# Numerical Solution of Differential Systems with Algebraic Inequalities Arising in Robot Programming

Raymond J. Spiteri*, Uri M. Ascher† and Dinesh K. Pai‡
Institute of Applied Mathematics
and Department of Computer Science
University of British Columbia
Vancouver, Canada
*e-mail:* {spiteri/ascher/pai}@cs.ubc.ca

## Abstract

Recently, new robot programming approaches have proposed the use of programmed constraints as an executable specification language for the desired behavior of a robot. The constraint-based approaches are intermediate level languages, promising a higher, more declarative level of programming than trajectory-based approaches, while being more tractable computationally than motion planning.

This paper considers a numerical algorithm for solution of differential systems subject to algebraic inequality constraints. These are the mathematical structures behind the constraint-based approach. Our approach is based on a principle of 'least constraint', consisting of a dynamic integration of the equations of motion coupled with invocation of a control mechanism to ensure that the robot trajectory avoids all constraint boundaries. This is achieved by minimization of a barrier function defined using buffer zones near the constraint boundaries. Determination of the buffer zones is done dynamically, corresponding to a local planning strategy.

## 1 Introduction

Most industrial robots are programmed by explicit specification of motion trajectories in joint space or Cartesian space. It is well known that such approaches, while simple to implement, are not simple to program with. This is particularly true in cluttered and dynamic environments with changing task requirements – such robot programming tasks are more naturally viewed as constraint satisfaction problems. Most constraints arise naturally from the geometry of the environment obstacles and the robot's kinematics and dynamics (e.g. [20, 5]). There has been considerable work in using only these natural constraints along with the goal state as the specification of the robot's behavior and solving the resulting motion planning problem (see [18] for a survey). A particularly important case is the optimal control problem in which a motion is sought which optimizes a given functional, such as the time or energy required for the motion, while satisfying constraints (e.g. [15, 24]). While motion planning can provide a very high level of programming, it is extremely expensive computationally, both theoretically and in practical situations with moderately high degrees of freedom.

More recently, new robot programming approaches have promoted the use of artificial, user-defined constraints (*programmed constraints*), as an executable specification language for the desired behavior of a robot (e.g. [21, 22, 26, 11]). These constraint-based approaches are intermediate level languages, promising a higher, more declarative level of programming than trajectory-based approaches, and being computationally more tractable than motion planning.

Consider an example from [21] of programming a pick-and-place task (as depicted in Figure 1.1) of placing an object in the robot's hand on the table. In a conventional robot programming language, this must be expressed by arbitrarily commanding a particular motion to some point on the table. In the Least Constraint approach, the task is programmed by placing constraints in the hand position domain. The hand is constrained to stay within the boundaries of the table by the constraints $f_1$ and $f_2$. The constraint $f_3$ moves down at constant speed $v$ from height $h$, so that after a time of $h/v$, the end-effector is on the table top. Note that the exact location on the table top is unspecified since this is not important for the success of the task. An advantage of this approach is that a program can be easily modified by adding new constraints. For example, if we now wish

*supported in part by an NSERC Centennial Grant, DAAD Fellowship and Killam Postgraduate Grant
†supported in part by an NSERC Research Grant
‡supported in part by grants from NSERC, B.C. ASI, and IRIS

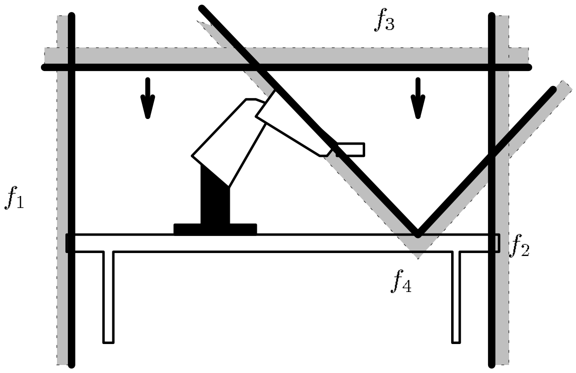


Figure 1.1: Place on Table

to place the object at a particular location on the table, this can be achieved by adding the cone-shaped constraint marked $f_4$ in Figure 1.1.

In this paper we consider the numerical solution of the system of differential equations and constraints arising from such robot programming methods. Specifically, we consider the numerical solution of initial-value problems of the form

$$\begin{aligned} \dot{q} &= f(t,q) + B(t,q)u && \text{(1.1a)} \\ q(0) &= q_0, && \text{(1.1b)} \\ \text{subject to} \quad c(t,q) &\geq 0, && \text{(1.1c)} \end{aligned}$$

where $q \in \Re^n$ are the state variables, $c \in \Re^{n_{cnstr}}$ are inequality constraints to be satisfied by $q$ for all $t \geq 0$, and $u \in \Re^{n_{cntrl}}$ are the controls. [1]

Since in practice controls cannot take on arbitrary values, they must also satisfy a feasibility condition,

$$|u_j| \leq u_{max,j} \text{ for } j = 1, 2, \ldots, n_{cntrl}. \qquad (1.2)$$

The aim of our algorithm is to prescribe feasible values of $u$ for the equations of motion (1.1a), which allows (1.1c) to be satisfied. We note that despite being similar in appearance, inequalities (1.2) are treated differently than inequalities (1.1c). The variables $u$ in (1.2) are 'independent' parameters of the problem to be determined by the algorithm, while the relations (1.1c) are quantities that must be preserved through this choice of $u$.

## 1.1 Mobile Robot Example

For concreteness, the concepts presented will be illustrated using the example of a mobile robot. The model is based on the Dynamite multiple mobile robot testbed developed by the Dynamo project in our laboratory [1]. Figure 1.2 shows the actual testbed and Figure 1.3 shows a schematic.

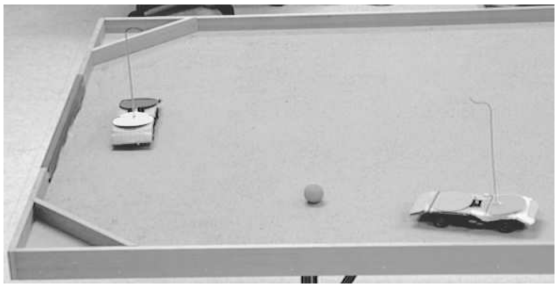

Figure 1.2: Dynamite mobile robots playing soccer

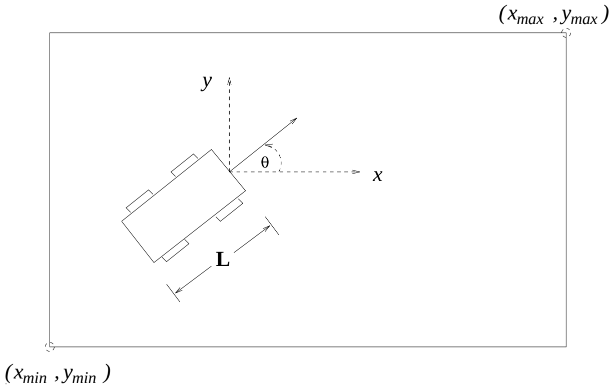


Figure 1.3: The mobile robot (not to scale).

The equations of motion of the robot may be written as

$$\begin{aligned} \dot{x} &= v\cos\theta && \text{(1.3a)} \\ \dot{y} &= v\sin\theta && \text{(1.3b)} \\ \dot{\theta} &= v\frac{\tan\psi}{L} := v\alpha && \text{(1.3c)} \\ \dot{v} &= a - \gamma v && \text{(1.3d)} \end{aligned}$$

where $\gamma \geq 0$ is a damping parameter ($\gamma = 0$ is the no-friction limit) [2]. Note the use of $\alpha$ as a control variable instead of the steering angle $\psi$ in order to cast (1.3) in the form (1.1). This redefinition is possible since $\psi$ and $\alpha$ are in one-to-one correspondence in the allowed range of $\psi$. Thus system (1.3) is of the form (1.1a), with $q = (x, y, \theta, v)^T$ and with the

[1] The vector notation $v \geq 0$ denotes the non-negativity of each component of the vector $v$.

[2] For a more precise treatment the $\psi$-dynamics and the rotational dynamics of the car would also be modeled.

inequality constraints taken to be

$$c = \begin{pmatrix} x - x_{min} \\ x_{max} - x \\ y - y_{min} \\ y_{max} - y \\ v_{max} - v \\ v_{max} + v \\ y \cos\phi - (x - v_w t) \sin\phi \end{pmatrix}. \qquad (1.4)$$

The first two pairs of constraints keep the robot on a table with opposite corners at Cartesian coordinates $(x_{min}, y_{min}) = (5, 5)$ and $(x_{max}, y_{max}) = (234, 112)$ (dimensions are in cm). The third pair of constraints define the absolute allowable size of the velocity of the robot. The final constraint is the driving constraint, represented as a plane wave traveling with $x$-component velocity $v_w$ at angle $\phi \in (0, \frac{\pi}{2}]$ with the $x$-axis. The bounds on the controls are

$$|a| \le 150\ cm\ s^{-2} \qquad\qquad |\alpha| \le \frac{1}{25}\ cm^{-1},$$

with the first bound representing the maximum allowed throttle and the second bound yielding a minimum turning radius of 25 cm.

# 2 Differential Systems with Algebraic Inequalities

A differential system subject to a nonempty set of inequality constraints and a (possibly) nonempty set of equality constraints will be referred to as a *Differential system with Algebraic Inequalities* (DAI). Mathematically, the system is formulated in (1.1).

Much is known for the DAE case, where the inequalities (1.1c) are replaced by equalities [2]. Some work has also been done on the numerical solution of DAIs [19, 25, 12]. For a multibody system with no control, or controls chosen irrespective of inequality constraints (1.1c), a popular strategy is to make constraints about to be violated *active*, e.g. for a given state and time, $c_i(t, q(t)) = 0$. In an *active set* algorithm one then maintains the set of active constraints as a function of time, introducing (equality) constraint forces into (1.1a), by constructing $B$ from the transpose of the active rows of $\frac{\partial c}{\partial q}$ and where now $u$ are Lagrange multipliers. Thus, a DAE of a dynamically varying size is integrated.

Now define the *feasibility region* $D = D(t)$ to consist of all points $q$ satisfying $c(t, q) \ge 0$. We assume that $D(t)$ has a nonempty interior for all $t$. Considering again (1.1) (with $B$ generally unrelated to $\frac{\partial c}{\partial q}$, as in the mobile robot example of § 1.1), an active set approach can be developed (so long as $B$ has a full column rank) where the solution $q(t)$ sticks to the boundary of the feasibility region (or to some prescribed time dependent path inside $D$). In contrast, we proceed here with the Least Constraint approach [21], which attempts to find a trajectory inside the feasibility region, away if possible from the boundary. This latter approach is more natural for robotics.

# 3 Required Properties of a DAI Solver

As discussed thus far, the objective of a DAI solver is to facilitate the numerical integration of the equations of motion (1.1a), with the control $u(t)$ prescribed so as to retain feasibility. The choice of $u(t)$ is made locally and is based on a *barrier method* [6]. Thus, when deemed too close to the boundary of the feasibility region $D(t)$, a minimization step is invoked, where $u(t)$ is chosen so as to approximately minimize an objective function which penalizes states which are close to the boundary. Given this purpose and basis for methodology, desirable features of a solution method should include the following:

- *Small computational expense* in determining the control. This is especially important since the computations must be carried out in real time.

- *Robustness* A suitable strategy should be capable of providing solutions in difficult or varying geometries, as well as exhibiting a degree of insensitivity to minor random errors in measurement, modeling and finding the precise value of the local optimal control.

Completeness would be a desirable property as well, and it is intuitively clear that a lack of planning will inevitably result in the system allowing itself to arrive at a state from which no feasible solution exists. In this sense, the situation is somewhat similar to the problem of systems ending up in local minima using artificial potential field methods [17, 18].

Although we wish to avoid a global motion planning problem, a completely local approach will be limited in the scope of problems that can be solved. In many situations feasible solutions are easily found with rudimentary planning schemes, but are impossible without. The space in which the differential components $q$ exist is continuous in time. As a consequence of the feasibility conditions (1.2) on $u$, the flow of the solution from a given configuration to any other in an arbitrary length of time is not possible. That is, not all points in phase space are *reachable* in a fixed amount of time from a given configuration by a feasible control. Thus, as part of a planning procedure, some sort of anticipatory or *buffer* zone near the boundary of the feasibility region must be created and maintained in order to give any algorithm a 'reasonable opportunity' to choose a control (or sequence of controls, perhaps at consecutive time steps) to keep the system inside the feasibility region. A prediction strategy is desired which allows the reachable set to be expanded to an acceptable size.

We now describe in more detail some of the more central aspects of a DAI solver: the discretization of the DAI system; the local planning strategy; the use of a sensitivity parameter to gauge problem difficulty; and finally the algorithm for choosing the local control in light of the given constraints.

# 4 A DAI Solver for Robot Control

We assume that system (1.1) can be defined in such a way that maintaining $u$ at some prescribed function leads to a 'natural' or otherwise desirable behavior. For convenience, we assume this function has been normalized to zero. The basic strategy is to proceed with natural motion ($u = 0$) until a constraint is deemed to be 'violated'. By this we mean only that a previously defined buffer zone has been penetrated. The control function $u(t)$ is taken to be piecewise constant (continuous on each open interval $(t_{n-1}, t_n)$) with 0 as the default.

## 4.1 Discretization

The discretization of the ODEs is required for two reasons. First, for the local control problem described below in § 4.3, we require the state $q_n$ at time $t_n$, as a function of the control $u$. In this case the equations of motion are integrated by the plant itself. The second requirement is for simulation of the system (1.1a), which is used for verification, either offline or for online, super-real-time simulation.

A suitable integration scheme is chosen based on the differential problem (1.1a) at hand. Typically, the system (1.1a) is *separable* in the following way

$$\dot{x} = f_1(t, y) \tag{4.5a}$$
$$\dot{y} = f_2(t, x, y) + B_2(t, x)u \tag{4.5b}$$

(of which (1.3) is an example). In order to satisfy the implicit requirements of (4.9a) below and still have an explicit scheme, we combine the backward Euler scheme for (4.5a) and the forward Euler scheme for (4.5b). This yields a *composite Euler scheme*:

$$\begin{aligned} y_n &= y_{n-1} + \Delta t f_2(t_{n-1}, x_{n-1}, y_{n-1}) \\ &+ \Delta t B_2(t_{n-1}, x_{n-1})u \\ x_n &= x_{n-1} + \Delta t f_1(t_n, y_n) \end{aligned}$$

where $\Delta t = t_n - t_{n-1}$. This is the simplest explicit method possible and is very useful in practice when higher-order schemes are not deemed worthwhile.

## 4.2 Local Planning Strategy

Having discussed the need for planning, we now discuss its implementation through the use of buffer zones. Let the quantities $c_{safe,i} \geq 0$, $i = 1, 2, \ldots, n_{cnstr}$ denote the inner edge of the buffer zone. The buffer zone itself at time $t = t_{n-1}$ is then the set of all states $q_{n-1}$ such that

$$0 \leq c(t_{n-1}, q_{n-1}) \leq c_{safe}.$$

Our strategy is based on treating the constraints $c$ as independent quantities and not inferring much into their dependence on $q, t$ or indirectly on $u$. That is, we treat the components of $c$ simply as functions which we would like to remain positive over time. Hence the quantities $\dot{c}$ and $\ddot{c}$ at a given time are useful in predicting how $c$ may evolve locally. These quantities may be computed formally or numerically by difference quotients.

For example, for $c_1$ of the mobile robot system (1.3), we have

$$\dot{c_1} = \dot{x} = v\cos\theta \tag{4.6a}$$

and

$$\begin{aligned} \ddot{c_1} &= \dot{v}\cos\theta - v\sin\theta\dot{\theta} \\ &= a\cos\theta - v^2\sin\theta\alpha. \end{aligned} \tag{4.6b}$$

Note that the controls are treated as constant over each time step, so $\dot{a} = \dot{\alpha} = 0$ wherever they occur.

The theoretical motivation behind our prediction strategy is the following. Let the system be in a feasible configuration $q_{n-1}$ at time $t = t_{n-1}$. If $\dot{c} \geq 0$ at $(t_{n-1}, q_{n-1})$, we continue normal integration with $u = 0$ since the system is not locally moving towards the boundary of the feasibility region. If for some $i$, $\dot{c}_i < 0$, we compute $\ddot{c}$ and form the discriminant $\mathbf{\Delta}$ with components defined by

$$\Delta_i = 2c_i\ddot{c}_i - \dot{c}_i^2 \quad i = 1, 2, \ldots, n_{cnstr}. \tag{4.7}$$

If $\mathbf{\Delta} > 0$, then the parabolas given by

$$c + \dot{c}t + \frac{1}{2}\ddot{c}t^2$$

have no real roots and hence *locally* we deem that constraint violation will not occur immediately.

If $\Delta_i \leq 0$, this indicates that an impact with the boundary of $c_i$ is likely to occur, provided $\dot{c}_i, \ddot{c}_i$ remain relatively constant and $u \equiv 0$. Of course, this is not necessarily a good long-time predictor, since generally $\dot{c}_i, \ddot{c}_i$ are *not* constant. However, these are reasonable local-time predictors and can be useful over the entire interval of integration provided they are updated with sufficient frequency.

The point $c_i$ at which $\Delta_i \leq 0$ becomes $c_{safe,i}$ and is fixed at this value until a time at which $c_i > c_{safe,i}$ occurs or is expected to occur based on the previous two tests involving $\dot{c}_i$ or $\Delta_i$.

However, there is a shortcoming since, in most common multibody and robotics systems, constraints occur in *opposing* pairs (so-called *box constraints* are typical examples). In the case of a shrinking feasibility region (which is usual in nontrivial problems), it is easily seen that the conditions $\dot{c} \geq 0$ or $\mathbf{\Delta} > 0$ can never be satisfied. We would therefore like a test of more practical utility, the motivation for which is described now, using the mobile robot example.

Suppose the mobile robot is moving toward the right end of the table (in the absence of friction) with $\theta = 0$ and at some (small) velocity $\epsilon\ cm\ s^{-1}$. We temporarily ignore the influence of the driving constraint $c_7$ by setting $v_w = 0$. Then

$$\dot{c}_2 = -\epsilon < 0, \quad \ddot{c}_2 = 0$$

so that

$$\Delta_2 = -\epsilon^2 < 0.$$

Thus the test would predict that a control step should be taken. However, if $a_{max} \geq \frac{\epsilon}{h}$, then theoretically, no planning is *ever* required (in the absence of discretization errors). That is, it will be possible to stop the car before it goes over the edge in the very last step. If now $v_w > 0$ this test would initiate a control at *every* step. So, on the one hand, no planning is very inexpensive (computational cost may only be little more than that of a nonstiff integrator) but suffers from not being robust in finding solutions, while advancing with some nonzero control at every step is too expensive computationally.

Thus we wish a means to establish how the system may respond to a nonzero control. For the above example, we propose to recompute $\dot{c}_2$, $\ddot{c}_2$ with a control having components of *magnitude* $u_{frac,j}u_{max,j}$ where $u_{frac,j} \in (0,1]$, $j = 1, 2, \ldots, n_{cntrl}$.[3] The signs of the various components are as yet unknown, since it is not known a priori which will be the most advantageous combination. In the preceding situation for example, the appropriate $a_{frac}$ is negative.

Such a strategy is not difficult to implement in practice. The expressions for $\dot{c}, \ddot{c}$ can be reduced to

$$\dot{c} = Gu + r \tag{4.8a}$$
$$\ddot{c} = Hu + s. \tag{4.8b}$$

Using these forms, we determine whether a choice of $u$ with components of magnitude $u_{frac,j}u_{max,j}$ can be found which will make $\dot{c} \geq 0$; and if not, then $\mathbf{\Delta} \geq 0$. That is, we are searching the vertices of an $n_{cntrl}$-dimensional box with sides $2u_{frac,j}u_{max,j}$ centred at $u = 0$ to see if the conditions $\dot{c} \geq 0$ or $\mathbf{\Delta} \geq 0$ hold. If such a vertex exists, then we define the system to be in a state of *strong approximate safety*. This indicates that, based on our second-order approximation of $c(t)$, a (small) control will likely enable the system to avoid all constraints. Unfortunately, this condition cannot be satisfied in the typical case of opposing constraints and shrinking feasible region. Moreover, testing for strong approximate safety can be computationally expensive.

However, a practical test can be borne from relaxing this requirement. If $u_j$ appears with coefficient $g_{ij}$, $h_{ij}$ in the expressions (4.8a) and (4.8b) respectively, then we recompute $\dot{c}_i, \Delta_i$ using the nonzero values $|g_{ij}u_j|$ and $|h_{ij}u_j|$. Thus we are now searching the vertices for $\dot{c}_i \geq 0, \Delta_i \geq 0$, noting that it could happen that a different vertex be suitable for different constraints. If such vertices can be found, we define the system to be in a condition of *weak approximate safety*. That is to say, a small control can be expected to avoid any one constraint, but not necessarily all constraints at once.

For example, recall equations (4.6a), (4.6b) for $\dot{c}_1$, $\ddot{c}_1$. The estimate of 'appropriate' control with components of magnitude $u_{frac,i}u_{max,i}$ then uses

$$\dot{c}_1^* = v\cos\theta \quad \text{(no direct dependence on control)}$$

and

$$\ddot{c}_1^* = a_{frac}a_{max}|\cos\theta| + \alpha_{frac}\alpha_{max}v^2|\sin\theta|$$

in the calculation of

$$\Delta_1^* = 2c_1^*\ddot{c}_1^* - \dot{c}_1^{*2}.$$

If the condition of weak safety holds, then the integration continues with $u = 0$. Thus far we have found this test to be of practical utility. The quantity $U_{frac}$ allows for errors in constraint measurements, the model, or predictions and potential inconsistencies there within. It can be considered a sensitivity parameter, initially specified by the user, but then dynamically altered.

[3] Henceforth let $U_{frac}$ be a diagonal matrix with nonzero entries $u_{frac,j}$, $j = 1, 2, \ldots, n_{cntrl}$.

## 4.3 A Local Control Algorithm

As mentioned, the basic strategy is to allow the system to execute natural motion ($u = 0$) and use constraint sampling combined with prediction strategies to determine when a control step is required. We now describe briefly how such a control is computed.

Let the system be in a feasible state $q_{n-1}$ at time $t_{n-1}$, and suppose the prediction strategy has deemed a control step necessary. We delineate the beginning of a buffer zone by assigning constants $c_{safe} = c(t_{n-1})$. These values remain fixed until such time that, for example, some component $j$ satisfies $c_j > c_{safe,j}$, in which case the buffer zone associated with constraint $j$ is renormalized. Having established the buffer zone, we consider solving (or at least taking one iteration towards the solution of) the following constrained nonlinear programming problem:

$$\min_u \left\{ F(q_n(u)) = F(u) = \sum_{i=1}^{n_{cnstr}} \phi_i(q_n(u)) \right\} \quad (4.9a)$$

subject to

$$|u_j| \leq u_{max,j} \quad for\ j = 1, 2, \ldots, n_{cntrl} \quad (4.9b)$$

where

$$\phi_i = \begin{cases} \frac{c_i}{c_{safe,i}} - 1 - \log(\frac{c_i}{c_{safe,i}}) & c_i < c_{safe,i} \\ 0 & c_i \geq c_{safe,i}. \end{cases}$$

We use a logarithmic barrier since it not only disallows infeasible solutions, but also serves to steer the system away from the boundaries. Other choices such as inverse or inverse square functions are possible, depending on the desired interaction of the system with the barrier function [3, 9, 10].

We note that this approach shares some features with the artificial potentials that have been used by many researchers in planning and navigation algorithms [16, 13, 17]. In those methods, the potential field is also given a direct physical interpretation, and the potential gradient is used as an applied force. We do not use this direct interpretation.

The advantage of logarithmic barriers lies in their relatively slow divergence near the singularity. Moreover, we choose $F \in \mathcal{C}^1$.[4] This behavior may deter a 'bang-bang' choice for components of the control (i.e. from 0 to $|u_j| = u_{max,j}$) when moving the system away from the boundary. In this way, the previous stability of the system evolving with a fixed natural value for the control is not excessively undermined.

A standard method for solving (4.9) is the so-called *penalty function method*, that is, through the solution of a sequence of unconstrained minimization problems where the objective function $F$ is augmented with a penalty term [8]. The strategy adopted for use in practice is taken from [7, 4, 8, 14, 23].

With the obtained control $u^*$ from (4.9), the system (1.1a) is advanced one time step, starting from $(t_{n-1}, q_{n-1})$. Then we return (at least initially) to the 'natural' control $u = 0$ for the next time step.

# 5 Numerical Experiments

In this section, numerical simulation is employed to demonstrate the general applicability of the solution procedure, the types of solutions obtained and some efficiency measures. This is done using the mobile robot example of § 1.1, in scenarios of varying difficulty. We plan to implement the algorithm on the physical system in the near future.

Initial conditions of the system (1.3) are $x(0) = 52.7$, $y(0) = 50$, $v(0) = 0$, with $\theta(0)$ and other parameters specified in the tables. All simulations use an adaptive sensitivity parameter, starting from a user-defined $U_{frac}$. Knowledge of the existence of opposing constraints is assumed and the stepsize is $\Delta t = t_n - t_{n-1} = \frac{1}{60}$ seconds, representing the actual sampling time of our system. A first order integration scheme was chosen due to simplifications made in the model equations. Simulations end when the trailing (lower) edge of the plane wave is one car-length from the right edge, ie. $t_{final} = (x_{max} - x_{min} - L)/v_w$.

The tables display three different measures of the performance. The performance of the algorithm is partly measured by the quantity $\%u$, which represents the number of time steps which invoked the logarithmic barrier subroutine to compute a control, as a percentage of the total number of time steps. Thus it measures the complexity of the control computation in a new way – as the percentage of time spent having to explicitly account for the constraints. The constraints, along with their velocities and accelerations, are monitored at a frequency $c_{freq}$. However, explicit constraint avoidance by means of barrier minimization is only executed a fraction $\%u$ of the total integration steps. In addition, we record the quantity $\%u_{max}$ which measures the percentage of *control steps* spent at maximum control, for each control variable. Finally, typical execution time runs (in seconds) on an SGI Indy are given under CPU. We note that this includes simulation time and disk I/O, but is still considerably faster than real time ($\sim$ 200 seconds).

**Case 1:** $\phi = \frac{\pi}{2}$, $\theta(0) = 0^o$, $c_{freq} = 2$.

[4] We thank P. Loewen for suggesting the smooth $\phi_i$.

| $v_w$ | $\gamma$ | $u_{frac}$ | $\%u$ | $\%u_{max}$ $(a)$ | $(\alpha)$ | CPU |
|---|---|---|---|---|---|---|
| 1 | 0 | 0.5 | 0.04 | 0.0 | 0.0 | 1.15 |
| 8 | 0 | 0.5 | 4.38 | 0.0 | 0.0 | 1.75 |
| 10 | 0 | 0.5 | 6.29 | 0.0 | 0.0 | 2.11 |
| 1 | $\frac{1}{6}$ | 0.5 | 38.38 | 0.0 | 0.0 | 24.85 |
| 8 | $\frac{1}{6}$ | 0.5 | 47.24 | 0.76 | 0.13 | 3.95 |
| 10 | $\frac{1}{6}$ | 0.5 | 48.01 | 1.09 | 0.0 | 2.86 |

Table 5.1: Simulation results: 1D case

For the undamped case, since $x(0)$ is fairly large, the robot builds speed (approaching $v_w$), and stays in front of the wave. Thus, $\%u$ is very small since the robot satisfies the constraints mostly without using any control. The behaviour is similar with damping turned on. As expected, the damping stabilizes the motion of the robot. The robot stays ahead of the wave but due to the damping falls back into the buffer zone of the wave. When this happens, the robot spurts further ahead of the wave. Hence we observe that $\%u$ is higher than in the undamped case, and increases with the speed of the wave. However, CPU times are still on the order of 10% of the real simulation times. We note that this is effectively a one-dimensional problem: there is no incentive for the system to change its $y$-coordinate.

**Case 2:** $\phi = \frac{\pi}{2}$, $\theta(0) = 45^o$, $u_{frac} = 0.01$, $v_w = 1$.

| $\gamma$ | $c_{freq}$ | $\%u$ | $\%u_{max}$ $(a)$ | $(\alpha)$ | CPU |
|---|---|---|---|---|---|
| 0 | 2 | 9.98 | 1.5 | 92.87 | 15.38 |
| $\frac{1}{6}$ | 1 | 80.28 | 0.0 | 25.38 | 77.15 |

Table 5.2: Simulation results: Case 2

Now, the initial configuration of the robot is changed to $45^o$. The robot initially continues to move at this orientation until it reaches the top wall (constraint $c_4$), at which point it turns to $\theta \approx 0$ and moves parallel to the top wall. Since the car must perform a $45^o$ turn to avoid the top boundary, the quantity $\%u$ is expected to be much higher than in the previous case. We note that despite this, the algorithm still exhibits low CPU indicating the efficiency of the approach.

Below are various other cases. Note that the difficulty level increases dramatically with decreasing $\phi$ since the final size of the feasible region is quite small.

**Case 3:** $\phi = \frac{\pi}{4}$, $c_{freq} = 1$, $v_w = 1$.

| $\theta(0)$ | $\gamma$ | $u_{frac}$ | $\%u$ | $\%u_{max}$ $(a)$ | $(\alpha)$ | CPU |
|---|---|---|---|---|---|---|
| 0 | 0 | $10^{-2}$ | 58.44 | 0.17 | 97.37 | 84.59 |
| 0 | $\frac{1}{6}$ | $10^{-2}$ | 100.0 | 0.0 | 15.02 | 81.03 |
| -45$^o$ | 0 | $10^{-3}$ | 34.88 | 0.09 | 98.02 | 47.64 |
| -45$^o$ | $\frac{1}{6}$ | $10^{-3}$ | 100.0 | 0.0 | 41.43 | 144.54 |
| 44$^o$ | 0 | $10^{-4}$ | 80.42 | 0.01 | 97.39 | 114.24 |
| 46$^o$ | 0 | $10^{-5}$ | 81.63 | 0.02 | 96.77 | 109.23 |

Table 5.3: Simulation results: Assorted Cases

Note that $u_{frac}$ was decreased in the tougher problems, and we used $\alpha_{max} = 0.042$ in the very last result. Finally, we should point out that due to the inherent local nature of the approach, the method can fail in certain cases, especially when several constraints are active and at small angles to each other, and when the feasibility region is narrow. However, the efficiency of the method suggests that it is effective and practical in many real applications.

# 6 Conclusions

We have described a numerical method for solving the DAIs arising in robot programming. A key motivation for the method is to reduce the amount of computation for such dynamic constraint satisfaction problems by exploiting the fact that the feasibility region will usually not have a pathological geometry and may also be smoothly changing. Our approach combines a composite-Euler discretization with a constrained minimization to obtain the control. Local planning is introduced which provides a state dependent test for future constraint violation, and is used to activate new constraints. Our simulation results to date indicate that the method is efficient, both in deciding when to use a control and then in performing the ensuing calculation. For the near future, we plan to focus on testing the algorithm with the real mobile robots in our laboratory, and further improving the robustness of the method in difficult cases.

# References

[1] R. Barman, S. Kingdon, J. J. Little, A. K. Mackworth, D. K. Pai, M. Sahota, H. Wilkinson, and Y. Zhang. Dynamo: real-time experiments with multiple mobile robots.

In *Proceedings of the IEEE Intelligent Vehicles Conference*, pages 261–266, Tokyo, Japan, July 1993.

[2] K. Brenan, S. Campbell and L.R. Petzold, *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*, North-Holland (1989).

[3] C.W. Carroll, *The created response surface technique for optimizing nonlinear, restrained systems*, Operations Res. **9** (1961) pp.169-184.

[4] J.D. Coope and R. Fletcher, *Some numerical experience with globally convergent algorithms for nonlinearly constrained optimization*, J. Opt. Theo. Applns, Vol.32, No.4 (1979) pp.1-16.

[5] B. R. Donald. A search algorithm for motion planning with six degrees of freedom. *Artificial Intelligence*, 31(3), 1987.

[6] A.V. Fiacco and G.P. McCormick, *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*, John Wiley and Sons, New York (1968).

[7] R. Fletcher, *An ideal penalty function for constrained optimization*, J. Inst. Maths. Applns **15** (1975) pp.319-342.

[8] R. Fletcher, *Practical Methods of Optimization, Volume I*, John Wiley and Sons, Chichester (1980).

[9] R. Fletcher, *Practical Methods of Optimization, Volume II*, John Wiley and Sons, Chichester (1981).

[10] K.R. Frisch, *The logarithmic potential method of convex programming*, Memorandum, University Institute of Economics, Oslo (1955).

[11] J. Funda, R. Taylor, K. Gruben, and D. LaRose. Optimal motion control for teleoperated surgical robots. In *SPIE Symposium on Optical Tools for Manufacturing and Advanced Automation*, Boston, MA, September 1993.

[12] C.W. Gear, *Maintaining solution invariants in the numerical solution of ODEs*, SIAM J. Sci. Stat. Comput., Vol.7, No.3 (1986).

[13] E. Gilbert and D.W. Johnson, *Distance functions and their application to robot path planning in the presence of obstacles*, IEEE Journal of Robotics and Automation, RA-1(1) (1985) pp.21-30.

[14] M.R. Hestenes, *Multiplier and gradient methods*, J. Opt. Theo. Applns **4** (1969) pp.303-320.

[15] M. E. Kahn and B. Roth *The near-minimum time control of open loop articulated kinematic chains*, ASME Journal of Dynamic Systems, Measurement, and Control, Vol. 93, No. 3, pp. 164-172, 1971.

[16] O. Khatib, *Real time obstacle avoidance for manipulators and mobile robots*, The International Journal of Robotics Research, Vol.5, No.1 (1986) pp.90-99.

[17] D.E. Koditschek, *Robot planning and control via potential functions*, The Robotics Review 1, MIT Press (1989).

[18] J. C. Latombe. *Robot Motion Planning.* Kluwer, 1991.

[19] P. Lötstedt, *Numerical simulation of time-dependent contact and friction problems in rigid body mechanics*, SIAM J. Sci. Stat. Comput., Vol.5, No.2 (1984).

[20] Tomás Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, C-32(2):108–120, February 1983.

[21] Dinesh K. Pai. Least constraint: A framework for the control of complex mechanical systems. In *Proceedings of the American Control Conference*, pages 1615 – 1621, 1991.

[22] Dinesh K. Pai. Robot programming and constraints. In *First Workshop on Principles and Practice of Constraint Programming*, pages 250–257, Newport, RI, April 1993.

[23] M.J.D. Powell, *A method for nonlinear constraints in minimization problems*, Optimization (Ed. R. Fletcher), Academic Press, London (1969).

[24] S. K. Singh, "Motion Planning wiht Obstacles and Dynamic Constraints," Ph.D. Thesis, Cornell University, Jan 1988.

[25] L.F. Shampine, *Conservation laws and the numerical solution of ODEs*, Comp. & Maths. with Appls., Vol.12B, No.5/6, (1986) pp.1287-1296.

[26] Ying Zhang and Alan Mackworth. Constraint programming in constraint nets. In *First Workshop in Principles and Practice of Constraint Programming*, 1993.