Optimal Robot Control Using Modelica and Optimica

Martin Hast[a] Johan Åkesson[a,b] Anders Robertsson[a]
[a)] Department of Automatic Control, LTH,
Lund University, Sweden
[b)] Modelon AB, Sweden

Abstract

In this paper, Modelica along with Optimica is used to formulate and solve a minimum time optimization problem. The problem concerns the traversal of a given path with a robot in as short time as possible under input constraints. Several problem reformulations, increasing the chance of finding optimal solutions, are discussed. This paper also discusses the use of these optimal solutions for control of industrial robots. A control structure, in which the optimal trajectories are essential, is used on an ABB IRB140B to ensure robustness for model errors and disturbances.

Keywords: Modelica, Optimica, Optimization, Robot Control

1 Introduction

In several robot applications such as gluing, painting and arc welding, not only the end points but also the path as such and the speed of traversal are strongly connected to quality and efficiency. Optimal input trajectories, minimizing the traversal time along a path, can be found by solving an optimization problem. The results can be used by a controller to slow down the motion along the path rather than deviate from it, if the robot is subject to disturbances or model errors. A controller with ability to slow down the motion along the path while still trying to minimize the traversal time is a Path Velocity Controller, PVC [7]. The PVC depends on the existence of nominal acceleration and velocity profiles. The profiles are obtained by formulating and solving a minimum time optimization problem. Modelica along with Optimica [2] can be used to formulate optimization problems in a natural and compact way. Modelica is used to formulate the dynamical system and the initial values for the optimization problem, while the Optimica language extension is used to impose limits on the variables and to formulate the cost function and the constraints. Hence, Modelica and Optimica provide a convenient method for formulating and solving optimal control problems which is necessary when using PVC.

2 Background

2.1 JModelica.org

JModelica.org is a novel Modelica-based open source project targeted at dynamic optimization [4]. JModelica.org features compilers supporting code generation of Modelica models to C, a C API for evaluating model equations and their derivatives and optimization algorithms. The compilers and the model C API has also been interfaced with Python [9] in order to enable scripting and custom application development. In order to support formulation of dynamic optimization of Modelica models, JModelica.org supports the Optimica extension [3]. Optimica offers constructs for encoding of cost functions, constraints, the optimization interval with fixed or free end points as well as specification of the transcription scheme.

The JModelica.org platform contains an implementation of a simultaneous optimization method based on orthogonal collocation on finite elements [6]. Using this method, state and input profiles are parametrized by Lagrange polynomials, of order three and four respectively, based on Radau points. This method corresponds to a fully implicit Runge-Kutta method, and accordingly it possesses well known and strong stability properties. By parametrizing the variable profiles by polynomials, the dynamic optimization problem is translated into a non-linear program (NLP), solved by a numerical NLP solver. The NLP is, however, very large. In order to efficiently find a solution to the NLP, derivative information as well as the sparsity patterns of the constraint Jacobians need to be provided to the solver. The simultaneous optimization algorithm has been interfaced with the large-scale NLP solver IPOPT [13], which has been developed particularly

DOI: 10.3384/ecp09430089

to solve NLPs, arising in simultaneous dynamic optimization.

The choice of a simultaneous optimization algorithm fits well with the properties of the dynamic optimization problems treated in this paper. In particular, simultaneous methods handle unstable systems well, and also, state and input inequality constraints are easily incorporated.

Formulating the Minimum Time Optimization Problem

We consider the optimization problem of traversing a given path in as short time as possible under given input constraints. The minimum time optimization problem and reformulations thereof follow the presentation in [7]. Subsequently, we assume that a model of order p with states q_i and n inputs, τ_i, is available.

A model of a rigid robot can be written

$$\tau = H(q)\ddot{q} + v(q,\dot{q}) + d(q)\dot{q} + g(q), \tag{1}$$

whereas a model of a flexible robot is given by

$$\begin{aligned} &H(q)\ddot{q} + v(q,\dot{q} + d(q)\dot{q} + g(q) + K(q-\theta) = 0 \\ &J\ddot{\theta} = \tau + K(q-\theta). \end{aligned} \tag{2}$$

See [12] for details concerning robot modelling. Common for these models, and for the models used in work described in this paper, are that they all can be written on the form

$$\tau = h(q,\dot{q},\ldots,q^{(p)}) \tag{3}$$

with limited inputs described by

$$\tau_i^{min} \le \tau_i \le \tau_i^{max}, \qquad 1 \le i \le n \tag{4}$$

We further assume that a traversable path,

$$q(t) = f(t) \tag{5}$$

is available, i.e., the path trajectories are defined in such a way that all states can reach all points on the prescribed path.

The objective is to traverse the path as fast as possible, i.e, minimizing the traversal time t_f. By setting the start time, $t_0 = 0$, the time-minimum optimization problem is formulated as

$$\min_{\tau} t_f = \min_{\tau} \int_0^{t_f} 1 \mathrm{d}t \tag{6}$$

under the constraints imposed by (4) and (5), along with boundary conditions for

$$\begin{aligned} &q(t_0),\dot{q}(t_0),\ldots,q^{(p)}(t_0) \\ &q(t_f),\dot{q}(t_f),\ldots,q^{(p)}(t_f) \end{aligned} \tag{7}$$

The problem formulation consists of pn states and is generally hard to solve. Consequently a reduction of the number of states is desirable. The reductions are conducted as presented in [7] and are here given for completeness.

Reducing the Number of States

The number of dynamical states in the optimization problem is reduced to p as described in [7]. By introducing the *path parameter*

$$s(t_0) = s_0 \le s(t) \le s(t_f) = s_f \tag{8}$$

and parametrizing the path f as a function of the nominal, scalar path parameter i.e., $f(s)$, the number of states can be reduced. Setting $q = f(s)$ and using the chain rule, $\frac{df}{dt} = \frac{df}{ds}\frac{ds}{dt}$, the model given by (3) is rewritten as

$$\tau = h_s(s,\dot{s},\ldots,s^{(p)}) \tag{9}$$

In addition to (4), (9) serves as constraints for the reduced optimization problem. The dynamics of the optimization problem are now expressed as a chain of p integrators

$$\begin{aligned} &\frac{ds}{dt} = \dot{s} \\ &\frac{d\dot{s}}{dt} = \ddot{s} \\ &\vdots \\ &\frac{ds^{(p-1)}}{dt} = s^{(p)} \end{aligned} \tag{10}$$

Following the state reduction the cost function is expressed as

$$\min_{s^{(p)}} \int_0^{t_f} 1 \mathrm{d}t \tag{11}$$

and the boundary conditions are imposed on s and its time derivatives

$$\begin{aligned} &s(t_0),\ldots,s^{(p-1)}(t_0) \\ &s(t_f),\ldots,s^{(p-1)}(t_f) \end{aligned} \tag{12}$$

Reformulating the Optimization Problem

The number of states in the optimization problem can be further reduced to $p-1$ as the problem is reformulated over a fixed interval. The problem is converted to optimization over a fixed interval by deriving a dynamic system in s [7]. New states $x_1,\ldots,x_{p-1}$ are in-

troduced. They are defined as

$$\begin{aligned} x_1 &= \frac{\dot{s}^p}{p} \\ x_i &= \frac{dx_{i-1}}{ds}, \qquad i = 2,\ldots,p-1 \end{aligned} \tag{13}$$

The free variable, u, in the optimization problem is defined as $u = s^{(p)}$. By defining a function, g, as

$$g(x_1) = (px_1)^{1/p} \tag{14}$$

the p-th order derivative of s is expressed as [7]

$$s^{(p)} = g'(x_1)g(x_1)^{(p-1)}\frac{dx_{p-1}}{ds} + F_p(x_1,\ldots,x_{p-1}) \tag{15}$$

where

$$\begin{aligned} F_p(x_1,\ldots,x_{p-1}) =& g''(x_1)x_2 g(x_1)g(x_1)^{p-2}x_{p-1} \\ &+ g'(x_1)(p-2)g(x_1)^{p-3}g'(x_1)x_2 g(x_1)x_{p-1} \\ &+ \sum_{i=1}^{p-2}\frac{\partial F_{p-1}(x_1,\ldots,x_{p-2})}{\partial x_i}x_{i+1}g(x_1) \end{aligned} \tag{16}$$

The constraints (9) are written as functions of s, x and u as

$$\tau = h_x(s,x_1,\ldots,x_{p-1},u) \tag{17}$$

still subject to (4). We now have boundary conditions for

$$\begin{aligned} &x_1(s_0),\ldots,x_{p-1}(s_0) \\ &x_1(s_f),\ldots,x_{p-1}(s_f) \end{aligned} \tag{18}$$

The cost function for the minimum time optimization problem is reformulated

$$\int_0^{t_f} \mathrm{d}t = \int_{s_0}^{s_f}\frac{1}{\dot{s}}\mathrm{d}s = \int_{s_0}^{s_f}(px_1)^{-1/p}\mathrm{d}s \tag{19}$$

Equations (17), (18) and (19) along with the dynamic system

$$\begin{aligned} \frac{dx_1}{ds} &= x_2 \\ \frac{dx_2}{ds} &= x_3 \\ &\vdots \\ \frac{dx_{p-2}}{ds} &= x_{p-1} \\ \frac{dx_{p-1}}{ds} &= u - F_p(x_1,\ldots,x_{p-1}) \end{aligned} \tag{20}$$

state the full optimization problem, reformulated to a fixed interval.

Figure 1: An ABB IRB140B. Picture from [1].

3 Modeling

The robot considered in this paper is an ABB IRB-140B, see Figure 1. The IRB140B is a six joint serial robot which allows an arbitrary positioning and orientation within the robot's work envelope. Due to the mechanical structure of robots, depicted in Figure 2, robot models in general are quite complex and non-linear, cf., Model (1) and (2). However, the robot model used here consists of six independent linear models. Each of the robot's six joints are modeled by a linear second order model. This is possible since the input-output relation for which the model has been identified contains a linearizing controller. The identified models, and the models used for optimization, describe the relation between a joint's velocity reference, τ, and its angular position, q. The model structure is given by (21).

$$T_i\ddot{q}_i + \dot{q}_i = K_i\tau_i \tag{21}$$

The parameter values for each joint are displayed in Table 1.

4 Path

A path has been recorded using so called *lead-through*, a force-control mode which allows the operator to freely move/lead the robot in the workspace, with the IRB140B [1]. The joint angles have been parametrized

Joint	K_i	$T_i \times 10^{-2}$
1	1.031	1.907
2	1.077	2.043
3	1.061	1.913
4	1.051	1.716
5	1.062	1.791
6	1.062	1.745

Table 1: Parameters for the controlled joint model of Eq. (21).

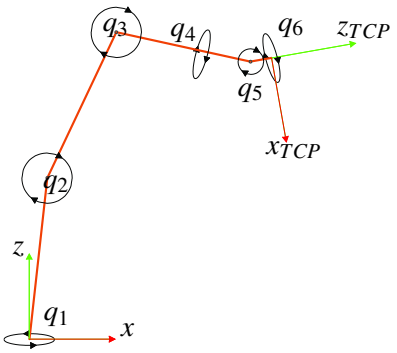

Figure 2: Mechanical structure of the ABB IRB140B showing the joint angles q_i. Picture from [8].

by the path parameter, s, defined between $s_0 = 0$ and $s_f = 1$. The path is described by splines implemented in Modelica as `if`-clauses and the derivatives of $f(s)$ have been calculated by derivation of the splines.

Figure 3 shows the TCP position. TCP is an abbreviation for Tool Center Point which is a user-defined point on the robot's end effector.

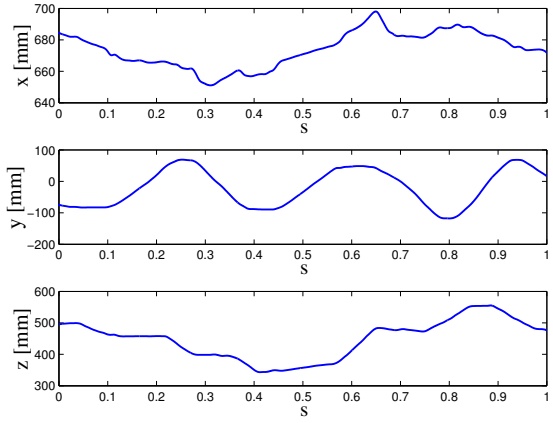

Figure 3: TCP as function of the path parameter.

5 Optimization

The goal of the optimization is to find the minimum time acceleration profile, $v_2(s)$, and the velocity profile, $v_1(s)$, to be used in the PVC, see Section 6. The acceleration profile is defined as the acceleration along the path expressed as a function of the nominal path parameter i.e., $v_2(s) = \ddot{s}(s)$. The velocity profile is defined analogous by i.e., $v_1(s) = \dot{s}(s)$. This can be done if the time derivative of s, $\dot{s}$, is assumed greater or equal to zero [7]. Formulation of the optimization problem is done using Modelica and Optimica. Modelica is used to code the constraints (17) and the dynamics (20), while Optimica is used to define the input limits, (4) the boundary conditions, (18), and the cost function, (19). The Modelica and Optimica codes are presented in Appendix A, Listings 1 and 2. Note that the built-in Modelica variable `time` is equivalent to the path parameter s due to the reformulation. Hence, the `der()`-operator is equivalent to derivation with respect to the path parameter s.

Using the reduction techniques from Section 2 the reformulated optimization problem considered is now given by

$$
\begin{aligned}
&\min_{u(s)} \int_{s_0}^{s_f} \frac{1}{\sqrt{2x_1}} \mathrm{d}s \\
&\text{s.t.} \\
&\quad x_1 = \frac{\dot{s}^2}{2}, \quad \frac{dx_1}{ds} = u \\
&\quad K\tau = T(f''(s)\dot{s}^2 + f'(s)u) \\
&\quad \tau^{min} \leq \tau \leq \tau^{max} \\
&\quad x_1(s_0) = 0, \quad x_1(s_f) = 0 \\
&\quad \dot{s} \geq 0
\end{aligned}
\tag{22}
$$

This work was done using a predecessor of the JModelica.org platform, see [11] and [4]. The predecessor version uses AMPL as intermediate representation format and supports an early version of Optimica. The Modelica and Optimica code is compiled by the Optimica compiler which translates the optimization problem into AMPL [5]. The external solver, IPOPT [10], is then called to solve the problem.

In order for IPOPT to find an optimal solution the occurrence of a good initial guess is crucial. Because of the free end time, finding an optimal solution for a general minimum time problem requires an initial guess close to the optimal solution. The reformulation of the optimization problem to a fixed interval is therefore preferable.

Finding an optimal solution for the optimization problem (22) turns out to be feasible. But in order to find a solution with a smooth acceleration profile the optimization problem has to be solved in two steps. First, the optimization problem as it is stated in (22) is

solved. This result can now be used as an initial guess when solving a second optimization problem. In the second optimization problem a two per cent slack on the final time is introduced and the cost function penalizes the square integral of the input signals. This renders smoother acceleration and velocity trajectories which are suitable for implementation on the robot actuators while the robot still traverses the path in close to minimum time.

6 Control System

The PVC algorithm [7] modifies the acceleration of a new path parameter, σ, in such a way that the path, $f(\sigma)$, is not deviated from while ensuring that the input limitations (4) are not violated. The algorithm uses the nominal acceleration profile, $v_2(\sigma)$, as a reference to update the path acceleration, $\ddot{\sigma}$. The path acceleration is limited to make sure that the input constraints (4) are not violated. Moreover, the algorithm includes internal feedback $\frac{\alpha}{2}(v_1(\sigma)^2 - \dot{\sigma}^2)$ that makes the path velocity, $\dot{\sigma}$, approach the nominal velocity, $v_1(\sigma)$.

A controller written on the form

$$\tau = \beta_1(\sigma)\ddot{\sigma} + \beta_2(\sigma, \dot{\sigma}, q, \dot{q}) \tag{23}$$

is assumed to be available. Combining (23) with the limits (4), it is possible to calculate the minimum and the maximum acceleration, $\ddot{\sigma}^i_{min}$ and $\ddot{\sigma}^i_{max}$, for each joint i.

$$\tau_i^{min} \leq \tau_i = \beta_{1_i}\ddot{\sigma} + \beta_{2_i} \leq \tau_i^{max}, \qquad 1 \leq i \leq 6 \tag{24}$$

$$\begin{aligned}
\ddot{\sigma}_i^{max}(\beta_{1_i}\beta_{2_i}) &= \begin{cases} \frac{\tau_i^{max} - \beta_{2_i}}{\beta_{1_i}} & \beta_{1_i} > 0 \\ \frac{\tau_i^{min} - \beta_{2_i}}{\beta_{1_i}} & \beta_{1_i} < 0 \\ \infty, & \beta_{1_i} = 0 \end{cases} \\
\ddot{\sigma}_i^{min}(\beta_{1_i}\beta_{2_i}) &= \begin{cases} \frac{\tau_i^{min} - \beta_{2_i}}{\beta_{1_i}} & \beta_{1_i} > 0 \\ \frac{\tau_i^{max} - \beta_{2_i}}{\beta_{1_i}} & \beta_{1_i} < 0 \\ -\infty, & \beta_{1_i} = 0 \end{cases}
\end{aligned} \tag{25}$$

By choosing the limits on $\ddot{\sigma}$ according to (26), the acceleration along the path is chosen in order not to violate the limits on the input

$$\begin{aligned}
\ddot{\sigma}_{max}(\beta_1, \beta_2) &= \min_i \ddot{\sigma}_i^{max}(\beta_1, \beta_2) \\
\ddot{\sigma}_{min}(\beta_1, \beta_2) &= \max_i \ddot{\sigma}_i^{max}(\beta_1, \beta_2)
\end{aligned} \tag{26}$$

The PVC algorithm is given by (27). The full control structure is presented in the block diagram in Figure 4.

$$\begin{aligned}
\frac{d\sigma}{dt} &= \dot{\sigma} \\
\frac{d\dot{\sigma}}{dt} &= \ddot{\sigma} \\
u_r &= v_2(\sigma) + \frac{\alpha}{2}(v_1(\sigma)^2 - \dot{\sigma}^2) \\
\ddot{\sigma} &= sat(u_r, \ddot{\sigma}_{min}(\beta_1, \beta_2), \sigma_{max}(\beta_1, \beta_2))
\end{aligned} \tag{27}$$

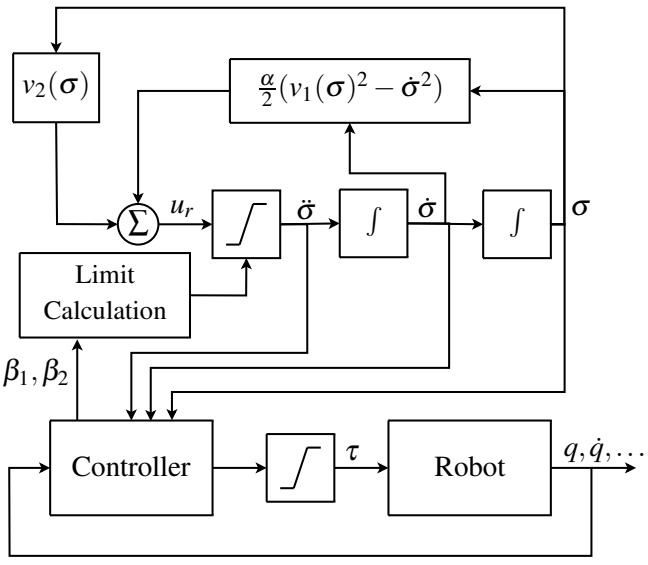

Figure 4: The PVC algorithm with controller and robot

7 Simulations

The control algorithm, (27), described in Section 6 has been implemented in Simulink. In order to evaluate the performance of the PVC a model error was introduced. The model error introduced was a 20% decrease in the gain for joint 1, i.e., $\tilde{K}_1 = 0.8K_1$. Two simulations were done using the perturbed model. In the first simulation, the regular controller 23 was used but the PVC was disabled, see Figure 5, whereas in the second simulation the PVC was used together with the controller 23. The internal feedback gain in the second simulation was chosen to $\alpha = 500$.

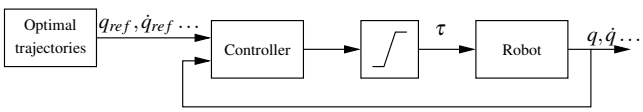

Figure 5: Block diagram showing the setup used in the first simulation, without the PVC.

The minimum time for traversing the path with the unperturbed model is 10.87 seconds and with the perturbation the minimum time is 12.09 seconds.

In the first simulation the path was traversed with the velocity profile obtained from the optimization. Since there was no PVC the deviated path was traversed in 10.87 seconds. In the second simulation, with the PVC, the path was instead traversed in 12.41 seconds which is 0.32 seconds longer than the optimal time for the perturbed model. The velocity along the path, $\dot{\sigma}(\sigma)$, for both simulations is displayed in Figure 6. Here one can clearly see that the PVC lowers the velocity along the path. When using the PVC, the path is traversed without violating the input boundaries, τ^{min} and τ^{max}, as can be seen in Figure 8. When not using the PVC, see Figure 9, it is obvious that the input signal calculated by the controller is well above the input limitations of the process. Since the input signals are saturated at their limits this gives rise to the deviation from the path visible in Figure 7. Figure 7 shows that perturbation mostly effects the error in y direction which is due the robot's mechanical structure and and the traversed path.

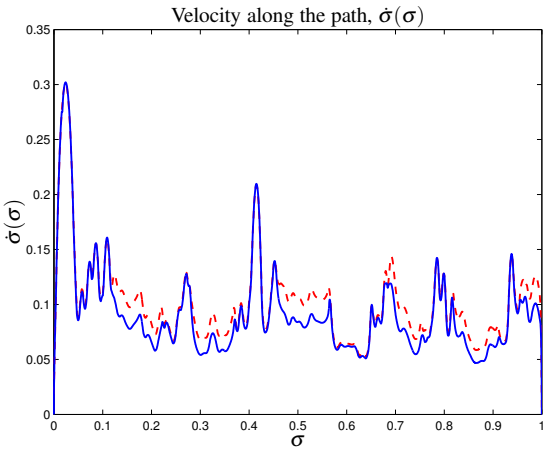

Figure 6: Continuous line is with the PVC enabled and dashed line is with the PVC disabled

8 Summary and Conclusions

In this paper we formulated and solved a minimum time optimization problem for an industrial robot. Reformulations were done in order to obtain acceleration and velocity profiles without having to find an initial guess close to the optimal solution. The robot joints were modeled with simple second order linear transfer functions. This was possible due to the presence of linearizing controllers working within the identified models. Lead-through was used to record the path which

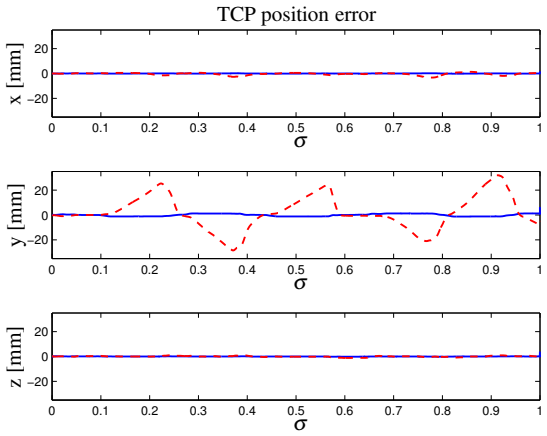

Figure 7: Continuous line is with the PVC enabled and dashed line is with the PVC disabled.

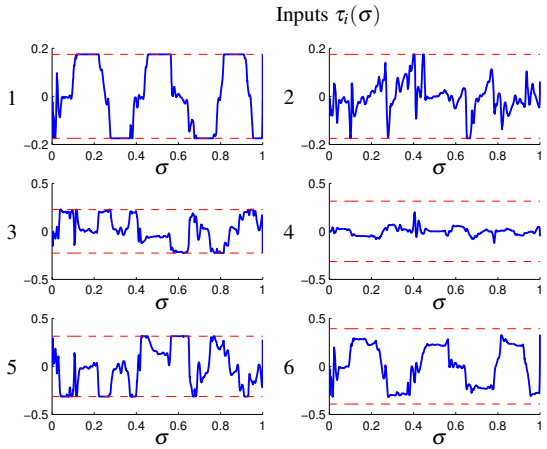

Figure 8: Input signals for the six joints with the PVC enabled

was represented by the joints angular positions. The dynamical model with its constraints and boundary conditions, the path and the cost function were done in Modelica and Optimica. Modelica along with Optimica provided an efficient and convenient way to formulate the dynamic optimization problem. The Optimica formulation is both in structure and syntax close to the mathematical description of the optimization problem which is beneficial. The closeness to the mathematical description facilitates the formulation of optimization problems which makes the work less time consuming as well as less error-prone than coding in for instance AMPL.

The optimization results were used as nominal acceleration and velocity trajectories in a PVC. The PVC has been tested, both in simulations and on an ABB IRB140B robot, showing that the path deviation is small, the input limitations are not violated and that the path traversal speed is close to the optimal. This paper shows that Optimica is well suited for the task

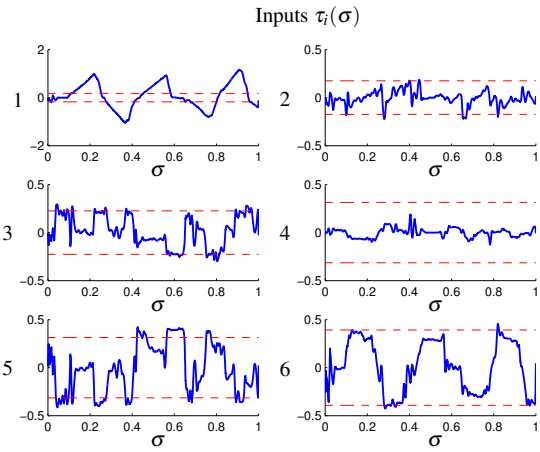

Figure 9: Input signals for the six joints with the PVC disabled

of finding the optimal acceleration and velocity profiles needed in order to use PVC.

References

[1] ABB. ABB Home Page, 2009. `http://www.abb.com/`.

[2] Johan Åkesson. *Tools and Languages for Optimization of Large-Scale Systems*. PhD thesis, Department of Automatic Control, Lund University, Sweden, November 2007. ISRN LUTFD2/TFRT--1081--SE.

[3] Johan Åkesson. Optimica—an extension of modelica supporting dynamic optimization. In *In 6th International Modelica Conference 2008*. Modelica Association, March 2008.

[4] Johan Åkesson, Tove Bergdahl, Magnus Gäfvert, and Hubertus Tummescheit. Modeling and Optimization with Modelica and Optimica Using the JModelica.org Open Source Platform. In *Proceedings of the 7th International Modelica Conference 2009*. Modelica Association, September 2009.

[5] AMPL - A Modeling Language for Mathematical Programming. AMPL Home Page, 2009. `http://www.ampl.com/`.

[6] L.T. Biegler, A.M. Cervantes, and A Wächter. Advances in simultaneous strategies for dynamic optimization. *Chemical Engineering Science*, 57:575–593, 2002.

[7] Ola Dahl. *Path Constrained Robot Control*. PhD thesis, Department of Automatic Control, Lund University, Sweden, April 1992. ISRN LUTFD2/TFRT--1038--SE.

[8] Fredrik Eriksson and Marcus Welander. Haptic interface for a contact force controlled robot. Master's Thesis ISRN LUTFD2/TFRT--5837--SE, Department of Automatic Control, Lund University, Sweden, May 2009.

[9] Python Software Foundation. Python Programming Language – Official Website, 2009. `http://www.python.org/`.

[10] IPOPT - Interior Point OPTimizer. IPOPT Home Page, 2009. `https://projects.coin-or.org/Ipopt`.

[11] Modelon AB. JModelica Home Page, 2009. `http://www.jmodelica.org`.

[12] Mark W. Spong, Seth Hutchinson, and M.Vidyasagar. *Robot Modeling and Control*. John Wiley & Sons, Inc, 2006.

[13] Andreas Wächter and Lorenz T. Biegler. On the implementation of an interior-point filter line- search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–58, 2006.

A Modelica and Optimica Code

```
model Robot
  // Robot inputs, tau
  Real tau[6];
  // Robot model gains, K
  parameter Real K[6] = {1.031, 1.077, 1.061,
    1.051, 1.054, 1.062};
  // Robot model time constants, T
  parameter Real T[6] = {0.019086, 0.020433,
    0.019129, 0.017158, 0.017909, 0.017447};
  // First and second derivative of s
  // with respect to time
  Real sd(start=0);
  Real sdd;
  // Auxiliary variable
  Real x1;
  // Optimization variable
  Modelica.Blocks.Interfaces.RealInput u
  // Path, f, stored as splines
  Splines f;
  // First and second derivative of the path f
  // with respect to s
  Real df[6];
  Real ddf[6];
equation
  K*tau = T*(ddf*sd^2 + df*u) + df*sd;
  df = der(f.f);
  ddf = der(df);
  x1 = sd^2/2;
  der(x1) = u;
  // Independant variable
  f.s = time;
end Robot;
```

Listing 1: The Modelica code for the optimization problem in (22).

```
optimization OptTraj (objective=cost,
                      startTime=0,
                      finalTime=1)
  // Cost function
  Real cost;
  // Instance of robot model
  Robot robot(u(free=true));
equation
  der(cost) = 1/sqrt(2*x1+1e-10);
constraint
  // Lower bounds on robot inputs
  robot.tau >= {-0.175, -0.175, -0.227,
                -0.314, -0.314, -0.395};
  // Upper bounds on robot inputs
  robot.tau <= {0.175, 0.175, 0.227,
                0.314, 0.314, 0.395};
  // Terminal constraint on x1
  robot.x1(finalTime) = 0;
  // Inequality constraint
  robot.sd >= 0;
end optTraj;
```

Listing 2: The Optimica code for the optimization problem in (22).