CONSTRAINED PARAMETRIC DESIGN OF MOTION LAWS

R.GARZIERA, A.TASORA

Università degli Studi di Parma, Dipartimento di Ingegneria Industriale
Parco Area delle Scienze, 43100 Parma, Italy
`garziera@ied.unipr.it, tasora@ied.unipr.it,`

ABSTRACT

A new method to design motion laws is proposed, featuring the ability of imposing constraints over sequences of functions, each defined by parameters. User constraints may require, for example, continuity between segments, acceleration or position at given time values, and so on. The optimal value for function parameters is obtained applying a modified Newton-Lagrange process over a constrained optimization problem. The outer Newton loop adopts a linear solver which exploits the sparsity of the coefficient matrix, hence allowing high computational efficiency even in case of many constraints. The resulting method has been implemented into a custom software for multibody simulation, thus providing a powerful yet expandable way to design motion laws for mechatronic devices.

KEYWORDS: Motion laws, robots, parametric design, optimization.

1 INTRODUCTION

The development of automatic devices and robots often entangles the cumbersome process of designing the motion laws for the controlled degrees of freedom.

Many kinds of motion laws are currently adopted for this purpose (polynomial, cycloidal, etc.), usually joined into a sequence of multiple functions if complex trajectories are required [1].

When dealing with long sequences of motion laws, the many degrees of freedom offered by the parameters of the function segments could be subject to a variable number of user-defined constraints: simplest cases can be solved immediately by analytical methods, if constraints are as simple as imposed continuity between segments or total movement over a fixed time span.

However, dealing for example with more complex constraints (such as imposing acceleration or position at generic time values) requires a generic, non-heuristic approach to the problem. Here we discuss a method which can be used to impose arbitrary constraints to the motion laws.

In order to define sequences of functions, we adopted an object-oriented approach where each function segment is an instance of a specific class. Instancing and sequencing of functions can be done at compile-time (with C++ language) or at runtime (using Javascript interpreted language).

Finally, an interactive graphical user-interface has been developed so that constraints over motion laws can be placed easily.

2 THE PROBLEM

Let consider a motion law as a piecewise function of the independent variable t, that is a $y = f(t)$ function $f : t \in \mathbb{R}^+ \to y \in \mathbb{R}$.

Most often the shape of such function can be defined by a finite set of parameters (for example the position and weights of knots, in case $f(t)$ is a spline), whose value can be decided in order to achieve some specific goal. Hence all n function parameters can be collected into a $\mathbf{q}$ vector, with $\mathbf{q} \in \mathcal{Q}$, $\mathcal{Q} \subset \mathbb{R}^n$ that is

$$\mathbf{q} = \{q_1, q_2, ..., q_n\}^T . \qquad (1)$$

Note that in case of functions which depend on w sub-functions, such as in the case of sequences or operations between children functions, a vector of design parameters can still be defined, as a collection of the parameter vectors $\mathbf{q}_{Fi}^T$ of the w children functions:

$$\mathbf{q} = \left\{ \mathbf{q}_{F1}^T, \mathbf{q}_{F2}^T, ..., \mathbf{q}_{Fw}^T \right\}^T . \qquad (2)$$

In general, we have a $f : \mathcal{T}, \mathcal{Q} \to \mathcal{Y}$ mapping:

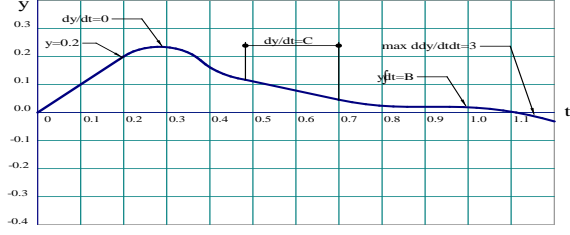$$y = f(t, \mathbf{q}) \qquad f : t \in \mathbb{R}, \mathbf{q} \in \mathbb{R}^n \to y \in \mathbb{R}. \qquad (3)$$

Figure 1: Motion law with user-imposed constraints.

Figure 2: A motion law made by a constrained sequence of basic segments.

This $f(t, \mathbf{q})$ function can be subject to m non-linear constraints, each with mapping $C_i : \mathcal{Q} \rightarrow \mathcal{C}_i$, which can be collected into a compact vectorial constraint:

$$\mathbf{C}(\mathbf{q}) = \mathbf{0} \qquad \mathbf{C} : \mathbf{q} \in \mathbb{R}^n \rightarrow \mathbf{C}(\mathbf{q}) \in \mathbb{R}^m, \qquad (4)$$

for $\mathbf{C} \in \mathcal{C}$. Different approaches for the solution of $\mathbf{q}$ will be discussed heretoafter.

2.1 Constraints

Depending on the features required to the motion law $y = f(t)$, appropriate constraint equations can be introduced in eq.4. For example, the function may have specific values of y at given instants of time t, as well as specific values of speed and acceleration, not necessarily at once (Figure 1).

Also, in most complex situations, constraints could put into relation either some parameters q and function values (or derivative, or integrals), for example one could require that the length of a function segment q_l is four times the derivative of $f(t)$ at $t = 2$, that is $q_l - 4\partial y/\partial t \mid_{t=2} = 0$, or one could require that two values at $t = 3, t = 6$ are spaced $10[mm]$ on y, that is $y \mid_{t=3} - y \mid_{t=6} -10 = 0$, and so on.

Following is a list of most useful constraints, whose meaning in a context of motion design is self-explaining.

- Prescribed value Y_a at given T_a time instant:
  $y|_{T_a} - Y_a = 0$

- Prescribed n-th derivative D_a at given T_a time instant:
  $(\partial^n y/\partial t^n)|_{T_a} - D_a = 0$

- Prescribed integral I_a at given T_a time instant:
  $(\int y dt)|_{T_a} - I_a = 0$

- Cn-continuity between segments:
  $(d^n y/dt^n)|_{T_a+} - (d^n y/dt^n)|_{T_a-} = 0$.

- Relation between parameters:
  $r(q_a, q_b, ...) = 0$
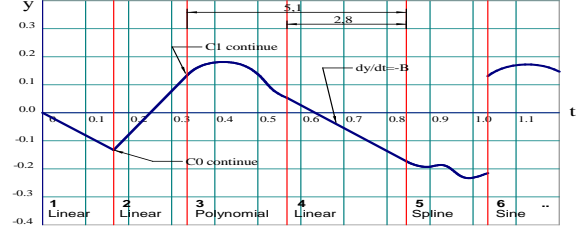
- Most generic constraint:
  $C(\mathbf{q}) = 0$

All the equations above can be written as $C_i(\mathbf{q}) = 0$, and these can be collected in a vector of equations, that is $\mathbf{C}(\mathbf{q}) = \mathbf{0}$ as needed in Equation 4.

2.2 Case of unique solution(s)

In the case $m = n$, where $\dim(\mathcal{C}) - \dim(\mathcal{Q}) = 0$, nonlinear equation 4 can be solved for a root $\mathbf{q} \in \mathcal{Q}$. This problem can be faced with the straightforward application of a Newton-Raphson method, which converges to a numerical approximation $\widetilde{\mathbf{q}}$ of the exact root $\overline{\mathbf{q}}$ after few iterations:

$$\Delta \mathbf{q}_{i+1} = -\left[\frac{\partial \mathbf{C}(\mathbf{q}_i)}{\partial \mathbf{q}}\right] \mathbf{C}(\mathbf{q}_i), \qquad (5)$$

$$\mathbf{q}_{i+1} = \mathbf{q}_i + \Delta \mathbf{q}_{i+1} \qquad (6)$$

where $[\partial \mathbf{C}/\partial \mathbf{q}]$ is the jacobian of $\mathbf{C}$ respect to the $\mathbf{q}$ parameter vector, and we will write it $[C_q]$ heretofore.

Of course convergence of the method depends on many factors, for example the initial guess $\mathbf{q}_0$ should be not too far from the root, $\mathbf{C}(\mathbf{q})$ must be first-order Fréchet differentiable in the surroundings of $\overline{\mathbf{q}}$, jacobian should be semidefinite and well conditioned, and so on.

However it often happens that the system has more degrees of freedom than the applied constraints. This is the case of $m < n$, that is $\dim(\mathcal{C}) < \dim(\mathcal{Q})$, and poses a different challenge. This is the case which we will deal with, in search of the highest generality.

2.3 Constrained minimization

Since an underconstrained problem leaves $n - m$ degrees of freedom, we can try to find the solution which minimizes some useful function, for example the least change from the initial state $\mathbf{q}_0$ given at the beginning of the Newton Raphson process (this can be an useful default behavior of the solver because each time the user tweaks a constraint, he would like to see the smallest change in other parts of the function under inspection).

Otherwise, introducing more freedom of control from the user point of view, the minimization could search the least value of some user-defined function, for example when one aims at least acceleration, least jerk and so on [2].

In either cases, one ends with a constrained minimization problem [3] of the type:

$$\min |_q F(\mathbf{q}) \tag{7}$$
$$\mathbf{C}(\mathbf{q}) = \mathbf{0} \tag{8}$$

where $F : \mathcal{Q} \to \mathcal{F}, \mathcal{F} \subset \mathbb{R}, \mathcal{Q} \subset \mathbb{R}^n$ is a generic objective function which should be minimized, $\mathbf{q} \in \mathcal{Q}$ are the optimization variables, and $\mathbf{C} : \mathcal{Q} \to \mathcal{C}$ are the constraints.

The Lagrangian,

$$\mathcal{L}(\mathbf{q}, \lambda) := F(\mathbf{q}) + \lambda^T \mathbf{C}(\mathbf{q}) \tag{9}$$

can be used to convert the constrained optimization problem into a system of nonlinear equations:

$$\left\{ \begin{array}{c} \partial \mathcal{L}/\partial \mathbf{q} \\ \partial \mathcal{L}/\partial \lambda \end{array} \right\} (\mathbf{q}, \lambda) = \mathbf{0}, \tag{10}$$

$$\left\{ \begin{array}{c} \partial F/\partial \mathbf{q} + [\partial \mathbf{C}/\partial \mathbf{q}]^T \lambda \\ \mathbf{C}(\mathbf{q}) \end{array} \right\} = \mathbf{0} \tag{11}$$

Equation 11 can be written in the compact form $\mathbf{G}(\mathbf{y}) = \mathbf{0}$, where $\mathbf{G}$ is the lagrangian gradient (a nonlinear function of variables $\mathbf{y} = \left\{ \mathbf{q}^T, \lambda^T \right\}^T$). The solution of the nonlinear problem $\mathbf{G}(\mathbf{y}) = \mathbf{0}$ may come from a Newton Raphson method, that is:

$$\Delta \mathbf{y}_{i+1} = - \left[\frac{\partial \mathbf{G}(\mathbf{y}_i)}{\partial \mathbf{y}} \right]^{-1} \mathbf{G}(\mathbf{y}_i), \tag{12}$$

$$\mathbf{y}_{i+1} = \mathbf{y}_i + \Delta \mathbf{y}_{i+1}. \tag{13}$$

The method may converge in few steps if a good initial estimate is provided, but the solution of the linear system in the step of equation 12 may be be difficult [4], either for the huge dimension of the jacobian matrix, either because such matrix may be ill conditioned, either because some parts of the matrix must be obtained by numerical differentiation, prone to roundoff and truncation.

In fact the linear system for the Newton step (equation 12) can be written as $[\mathrm{K}]\Delta \mathbf{y}_{i+1} = -\mathbf{G}(\mathbf{y}_i)$:

$$\begin{bmatrix} [\mathrm{W}] & [\mathrm{C_q}]^T \\ [\mathrm{C_q}] & [0] \end{bmatrix} \left\{ \begin{array}{c} \Delta \mathbf{q}_{i+1} \\ \Delta \lambda_{i+1} \end{array} \right\} = \left\{ \begin{array}{c} -\mathbf{g}_i + [\mathrm{C_q}]^T \lambda_i \\ -\mathbf{C}_i \end{array} \right\} \tag{14}$$

where the terms have the following meaning:

- $[\mathrm{W}] := \left[\left[\partial^2 F/\partial \mathbf{q}^2 + \sum_j \lambda_j \partial_{qq} c_j \right] \right]$

- $[\mathrm{C_q}] := [\partial \mathbf{C}/\partial \mathbf{q}]$

- $\mathbf{g} := \partial F/\partial \mathbf{q}$

The block-structured $[\mathrm{K}]$ matrix,

$$[\mathrm{K}] = \begin{bmatrix} [\mathrm{W}] & [\mathrm{C_q}]^T \\ [\mathrm{C_q}] & [0] \end{bmatrix} \tag{15}$$

often called *KKT (Karush-Kuhn-Tucker) matrix*, is highly sparse, symmetric and indefinite. It may be singular if constraints are ill conditioned, that is when $\mathrm{rank}([\mathrm{C_q}]) < \dim(\mathcal{C})$, or when the Hessian block is near singularity, that is when $\det[\mathrm{W}] \approx 0$.

Let consider the following example. If one wants the $\mathbf{q}$ solution which minimizes the objective function $F = \frac{1}{2} \mathbf{q}^T \mathbf{q}$, subject to constraints $\mathbf{C}$, the hessian becomes a simple diagonal matrix $[\mathrm{I}]$. Also, the first step of the Newton procedure turns into the following system, for $\mathbf{q}_0 = \mathbf{0}$:

$$\begin{bmatrix} [\mathrm{I}] & [\mathrm{C_q}]^T \\ [\mathrm{C_q}] & [0] \end{bmatrix} \left\{ \begin{array}{c} \Delta \mathbf{q}_{i+1} \\ \lambda_{i+1} \end{array} \right\} = \left\{ \begin{array}{c} \mathbf{0} \\ -\mathbf{C}_i \end{array} \right\} \tag{16}$$

which can be solved symbolically, given $[\mathrm{I}] = [\mathrm{I}]^{-1}$, with these few steps:

$$\Delta \mathbf{q} = [\mathrm{C_q}]^T \lambda, \tag{17}$$

$$[\mathrm{C_q}][\mathrm{C_q}]^T \lambda = -\mathbf{C}, \tag{18}$$

$$\Delta \mathbf{q} = [\mathrm{C_q}]^T \left[[\mathrm{C_q}][\mathrm{C_q}]^T \right]^{-1} - \mathbf{C}. \tag{19}$$

Then it is interesting to notice how, in this special case, equation 19 gives the same result which could be obtained with the application of a Moore-Penrose pseudoinverse $[\mathrm{C_q}]_i^+ = [\mathrm{C_q}]^T \left[[\mathrm{C_q}][\mathrm{C_q}]^T \right]^{-1}$. However note that, in this case, the direct solution of system 16 through a LU or Gauss method is more favorable to the solution with the pseudoinverse, because $[\mathrm{K}]$ is very sparse, while $[\mathrm{C_q}]^+$ is smaller but full. Indeed, special methods can solve the linear system of equation 16 by exploiting the block-sparsity of the $[\mathrm{C_q}]$ matrix, for example the decomposition of $[\mathrm{K}]$ into a Bunch-Parlett $[\mathrm{L}][\mathrm{D}][\mathrm{L}]^T$ form can be very efficient in this circumstance [5], especially when adopting custom algorithms for sparse matrices factorization [6].

When equation 16 is used in *all* Newton steps, keeping diagonal hessian $[\mathrm{W}] = [\mathrm{I}]$ and always setting $\mathbf{g}_i = \mathbf{0}$, it behave exactly as using the Moore-Penrose pseudoniverse.

Otherwise, if $\mathbf{g}_i$ is kept updated as $\mathbf{g}_i = \partial F/\partial \mathbf{q}$, given objective

$$F = \frac{1}{2}[\mathbf{q} - \mathbf{q}_0]^T[\mathbf{q} - \mathbf{q}_0] \tag{20}$$

we get:

$$\begin{bmatrix} [\mathrm{I}] & [\mathrm{C_q}]^T \\ [\mathrm{C_q}] & [0] \end{bmatrix} \left\{ \begin{array}{c} \Delta \mathbf{q}_{i+1} \\ \lambda_{i+1} \end{array} \right\} = \left\{ \begin{array}{c} -\mathbf{q}_i + \mathbf{q}_0 \\ -\mathbf{C}_i \end{array} \right\} \tag{21}$$
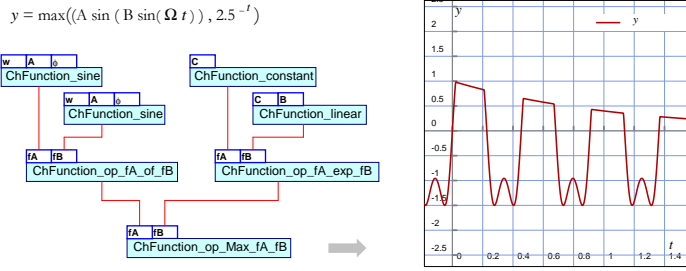
Figure 3: Example of a complex function as a tree of basic building blocks.

whose meaning can be more intuitive: at the end of the Newton loops, the new state will satisfy the constraints, with the smallest change from the initial guess $\mathbf{q}_0$. This is useful in an interactive context, where an user may change the constraints on the functions in successive steps, testing different design solutions: each time a constraint is changed/added, the last state is used as $\mathbf{q}_0$ and the Newton loop with equation 21 is performed, hence providing a new function whose shape is not too far from the last before the tweaking.

3 IMPLEMENTATION

The above theory has been implemented by using the C++ programming language, either in form of a stand-alone library, either integrated into our multibody simulation software [7].

3.1 Software design

Driven by a modular approach, the features of our library allow the building of complex motion laws starting from basic function objects (called *ChFunctions* heretoafter), which can be subject to operations (sequencing, multiplication, time warping, integration, etc.) by linking them to other *ChFunction* objects.

This building-block architecture means that a complex function can be seen as an hierarchical tree of simpler functions (Figure 3).

Also constraints are objects which can be linked to functions as needed, without limits in number or type.

Instancing of functions and constraint objects (as well as setting up their hierarchy) can be done at C++ level, with compiled statements, or at scripting level, using an interactive shell and the compiled language Javascript (ECMA-262 specification, ISO-16262 standard), as in Figure 4.

An interactive graphical user interface has been implemented in our C++ multibody software, thus allowing an intuitive creation and manipulation of *ChFunctions* while designing the motion-laws of robotic devices (Figure 5 and 6).

3.2 Function classes

The *ChFunction* base class implements the common features of all inherited functions, for example a default numerical differentiation is provided, to get $\partial^n y / \partial t^n$. Then, all inherited functions must at least provide the $y = y(t)$, but they could also overload / override the base methods when necessary, for example the *ChFunction_sine* overrides the default numerical differentiation because the analytical derivative is known.

A partial view of the inheritance tree can be seen in Figure 7. Among the many specialized classes, we stress the importance of the *ChFunction_sequence* class, which is used to collect many sub-functions (maybe other sequences too) into a sequence of segments.

There are also functions which can modify other functions, for example *ChFunction_op_Fa_of_Fb* can be used to build functions of the $f_a(f_b(t))$ type.

Note that all functions must provide a method which exposes their optimization parameters, that is the vector of variables which can be collected into the $\mathbf{q}$ vector of equation 2.

3.3 Constraint classes

All constraints are handled as C++ objects. Each *ChFunction* can own an unlimited number of constraints, as they are inserted in a linked list.

Each specialized constraint class must implement the base methods which compute its $\mathbf{C}(\mathbf{q})$ residual, to be used in equation 21 for the Newton Raphson process.

4 EXAMPLES

4.1 Constant acceleration

Motion law ramps with constant acceleration (and deceleration) are well known, being made of two parabolic pieces, with durations t_v and $t_s - t_v$, each with own constant acceleration A and B. If the ramp height h is imposed, for zero start-speed and zero

```
f1 = new ChFunctionSine;          f6op_a = new ChFunctionRamp;
f1.amp = 0.3;                     f6op_b = new ChFunctionSine;
                                  f6.fa = f6op_a;
f2= new ChFunctionRamp;           f6.fb = f6op_b;
                                  f6.op_type = CHOP_MULTIPLY;
f3= new ChFunctionPoly;
f3.order =2;                        // Append functions in seq.:
f3.C0=0.2;                        f_sequence = new
f3.C1=0.1;                             ChFunctionSequence;
                                  f_sequence.append(f1,1,
f4= new ChFunctionSigma;          f2,0.2, f3,0.6, f4,1.5,
f4.amp=-0.4;                      f5,0.8, f6,0.6);
f4.end;
                                    // Create some constraints:
f5= new ChFunctionConstAcc;       c1 = new ChConstraint_Chf_Val(
f5.h=0.9;                              f_sequence, "", 3,1);
f5.av=0.4;
f5.aw=0.6;                        c2 = new
                                  ChConstraint_Chf_Continuity(
f6= new ChFunctionOperation;           f_sequence, "", 0,1);
                                  . . .
```
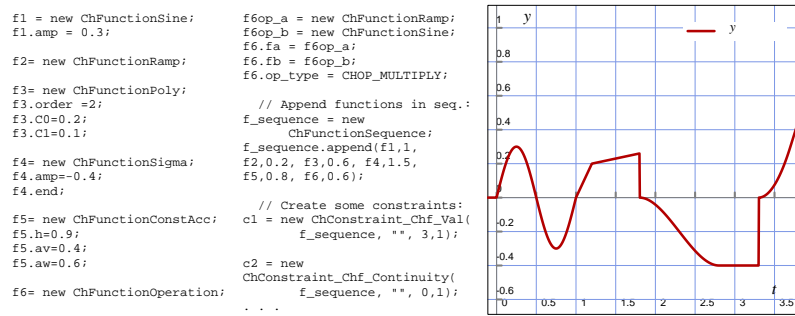
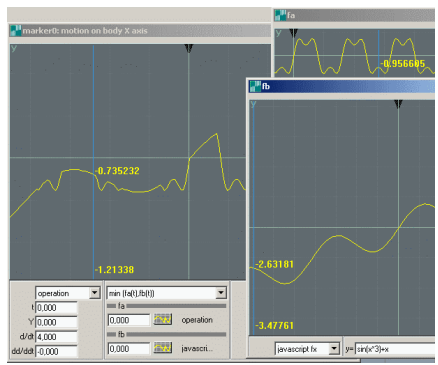Figure 4: Example of function sequence created via Javascript syntax.

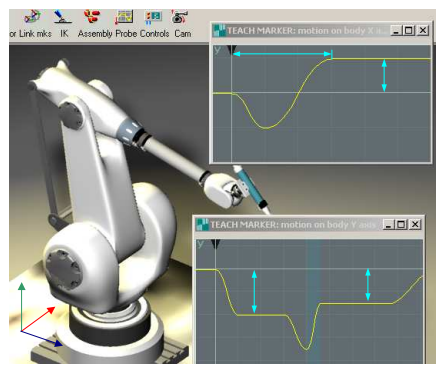Figure 5: Graphical user interface for easy function creation and manipulation.

Figure 6: Constrained parametric design of motion laws as a tool for multibody simulation of robotic devices.

end-speed there is a simple analytical solution: $A = 2h/t_v t_s$, $B = 2h/(t_s(t_s - t_v))$. The same result may be obtained automatically with the method presented in this paper, even if one does not know/remember the analytical solution.

We starts with a function of type *sequence*, containing two sub-functions of type *constant*. Let the duration and values of these constant be arbitrary values at the beginning. Now, put the *sequence* function inside a function of type *integrator*, then again into another *integrator*: we get the data tree of figure 8. Hence the root function is the motion

ramp $y(t)$, for generic constant accelerations whose value is unknown. Finally, adding constraints such as $y|_0 = 0$, $y|_{t_s} = h$, $(\partial y/\partial t)_0 = 0$, $(\partial y/\partial t)_{t_s} = 0$, $t_v/t_s = 0.5$, the method will automatically find the right parameters $\mathbf{q}$ which satisfy all requirements, in detail the A and B values of the acceleration will correspond exactly to the analytical formulas.

4.2 Complex sequence

Now a more complex motion law is presented, where the values of function parameters which can satisfy
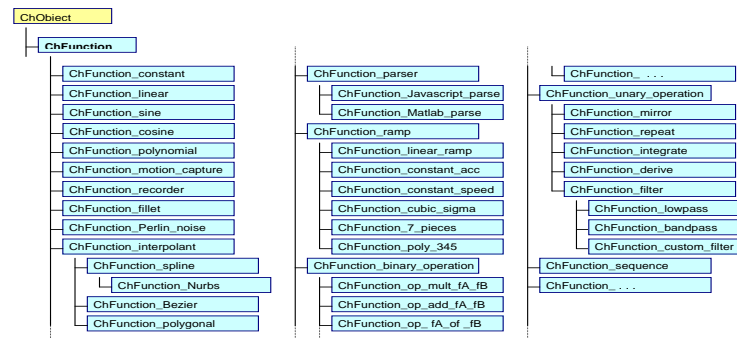

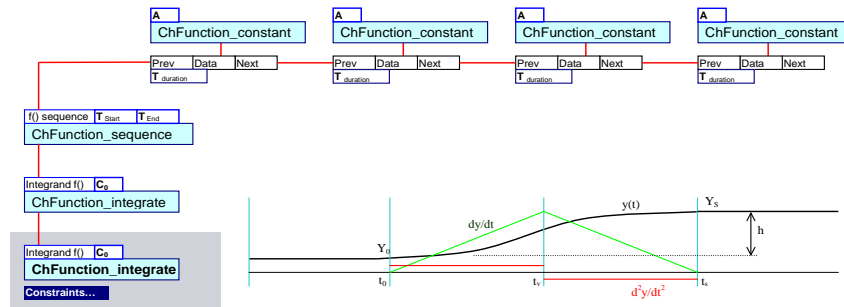Figure 7: Class hierarchy of motion laws.

Figure 8: Example of a block-structured parametric function which reproduces a constant-acceleration motion law.
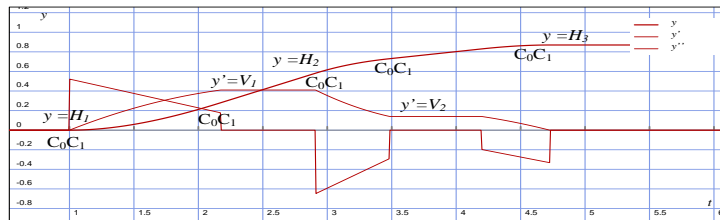

Figure 9: Motion law with multiple constraints. Function segments are of polynomial type.

the constraints are not as trivial as in the previous example. While designing the motion for a tool of an automatic packaging device, a request was that the motion law must have two constant-speed segments (at given time intervals) before reaching the final ramp height. Also, at a given instant of time, the function must pass through a specific y value. The segments between the two constant-speed intervals are of polynomial type, and the resulting motion law (after application of constraints) can be seen in Figure 9. Note that C0 and C1 continuity has been imposed across the interfaces of all the segments.

5 CONCLUSION

A new method to design motion laws has been proposed. It is based on the ability of imposing constraints over sequences of functions, each defined by parameters. The optimal value for these parameters is obtained after the application of a modified Newton-Lagrange process over a constrained optimization problem of KKT type. The outer Newton loop is endorsed by a Bunch-Parlett linear solver which exploits the sparsity of the coefficient matrix, hence allowing high computational efficiency even in case of many constraints.

The resulting theory has been implemented into our software for multibody simulation, thus providing a powerful yet expandable way to design motion laws for robots and automatic devices.

Future works may embrace the automatic adoption of several polynomial shape functions instead of the user-prepared sequence of few function segments.

References

[1] G.Ruggieri, P.Magnani, *Progettazione meccanica funzionale*, Ed. UTET, Torino, Italy, 1990.

[2] R. Faglia, *Progetto del movimento per sistemi a camma tramite algoritmo genetico*, XII AIMETA, Napoli, Ottobre 1995.

[3] S. Rao, *Engineering Optimization: Theory and Practice*, Ed. John Wiley and Sons, New York.

[4] J.C. Haws and C.D. Meyer *Preconditioning KKT systems*, Numer. Linear Algebra Appl. 2001; 00:16

[5] J.R. Bunch, L.Kaufman, and B.N.Parlett, *Decomposition of a symmetric matrix*, Numer. Math., 27 (1976), pp. 95–109

[6] A.Tasora, *An optimized lagrangian multiplier approach for interactive multibody simulation in kinematic and dynamical digital prototyping*, VII ISCSB, Ed. CLUP, Milano, 2001.

[7] P.Righettini, A.Tasora *Implementazione object-oriented delle strutture dati di un codice di calcolo multibody general-purpose*, GIMC 2000 - XIII Convegno Italiano di Meccanica Computazionale, Brescia, 13-15 november 2000, Italy

[8] *Javascript Language Reference*, http://developer.netscape.com/docs/manuals/js/core/jsguide15/contents.html