by using the equations of interpolation (3) and by minimizing the Frobenius norm of the difference between H_k and H_{k-1} [12]. When the quadratic model is identified, NEWUOA finds its minimum within the trust region, the cost function is then evaluated at this point, if the decrease of the cost function is worse than the decrease predicted by the model, the radius of the trust region is scaled down, the radius is not changed otherwise. The minimization algorithm by successive quadratic approximations can be summarized by:

(i)	Initialize the set Y, the radius of the trust region and the first iterate
(ii)	build the quadratic model
(iii)	Minimize this model within the trust region
(iv)	Update the set of interpolation Y
(v)	Update the radius of the trust region
(vi)	Update the current iterate and go to step (ii)

The algorithm finishes when the radius of the trust region attains a lower bound fixed by the user. NEWUOA appears to be one of the best algorithms available today for minimizing a cost function without derivatives. But NEWUOA doesn't deal with constraints, and our problem is subject to constraints. We need therefore to focus now on how to take into account inequality constraints when the derivatives are not available.

III. Penalty methods in non-linear programming

Historically, the earliest developments in non-linear programming were sequential minimization methods with the use of penalty and barrier functions, what represents a global approach to non-linear programming as opposed to local methods based on the linearization of the constraints. Since we don't have access to the gradients of our criterion and constraints, we must use such sequential methods which can be separated in two classes: penalty methods and exact penalty methods.

A. Penalty methods

Historically, Courant developped a method in order to take into account equality constraints which can be adapted easily to inequality constraints by the use of the $\max$ function [13]. The point is to consider the penalized function:

$$\Phi(p, \sigma) = L(p) + \sigma \sum_{i=1}^{m} [\max(c_i(p), 0)]^2. \qquad (4)$$

This function can be used in an iterative scheme:

(i) Choose a fixed sequence $\{\sigma^{(k)}\}$, for example $\{1, 10, 10^2, 10^3, ...\}$,
(ii) For each σ^k, find a local minimizer $p(\sigma^k)$ to $\min_p \Phi(p, \sigma^k)$,
(iii) Terminate when $\max(c_i(p), 0)$ is sufficiently small.

Proofs of convergence of such a method are available in [13]. There exist other penalty functions such as barrier functions which are usually used in interior point methods. Since these barrier functions, usually inverse or logarithm functions, are not defined when the constraints are active, we won't use this kind of penalty functions.

One experience in the next section will be devoted to compare an exponential penalty function:

$$\Phi(p, \sigma) = L(p) + \sum_i \sigma_i e^{c_i(p)} \qquad (5)$$

to a second class of penalty methods: the exact penalty functions.

B. Exact penalty functions and the augmented Lagrangian method

If the function to minimize $L(p)$ is *k-Lipschitz*, then the penalization:

$$\Phi(p, \sigma) = L(p) + \sigma \sum_i \| \max(c_i(p), 0) \|_1 \qquad (6)$$

is exact if $\sigma > k$, that is to say the minimum of this function is the same as the optimum of problem (1), as proved in [14] or [13].

An advantage is the exactness of the solution to the original problem for a finite σ, with no need to iterate on the value of

σ. But this penalty function (6) is not differentiable at the optimum, and since minimization algorithms without derivatives are not designed for such non differentiabilities, that can lead to severe troubles: we should look for another solution.

Lagrangian methods [15] can also be seen as exact penalty methods when the function to minimize and the constraints are convex. In our case, we don't know whether the functions $L(p)$ and $c_i(p)$ are convex or not, but they are most probably not. The augmented Lagrangian method deals with this kind of problem, with a Lagrangian actually augmented with a quadratic form of the constraints, tending to create a convex basin around the optimum of problem (1) [14]. When Powell discovered this method [16], he exposed this technique in a more intuitive way. Actually, to get the minimum of problem (1) with the Courant penalty method, σ must tend to infinity. To avoid this problem, we could add a shift parameter θ_i for each constraint :

$$\Phi(p, \theta, \sigma) = f(p) + \frac{1}{2} \sum_i \sigma_i (\max(c_i(p) - \theta_i, 0))^2. \quad (7)$$

More details are available in [13].

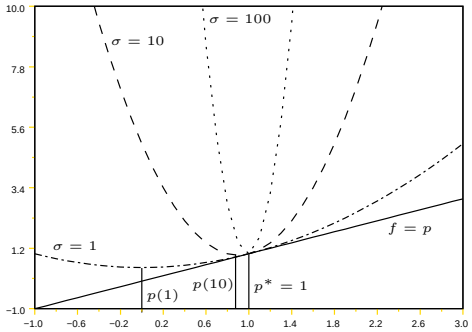

Fig. 1. Courant penalty method [13]

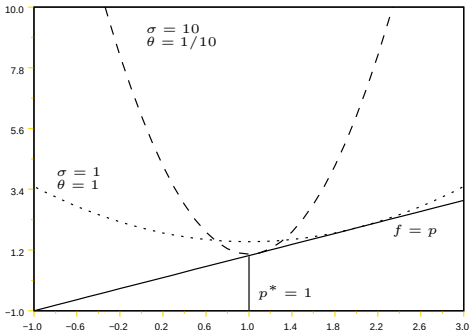

Fig. 2. Augmented Lagrangian method [13]

Figures 1 and 2 underline the differences between inexact and exact penalty functions for a simple problem: $\min_{p \in \mathbb{R}} p$ subject to $p - 1 = 0$. We can observe in figure 1 that the bigger σ is, the closer the minimum of the penalty function is to the solution of the original problem, but it never reaches it exactly, on the opposite to figure 2 which shows that the augmented Lagrangian has exactly the same minimum as the original problem, and for finite values of both θ and σ.

By using the augmented Lagrangian method, a constrained minimization problem is translated into a sequence of unconstrained problems (with $\lambda_i = \theta_i \sigma_i$):

(i)	$\lambda \leftarrow \lambda^{(1)}$, $\sigma \leftarrow \sigma^{(1)}$, $k \leftarrow 0$, $\|\nabla \Psi^{(0)}\|_\infty \leftarrow \infty$
(ii)	Find the minimizer $\mathbf{p}(\lambda, \sigma)$ of $\Phi(\mathbf{p}, \lambda, \sigma)$ and denote $\mathbf{c} = \mathbf{c}(\mathbf{p}(\lambda, \sigma))$
(iii)	If $\|[-\max(c_i, \frac{\lambda_i}{\sigma_i})]_{i=1..m}\|_\infty > \frac{1}{4}\|\nabla \Psi^{(k)}\|_\infty$ then : $\forall i$, if $\|c_i\| > \frac{1}{4}\|c^{(k)}\|_\infty$ then $\sigma_i \leftarrow 10\sigma_i$ go to step (ii)
(iv)	$k \leftarrow k+1$, $\lambda^{(\mathbf{k})} \leftarrow \lambda$, $\sigma^{(\mathbf{k})} \leftarrow \sigma$, $\mathbf{c}^{(\mathbf{k})} \leftarrow \mathbf{c}$
(v)	$\forall i = 1..m, \lambda_i \leftarrow \lambda_i^{(k)} - \max(\sigma_i c_i^{(k)}, \lambda_i^k)$ and $\|\nabla \Psi^{(k)}\|_\infty \leftarrow \|[-\max(c_i, \frac{\lambda_i}{\sigma_i})]_{i=1..m}\|_\infty$

The step (ii) is the non trivial step of the algorithm and can be solved by using a derivative free algorithm like NEWUOA. This algorithm has several advantages :

- derivatives of the cost function and the constraints never appear in this scheme,
- this is an exact penalty method
- the usual ill conditionning of penalty methods when $\sigma \to \infty$ doesn't appear here since the σ_i and θ_i are finite numbers.

Since we don't use derivatives, the convergence of step (ii) of the algorithm can take a long time, and the convergence of the whole algorithm can really be much longer than inexact penalty methods. Moreover, this algorithm doesn't guarantee that the constraints won't be violated during the iterations (this is not an interior point algorihtm). These are the two major disavantages of our approach. We will be able to quantify the impact of these disavantages on our on-line trajectory optimizer during the experiments with industrial applications in the last section.

IV. Experiments on industrial robot applications

The main purpose of our work is to optimize the duration of industrial applications without exceeding a maximum authorized temperature and maximum authorized torques of each actuator of a Stäubli Rx90 with a CS8 controller. That is why $L(.)$ of the problem (1) is the duration of the application, and the $c_i(.)$ represent then the temperature and torque limitations of a Stäubli Rx90 (Torques are computed from motor currents and the temperature from motor currents and velocities, using a thermal model). Let us summarize the structure of our on-line trajectory optimizer in figure 3 before testing it on industrial applications.

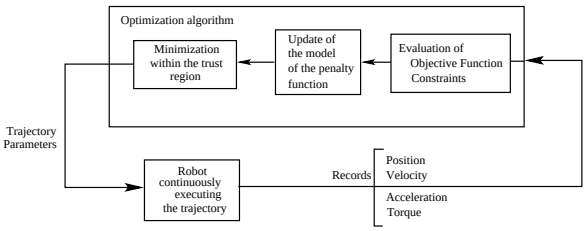

Fig. 3. Algorithm overview

The previous sections dealt with the application of non-linear optimization theory without derivatives to the on-line

optimization of robot trajectories. We are going to describe now typical industrial applications on which we will test this method in order to test its convergence properties and its robustness with respect to a high number of parameters and changes in the robot task.

A. Description of typical industrial applications

First, we will describe two typical industrial applications on which we are going to use our algorithm to minimize the duration of the application cycle (the cycle time) subject to maximum torque and temperature constraints. These two applications are:

- a pick and place application illustrated in figure 4
- a load and unload of a machine tool and palettization application illustrated in figure 5

The differences between the two applications are:

- the first one has a limited number of trajectory parameters, 12, whereas the second one has a high number of trajectory parameters, 54,
- the active constraints aren't the same in the two applications. In the first application, the temperature constraints are actually active, and only the torque constraints are active in the second industrial application.

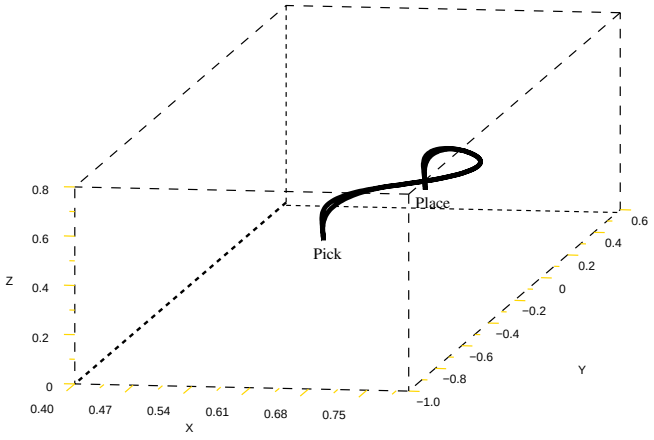

Fig. 4. Real geometric trajectory of the robot tool during the pick and place application

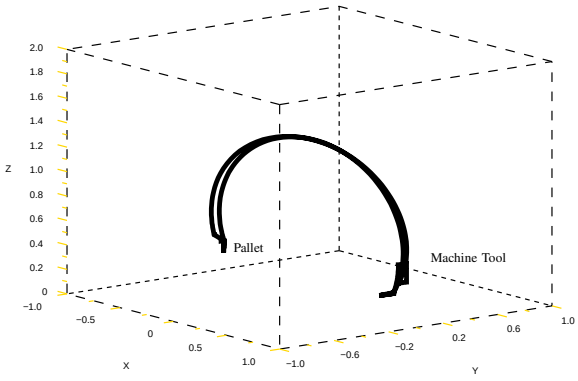

Fig. 5. Real geometric trajectory of the robot tool during the load/unload application

These experiments are designed to test the following points:

- comparison between an inexact penalty function, the exponential penalty function (5), and an exact penalty function, the augmented Lagrangian (7),
- influence of the number of trajectory parameters on the speed of convergence,
- robustness to task changes.

B. Results

The first experiment concerns the optimization of the pick and place application without load and deals with the comparison of an exponential penalty function and the augmented Lagrangian method. The weighting coefficients of the penalty functions has been initialized as in table I.

Exponential penalty function	Augmented Lagrangian penalty function
$\sigma^1 = [50, ..., 50]^T$	$\sigma^{(1)} = [100, ..., 100]$ $\lambda^{(1)} = [0, ..., 0]$

TABLE I
PENALTY COEFFICIENTS INITIALIZATION

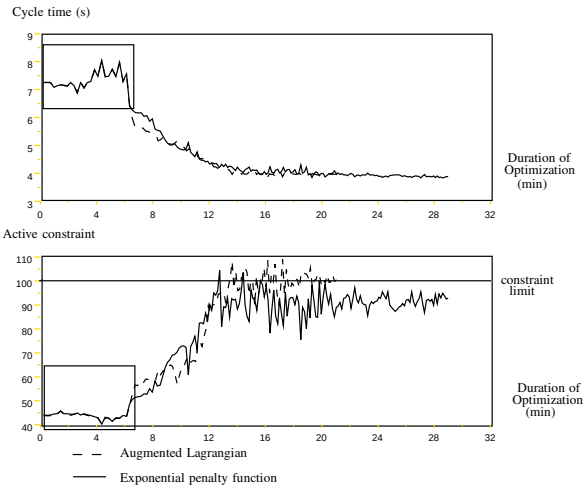

Fig. 6. Convergence of the algorithm for the pick and place application without load

Figure 6 shows an identical evolution of the two experiments in the beginning (in the boxed area), this is due to the NEWUOA software which always uses the same parameter variations to initialize the quadratic approximation of the cost function. After this initialization, we can observe a decrease of the cycle time while the constraints rise up to their limit, and the algorithm converges to a solution. We can also see that the augmented Lagrangian method converges to a solution where the value of the constraint is closer to the limit than with the exponential penalty function, demonstrating the difference between exact and inexact penalty methods. We can observe also that there is a difference of 5% in the value of the active constraint between the two methods, but the two cycle times are equivalent. This may lead to the conclusion that the active constraint is actually more sentive to parameter variations than

the cycle time. Note also that since we don't use interior point methods, the constraints can be temporarily violated during the convergence process, as can be seen in figures 6 and 7: this is an important characteristic of this method, not to be forgotten.

In a second experiment, the robot must execute a pick and place application and carry a load of 6 Kg: without changing anything in the algorithm, it converges to a solution (showed in figure 7) different from the one of the first experience: the algorithm automaticaly reduces the speed of the robot thanks to the use of data directly recorded by sensors on the robot.

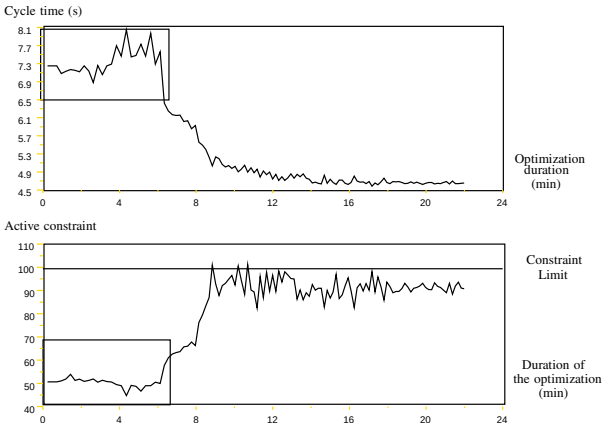

Fig. 7. Convergence of the algorithm for the pick and place application with a load of 6Kg

The last experiment consists in testing the capacity of the algorithm to converge with a more complex task. To test this property, we apply our algorithm on the load/unload application (54 trajectory parameters compared to 12 for the pick and place application). The optimization finishes after 5 hours, compared to 30 minutes for the pick and place application.

These results can be summarized in two arrays. The first one describes the gain in cycle time compared to the original constructor settings:

Application	Nominal cycle time	After Optimization.	Gain
Pick & Place	6.52s	3.89s	**40%**
Pick & Place with load	6.52s	4.66s	**28%**
Load/unload	8.12s	7.44s	**8.4%**

The second one shows the influence of the number of trajectory parameters on the duration of the optimization:

Application	Nunber of trajectory parameters	Optimization duration
Pick & Place	12	30 min
Pick & Place with load	12	30 min
Load/unload	54	5 h

From an industrial point of view, 5 hours is not a long time for the optimization of a trajectory which is going to be executed repeatedly for months or years, and an application with 54 trajectory parameters corresponds to a really large application. Our algorithm appears therefore to be well adapted to the optimization of industrial applications.

V. Conclusion

The optimization of trajectories is a typical problem in robotics research. But since robots are often integrated in complex robot cells and must interact with them, derivatives of the cost function and the constraints generally don't exist or can't be calculated efficiently, reason why tools for optimization without derivatives must be used. In a first section, we described how we could solve an unconstrained optimization problem without derivatives efficiently. Then we saw how to take into account the constraints by translating the constrained problem into a sequence of unconstrained problems by using penalty methods. In the last part, we tested our algorithm on two real industrial applications in order to minimize the cycle time of the applications taking into account maximum authorized temperature and maximum authorized torque on each actuator. Our algorithm is robust to task changes and to the number of trajectory parameters. On the other hand, it is difficult to conclude on the best choice of the penalty function what may strongly depend on the nature of the optimization problem. The experiments presented here showed for example that exponential penalty methods seem to be good enough in that specific case.

Our optimizer has been used on industrial applications, and it allowed to reach a high level of efficiency. More than an optimizer of industrial applications, this algorithm can be seen as a learning program. This approach could be applied then in artificial intelligence, on autonomous mobile robots or simply on difficult optimization problems where genetic algorithms are usually used today. We hope this approach might prove to be more efficient.

References

[1] J. M. Hollerbach, "Dynamic scaling of manipulator trajectories," *Journal of Dynamic Systems, Measurement, and Control*, 1984.
[2] Y. Bestaoui, "On line reference trajectory definition with joint torque and velocity constraints," *The International Journal of Robotics Research*, 1992.
[3] A. D. Luca, L. Lanari, and G. Oriolo, "A sensitivity approach to optimal spline robot trajectories," *Automatica*, 1991.
[4] J. E. Bobrow, S. Dubowsky, and J. Gibson, "Time optimal control of robotic manipulators along specified paths," *The International Journal of Robotics Research*, 1985.
[5] Y. Zhang and H. Munch, "Optimal motion planning for industrial robots," *25th ISIR*, 1994.
[6] R. Horowitz, "Learning control of robot manipulators," *ASME Journal of Dynamic Systems Measurement and Control 50th Anniversary Issue*, 1993.
[7] M. Powell, "Direct search algorithms for optimization calculations," *Acta Numerica, Vol. 7, Cambridge University Press*, 1998.
[8] A. Conn, K. Scheinberg, and P. Toint, "Recent progress in unconstrained nonlinear optimization without derivatives," *ISMP97, Lausanne*, 1997.
[9] M. Powell, "An efficient method for finding the minimum of a function of several variables without calculating derivatives," *Computer Journal*, 1964.
[10] D. Winfield, "Function minimization by interpolation in a data table," *Journal of the Institute of Mathematics and its applications*, 1973.
[11] M. Powell, "The newuoa software for unconstrained optimization without derivatives," *40th Workshop on Large Scale Nonlinear Optimization (Erice, Italy)*, 2004.
[12] ——, "Least frobenius norm updating of quadratic models that satisfy interpolation conditions," *Mathematical Programming, vol. 100*, 2004.

[13] R. Fletcher, *Practical Methods of Optimization, Second Edition*. John Wiley and Sons, 1987.
[14] J. F. Bonnans, J.-C. Gilbert, C. Lemaréchal, and C. Sagastizabal, *Numerical Optimization : Theoritical and Practical Aspects*, 2003.
[15] J.-B. Hiriart-Hurruty, *Optimisation.* Que sais-je ?, 2001.
[16] M. Powell, "A method for nonlinear constraints in minimization problems," *Optimization (Symposium, Univ. Keele) pp. 283-298*, 1969.