Programming and Control of Robots by means of Differential Algebraic Inequalities

Raymond J. Spiteri, Dinesh K. Pai, and Uri M. Ascher

Abstract— **The method of programmed constraints has recently been proposed as an executable specification language for robot programming. The mathematical structures behind such problems are viability problems for control systems described by ordinary differential equations (ODEs) subject to user-defined inequality constraints. This paper describes a method for the numerical solution of such problems, improving and extending results presented in [1].**

The algorithm presented is composed of three parts: delay-free discretization, local control, and local planning. Delay-free discretizations are consistent discretizations of control systems described by ODEs with discontinuous inputs. The local control is based on the minimization of an artificial, logarithmic barrier potential function. Local planning is a computationally inexpensive way to increase the robustness of the solution procedure, making it a refinement to a strategy based on viability alone.

Simulations of a mobile robot are used to demonstrate the proposed strategy. Some complementarity is shown between the programmed-constraints approach to robot programming and optimal control. Moreover, we demonstrate the relative efficiency of our algorithm compared to optimal control: Typically, our method is able to find a solution on the order of one hundred times faster than an optimal-control solver.

I. Introduction

There has been considerable work done in the field of motion planning (see e.g., [2], [3] and the references therein), where the constraints formed by obstacles, the robot's kinematics and dynamics, and knowledge of the goal state are used to determine the robot's behaviour. Optimal control, which attempts to provide a motion that optimizes a given functional such as time taken or energy required, forms an important class of this type of problem. Provably good kinodynamic algorithms exist that yield ϵ-optimal path plans [4], [5], [6], [7]. However, although such motion planning can provide a high level of programming, it comes at great computational expense, both theoretically and in practical situations with moderately high degrees of freedom.

Other successful robot programming methods can be classified as control-theoretic methods. The algorithms developed are generally constructive in nature, and they make good use of theoretical tools from nonlinear control theory, differential geometry, and topology (see, e.g., [8], [9], [10], [11], [12], [13] as representative work). These methods can be classified as open-loop in nature, and typically treat kinematic constraints but not dynamic ones.

Methods that use potential fields for robot programming have also been proposed [14], [15], [16], [17], [18], [19], [20], [21]. These methods offer a unified approach to task description and control. Moreover, the ensuing computations can also be carried out in real time. However, the construction of complex potential functions through the combination of simpler ones does not necessarily lead to easily predictable (or desirable) behaviour. Potential field methods can suffer from the presence of spurious minima in the potential function. This can lead to disappointing results, especially in the transient behaviour of systems subject to these potentials [22], [23], [14]. We note that, in a few instances, potential functions without spurious minima can be found, e.g., [17], [19], [20], [21]. However, many theoretical assumptions must ultimately be relaxed in a practical implementation.

Constraint programming methods [24], [25], [26], [27], [1], [28] have been proposed as an executable specification language for robot programming. These methods are intermediate-level languages, residing between low-level, trajectory-based approaches and high-level motion planning. Their goal is to offer a declarative, yet computationally tractable framework for the solution of such problems.

The mathematical models behind these methods are initial-value, (ordinary) Differential systems with Algebraic Inequality constraints (DAI). This acronym is motivated from the more popular class of problems known as Differential-Algebraic Equations (DAEs) [29], [30]. DAIs arise in many other applications as well, see [31]. Finding a feasible solution for a DAI corresponds to the viability problem (e.g. [32]) of a control system subject to a nonempty set of (user-defined) inequality constraints.

The purpose of this paper is to develop robust, efficient numerical methods for DAIs. Our strategy follows the principle that the control for a system should be chosen so that the boundaries of the viability region are avoided whenever possible. We put forth such a selection as a refinement to choosing a control based on viability alone.

In this paper, we consider control systems of the form

$$\begin{aligned} \dot{q} &= f(t, q(t)) + B(t, q(t))u(t), \qquad (1.1a)\\ &\quad 0 < t < t_f, \end{aligned}$$

where the state q and the control u belong to finite-

R. Spiteri is with the Department of Mathematics and Statistics, School of Computer Science, and Centre for Intelligent Machines, McGill University, Montréal, Québec, H3A 2A7, Canada. E-mail: spiteri@cim.mcgill.ca. The work of this author was partially supported under a CRM-ISM Postdoctoral Fellowship and NSERC Canada Grant OGP0004306.

D. Pai is with the Department of Computer Science, University of British Columbia, Vancouver, BC, V6T 1Z4, Canada. E-mail: pai@cs.ubc.ca. The work of this author was partially supported by grants from NSERC, IRIS, and BC ASI.

U. Ascher is with the Institute of Applied Mathematics and Department of Computer Science, University of British Columbia, Vancouver, BC, V6T 1Z4, Canada. E-mail: ascher@cs.ubc.ca. The work of this author was partially supported under NSERC Canada Grant OGP0004306.

dimensional vector spaces X and Z of dimension n and n_{cntrl}, respectively. The system is subject to the constraints

$$|u_j| \leq u_{\max,j}, \qquad j = 1, 2, \ldots, n_{\mathrm{cntrl}}, \tag{1.1b}$$

and

$$c_i(t, q) \geq 0,\ i = 1, 2, \ldots, n_{\mathrm{cnstr}}. \tag{1.1c}$$

The bounds on the control variables (1.1b) can represent, for example, physical limitations (saturation) of electromechanical actuators. Such restrictions take the form

$$u(t) \in U\,, \tag{1.2}$$

for some appropriate set U. More generally, when the right-hand side of the inclusion (1.2) is a function of the state q, U is a set-valued map from X to Z representing an a priori feedback that describes any state-dependent constraints on the controls.

We further define the *viability set* $K(t)$ for the control system (1.1) by

$$K(t) = \{q \in \Re^n \mid c_i(t, q) \geq 0,\ i = 1, 2, \ldots, n_{\mathrm{cnstr}}\}. \tag{1.3}$$

This is a closed subset of X. It is through the definition and manipulation of the viability set $K(t)$ that the robot program is specified [26]. This formulation represents a unified task-description and control approach to robot programming. For consistency, we assume that the system starts from a feasible point $q_0 \in K(0)$.

The problem (1.1) is a viability problem. Viability theory (see, for example, [32], [33], [34]) attempts to give conditions relating $K(t)$, U, and the dynamics in (1.1a), for which a trajectory to (1.1a) satisfies (1.2), (1.3) for $t \geq 0$.

The concept of solution that we adopt here is consistent with discrete sampling and is therefore the most natural in terms of application. Consider a partition of $[0, t_f]$:

$$0 = t_0 < t_1 < \ldots < t_N = t_f.$$

In many applications, the partition is uniform, with spacing $\Delta t = t_i - t_{i-1}$ representing the sampling interval. The trajectory associated with the feedback $\tilde{u}(q)$ defined on each subinterval $[t_{i-1}, t_i)$ is the solution with constant control $u_i = \tilde{u}(q(t_{i-1}))$. This interpretation for a solution is common in differential-game theory [35], [36]. The forward Euler discretization of such a strategy can lead to another interpretation of a solution to (1.1a) given a feedback control $\tilde{u}(q)$; however, this distinction is not important for the purposes of this paper. Thus, we define a solution to (1.1) as a piecewise control $u(t)$ such that $q(t) \in K(t)$ for all $t \in [0, t_f]$. The aim of the algorithm is to construct a solution (when such exists) to the viability problem (1.1) in an efficient and robust manner.

Solutions of (1.1) are generally *non-unique*. That is, there is usually a bundle of trajectories $q(t)$ that satisfy (1.1). We denote this set of all possible trajectories as Q_K, explicitly noting its dependence on the viability set.

The problem considered here may at first be thought of as intimately related to problems in optimal control, where one optimizes an objective function depending on q and u over the time horizon $[0, t_f]$, subject to a system like (1.1). Indeed, in the absence of an objective function, one can always construct one in order to find a viable solution (as, e.g., in the first phase of a two-phase Simplex algorithm for linear programming [37]). However, the process of solving optimal-control problems is slow and sometimes difficult. Part of the difficulty is that the solution process must be *global* in time, because changes in the data anywhere over the time horizon cause changes everywhere in the optimal solution. In the current setting (1.1) (given only initial, and not boundary data), it is possible to devise cheaper and more robust *local* solution processes, as for initial-value ODEs. Accordingly, in the programmed-constraints approach, one uses time-dependent, user-defined constraints to specify a target position for the robot, thus maintaining the local nature of the model.

The remainder of the paper is organized as follows. In §2, we present a brief description of constraint programming as applied to robotics. Relevant concepts from viability theory are also presented. In §3, we describe in detail a method for the numerical solution of a general DAI. This is followed in §4 by some theoretical results for the local control aspect of the proposed algorithm. We use simple test cases to show analytically that our control is consistent with basic physical intuition. Finally, in §5, we demonstrate the use of the algorithm in solving various DAIs associated with the problem of programming a mobile robot. Simulations show our solution method to be computationally efficient on a number of challenging problems. The complementarity and comparison of this approach to optimal control are discussed and demonstrated as well.

II. Constraint Programming

The Least Constraint approach [26] incorporates a *weaker specification* of control actions, utilizing large time-varying sets of nonzero measure (which we identify as *viability sets*) as the means by which desired goals are specified, in contrast to explicit trajectory specification. The underlying philosophy is that all locations in the viability set are equally acceptable, and it is often unnecessary and unnatural to specify more for a task to be completed. The viability sets are described by the (nonempty) intersection of inequality constraints that are satisfied at run time to produce the control. A solution to such a program necessarily produces behaviour that satisfies all the constraints.

Programmed constraints provide a unified approach to programming. The particular advantage to robot programming is the ease with which complex programs can be built up and modified [26], [25]. Constraints can easily be added or deleted as the requirements of the solution become clearer. These requirements may not be known a priori. Such an approach is also a powerful technique for non-traditional control problems such as minimally invasive surgery performed by teleoperated robots [27], [38].

In addition, the use of constraints for robot programming also offers some advantages that can be complementary to the more traditional approaches. For example, the simplicity and intuitive geometric appeal of explicit path specification can be retained by means of a (fictitious) moving circular 'spotlight' whose centre follows a predetermined path, gently restricting the motion of a mobile robot by requiring that it stay within the spotlight. The radius of the spotlight is a measure of the tolerance within which the path should be followed. There are several advantages to this approach. First, it is easier for the programmer to specify a geometric entity such as the path of the spotlight and allow the plant itself to (implicitly) solve the inverse dynamics problem of staying within it. Second, flexibility in avoiding obstacles or changing tasks is not sacrificed because the spotlight path can be changed at any time without adversely affecting the solution procedure[1]. In other words, new information can be processed to some extent during the motion. Third, this method also dispenses with the problem of being forced to choose a functional to optimize so as to apply an optimal-control approach. Often in robotics, as in everyday life, there is no clear choice of performance measure for a given task.

It is well known that there are processes for which optimal control is a natural (and sometimes even easily computable) control strategy, or where the robot configuration space is sufficiently complex that geometric intuition falters. In such cases, the DAI framework may be useful for programming in more of a complementary sense, rather than strictly as an alternative to other path-planning methods, for example, [39], [6], [19]; see §5.

A. Constraint Programming as a Viability Problem

Viability theory is a useful mathematical framework for describing the evolution of systems arising in many applications, in particular control systems. The main connections between DAIs and viability theory are threefold: First, viability theorems yield selection procedures for viable evolutions [32], [33]. That is, they characterize the relations between dynamics and constraints in order to guarantee the existence of a viable solution starting from any initial viable state. Second, similar to an initial-value DAI program, viability theory does not require optimization of an inter-temporal objective function. Rather, the theory allows for the possibility to adapt to a changing environment as defined by the viability constraints. Finally, as part of our solution procedure, we adopt from viability theory the use of a generalized *inertia principle*, which states that the controls are kept on a certain strategy, $u_{\text{nat}}(t)$, as long as the viability of the system is not at stake.

For a more detailed treatment of viability theory, see, for example, [32], [33].

[1]Even unexpected events, e.g., obstacles straying into the spotlight, are handled automatically, to within the limitations of the system.

III. A DAI Solver

We now describe in more detail the method used to produce a solution to the robot-programming problem defined by (1.1). An early version of the method appeared in [1]. Following an overview, we proceed with a detailed specification of the various components of the algorithm. We then conclude with a summary.

At each step of the DAI integration, a local selection is made for the control $u(t)$, based only on 'current' information about the system and the viability set $K(t)$. The default control $u_{\text{nat}}(t)$ is maintained as long as the viability of the solution is not in jeopardy. When the system is deemed too close to the boundary $\partial K(t)$ by a local prediction (or anticipation) strategy, a (new) control $u^*(t)$ is invoked. The anticipation process involves not only monitoring the constraints and their various rates of change, but also whether a sufficiently small deviation of the control from u_{nat} will allow the system to avoid each constraint boundary *individually*, though perhaps not simultaneously.

Desirable features of a solution method include:

- *Small computational expense* to determine the control. This is especially important in real-time applications.
- *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, modelling, and finding the precise value of the locally optimal control.

Although we do not wish to solve a global motion-planning problem, a completely local approach will be severely limited in the scope of problems that can be solved. Define the *K-reachable* set at time t as

$$\begin{aligned} R_K(q_0,t) \quad = \quad & \{q(t) \in \Re^n,\ t \in [0,t_f] \mid q(\cdot) \in Q_K \\ & \text{with } q(0) = q_0\}. \end{aligned}$$

It is the usual reachable set, but with the added restriction of viability – hence, its dependence on K. Thus, as part of a planning procedure, an anticipatory or *buffer* zone near the boundary of the feasibility region must be created and maintained in order to give any algorithm the opportunity to choose a control (or sequence of controls, e.g., at consecutive time steps) to keep the system in the viability region.

The values for the controls are computed based on the approximate minimization of a $\mathcal{C}^1$ logarithmic artificial barrier potential, where the singularities are aligned with $\partial K(t)$. In this way, future states that continue to approach the boundary are penalized.

The algorithm can be divided into three basic components: time discretization, local control, and local planning. A detailed treatment of local control and local planning is given below. For the time discretization, we use an explicit, delay-free, one-step method; we refer to [40], [31] for a more extensive treatment.

A. Local Control

This component of the algorithm finds a control that differs locally from the inertial behaviour u_{nat} when the viability of the system is at stake.

Suppose that the system is in a viable state q_{n-1} at time t_{n-1}, and that an update to $u_{\text{nat}}(t_{n-1})$ is deemed necessary. We delineate a buffer-zone edge by assigning constants

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

We obtain a new value for the control u^* as an (approximate) solution to the following constrained nonlinear programming problem:

$$u^* \in \arg\min_{u \in U} \phi(u) \tag{3.1a}$$

such that

$$c_i(t_n, q_n(u)) \geq 0, \tag{3.1b}$$

where

$$\phi(u) = \sum_{i=1}^{n_{\text{cnstr}}} \phi_i(t_n, q_n(u)), \tag{3.1c}$$

$$\phi_i = \begin{cases} \frac{c_i}{c_{\text{safe},i}} - 1 - \log\left(\frac{c_i}{c_{\text{safe},i}}\right) & c_i < c_{\text{safe},i} \\ 0 & c_i \geq c_{\text{safe},i}. \end{cases} \tag{3.1d}$$

We have chosen a barrier that not only disallows infeasible solutions, but also serves to keep the system away from the boundaries. We have chosen to use logarithmic barrier functions; however, other choices are possible [41], [14]. For example, the original potential proposed in [14] varies inversely with the square of the distance to the obstacle. Further, we use the potential function to search for a viable solution, in contrast to [14], where the gradient of the potential is interpreted as an applied force.

The particular choice of the barrier function (3.1d) was made as a deterrent to a 'bang-bang' choice for the control when moving the system away from the boundary. In this way, the previous stability of the system evolving with a natural value for the control is not excessively undermined. The advantage of logarithmic barriers lies in their relatively slow divergence near the singularity. The logarithmic terms are augmented by terms to make ϕ a $\mathcal{C}^1$-function. Moreover, (3.1d) has some nice theoretical properties, such as convexity, that produce intuitively correct controls in the limit of continuous observations (see §4).

The problem (3.1) is solved by means of a penalty function method, that is, through the solution of a sequence of unconstrained minimization problems where the objective function ϕ is augmented with a penalty term [37]. The strategy adopted for use in practice is taken from [42], [37], [43], although steps are taken so that an approximate solution to (3.1) is found quickly, sometimes at the expense of accuracy. Due to the local nature of the problem, highly accurate solutions offer little advantage. A more detailed description of the minimization algorithm appears in [31].

If a solution u^* to (3.1) is found, the system (1.1a) is advanced one time step, starting from (t_{n-1}, q_{n-1}). In the absence of discretization or modelling errors, the existence of a u^* guarantees that the system will be viable at the next step. We then return to the inertial control u_{nat} (at least initially) as the control to try for the next time step.

B. Local Planning

The minimization process just described is relatively expensive computationally; so reducing the number of steps in which (3.1) is invoked leads to a more efficient method. Moreover, typical results from viability theory or differential games do not offer a strategy for control until the boundary of the viability set is reached [32], [35], [34]. By then, depending on the form of the control system (1.1), the intersection of the K-reachable set and the viability set may be empty. We propose to monitor the relative movement of the boundaries to decide whether they can be *easily* avoided. If not, then the system should act immediately towards improving upon this. To determine when a control step should be executed, we form a local approximation to the current behaviour of the constraints and perform a test on whether each constraint can be avoided individually by a control that is sufficiently close to u_{nat}. This is referred to as *weak approximate safety.* The size of this deviation is governed by sensitivity factors which we write as a diagonal matrix U_{frac} with nonzero entries $u_{\text{frac},j}$, $j = 1, 2, \ldots, n_{\text{cntrl}}$. These values represent the size of a relative deviation that is deemed to be 'small'. Possible choices for u_{nat} will be illustrated in the examples section.

The prediction strategy is implemented through the use of buffer zones. Let the positive quantities $c_{\text{safe},i}$, $i = 1, 2, \ldots, n_{\text{cnstr}}$, 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 such that $q \in R_K(q_0, t_{n-1})$ and

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

We generalize the concept of an active set to refer to the set of those constraints whose buffer zone is currently being penetrated by the system.

Consider Figure 3.1. Let the system be in a viable configuration q_{n-1} at time $t = t_{n-1}$. A strategy based on viability alone (or no knowledge of the future) would be to simply maintain the schedule $u_{\text{nat}}(t)$ until the system reaches the boundary (Figure 3.1(a)). Clearly, there are limitations to the robustness of such a strategy. Our prediction strategy involves using time-derivative information of the constraints. If $\dot{c} \geq 0$ at (t_{n-1}, q_{n-1}), we continue normal integration with $u(t) = u_{\text{nat}}(t)$, because the system is not moving locally towards the boundary of the feasibility region (Figure 3.1(b)). The system will be viable using $u_{\text{nat}}(t)$ under a linear model for the evolution of the constraint equations. If for some i, $\dot{c}_i < 0$, we compute $\ddot{c}$ and form the discriminant Δ with components defined by

$$\Delta_i = 2c_i\ddot{c}_i - \dot{c}_i^2 \quad i = 1, 2, \ldots, n_{\text{cnstr}}. \tag{3.2}$$

If $\Delta > 0$, then the quadratics given by

$$c + \dot{c}(t - t_{n-1}) + \frac{1}{2}\ddot{c}(t - t_{n-1})^2, \qquad t > t_{n-1},$$

have no real roots and hence *locally* we deem that constraint violation will not occur based on a second-order

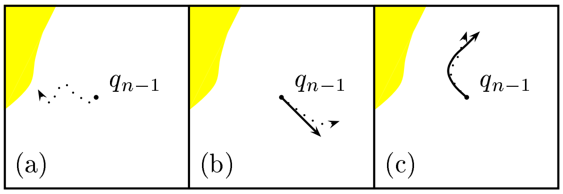

Fig. 3.1. Summary of prediction strategy.

model for the evolution of the constraint equations (Figure 3.1(c)). If $\Delta_i \leq 0$, this indicates that an impact with the boundary of c_i is likely to occur, if $\dot{c}_i, \ddot{c}_i$ remain relatively constant and $u(t) = u_{\text{nat}}(t)$. Of course, this is not a good long-time predictor, because $\dot{c}_i, \ddot{c}_i$ are not generally constant. However, these are reasonable local-time predictors, and they can be useful over the entire interval of integration provided they are updated with sufficient frequency.

However, in many common multi-body and robotics systems, constraints occur in opposing pairs (so-called *box constraints* provide a typical example). 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 $\Delta > 0$ can never be satisfied.

Our strategy, then, is to recompute $\dot{c}_2$, $\ddot{c}_2$ with a control having components $u_{\text{nat}} \pm \Delta u$ where Δu is a vector with components $\Delta u_j = u_{\text{frac},j} u_{\max,j}$ and

$$u_{\text{frac},j} \in \left(0, \frac{|u_{\text{nat}\,j} - u_{\max,j}|}{u_{\max,j}}\right), \; j = 1, 2, \ldots, n_{\text{cntrl}}.$$

If either (b) or (c) in Figure 3.1 holds for all c_i individually, then we have weak approximate safety. Such a strategy is not difficult to implement in practice.

If the condition of weak approximate safety holds, then the integration continues with $u = u_{\text{nat}}$. Thus far, we have found this test to be of practical utility. The quantity U_{frac} allows for errors in the model, the constraint measurements, or predictions and potential inconsistencies therein. It can be considered a sensitivity parameter, initially specified by the user, but then dynamically altered. We refer to [1], [31] for more details of how U_{frac} is tuned.

C. Summary

The various components of the algorithm are combined as follows. The control function $u(t)$ is taken to be piecewise constant on each interval $[t_{n-1}, t_n)$. Without loss of generality, we assume that the buffer zone is not activated. Starting from a viable state q_{n-1} at time t_{n-1}, and a sensitivity parameter U_{frac}:

1. Predict safety of using $u_{n-1} = u_{\text{nat}}(t_{n-1})$.
2. If *weak approximate safety* then
   (a) Attempt integration of (1.1a) over time interval $[t_{n-1}, t_n)$ using default control $u_{n-1} = u_{\text{nat}}(t_{n-1})$.
   (b) If resulting state q_n is viable, then (successful step)
      i. Accept u_{n-1}, q_n;
      ii. Increment n;
      iii. Break to Step 7;
      else (unexpected constraint violation) Reduce U_{frac}.
3. If buffer zone is not activated, then activate it by setting $c_{\text{safe}} = c(t_{n-1}, q_{n-1})$.
4. Solve the barrier minimization problem (3.1) to obtain a control $u^*(t_{n-1})$.
5. Attempt integration of (1.1a) over $[t_{n-1}, t_n)$ using the updated control $u_{n-1} = u^*(t_{n-1})$.
6. If q_n is viable, then
   (a) Accept u_{n-1}, q_n;
   (b) Increment n;
   (c) For each j such that $u_j = u_{\max,j}$, reduce $u_{\text{frac},j}$;
   (d) If $u_{n-1} = u_{\text{nat}}(t_{n-1})$, increase U_{frac};
   else (failure) System dies.
7. If $u_{n-1} = u_{\text{nat}}(t_{n-1})$, deactivate buffer zone.

IV. Theoretical Considerations

We now present some theoretical results for the algorithm, focusing on the control obtained from the barrier function in certain identifiable situations. We find that the control produced maintains the *relative* position of the system within the viability set from one time step to the next. These results are given for model systems with velocity and acceleration controls in various viability regions, such as n-dimensional rectangular or spherical spotlights and shrinking regions. In this section, we sometimes suppress the additional dependence on q_n and t_n in the notation of (3.1b) and write simply $c_i = c_i(u)$.

We begin with a statement of some convexity properties of the artificial potential function. Convexity is a very useful property because it aids the efficiency and reliability of the minimization procedure.

Theorem 4.1 (Convexity of Barrier) If the constraints $c_i(u)$ are linear in u for $i = 1, 2, \ldots, n_{\text{cnstr}}$, then the barrier function (3.1d) is convex.

The proof of this theorem can be found in [31].

Next, in order to better understand the behaviour of the algorithm proposed in §3, we examine what control is computed upon minimization of (3.1) for several simple model problems. For simplicity, we do not include the effect of a prediction strategy.

Model Problems

1. Consider the initial-value control system

$$\dot{x} = u, \tag{4.1a}$$
$$\|u\|_\infty \leq u_{\max}, \tag{4.1b}$$

subject to

$$-r \leq x(t) - x_c(t) \leq r, \tag{4.1c}$$

where $x(t), u(t), x_c(t), r \in \Re^n$. This is the n-dimensional analogue of a moving rectangular spotlight. The objective is to choose a control u such that each component of the system $x_i(t)$ remains within a distance r_i from a moving point $x_{c,i}(t)$. In the limit $\Delta t \to 0^+$, the control strategy obtained from solving (3.1) on such an interval is

$$u_i(t) = \begin{cases} \dot{x}_{c,i}(t) & |\dot{x}_{c,i}(t)| < u_{\max} \\ \operatorname{sgn}(\dot{x}_{c,i}(t)) u_{\max} & \text{otherwise.} \end{cases}$$

Thus, the control obtained from (3.1) matches the velocity of the system with that of the centre of the spotlight as closely as possible. If $\| \dot{x}_c \|_\infty \le u_{\max}$, the strategy always yields a globally viable solution. Note that a condition such as $\| \dot{x}_c \|_\infty \le u_{\max}$ can be enforced by the programmer,[2] and hence represents a guide in designing the program. When $|x_i(t) - x_{c,i}(t)| = r_i$, it is possible to interpret this solution as a *heavy* solution, i.e., the solution that uses the control of smallest size to preserve viability. □

2. In a similar fashion to the previous example, consider now (4.1) with the ODE replaced by

$$\ddot{x} = u. \tag{4.2}$$

Here, each component of the control obtained from the minimization (3.1) uses at first a maximal acceleration to match the velocity of the system to that of the spotlight centre. After this is achieved, it follows the spotlight acceleration. For appropriate matching times τ_i,

$$u_i(t) = \begin{cases} \operatorname{sgn}(\dot{x}_{c,i}(t) - v_i(t)) u_{\max} & 0 \le t \le \tau_i \\ \ddot{x}_{c,i}(t) & t > \tau_i. \end{cases}$$

It can be shown that, on each interval of length t where the spotlight acceleration $\ddot{x}_c(t)$ is constant and satisfies $\| \ddot{x}_c(t) \|_\infty \le u_{\max}$, the relative position of the system within the spotlight is constant. □

In a practical setting where the sampling time cannot be made arbitrarily small, this means that the *relative position* of the system within the spotlight is preserved from mesh point to mesh point while the velocity matching takes place. Thus, as is common with optimization codes that solve discretizations of continuous problems subject to state-dependent inequality constraints, viability can only be guaranteed at mesh points. The possibility of infeasibility in between mesh points exists. However, this can generally be mitigated by a suitable mesh refinement.

Similar results are obtained [31] for the damped system

$$\ddot{x} = u - \gamma \dot{x}.$$

3. Consider now the control system (4.1a) with the spotlight radius shrinking in time: $r = r(t)$ and $\dot{r} < 0$. Typically, the program ends with $r(t_f) = \epsilon > 0$, where ϵ is a vector of tolerances. In this case, it can be shown [31] that the control is chosen so that the ratios of the constraint values are preserved at the end of the integration step. □

Although these three examples may seem simple, they have been used in more sophisticated applications. For example, the control system (4.2) has been used in [45] to describe the open-chain manipulator dynamics of a SIGMA robot for control in a time-optimal contour-following task. In addition, the forms of some models presented here with velocity controls are special cases of the canonical form of non-holonomic control systems (e.g., [38], [45], [46]):

$$\dot{q} = \sum_{i=1}^{n_{\text{cntrl}}} g_i(q) u_i.$$

We note that the right-hand side of this equation is *linear* in the control variables u. Hence, the hypothesis of Theorem 4.1 can be satisfied for constraints $c(t, q)$ that are linear functions of the state.

4. Our final test case involves an n-dimensional 'spherical' spotlight. Because the constraints c are no longer linear, the control defined by our algorithm is not unique. However, the minima are not spurious, but rather multiple, because each of them yields an equally acceptable solution. In practice, the choice of solution can be influenced by the default control $u_{\text{nat}}(t)$ specified by the programmer.

Consider the control system (4.1a) required to move within an n-dimensional spherical spotlight with centre at $(x_c(t), y_c(t))$ and fixed radius r. The solution u that yields a minimum of (3.1) can be shown to satisfy

$$\sum_{i=1}^{n} (\tilde{x}_i(t; u) - x_{c,i}(t))^2 = \sum_{i=1}^{n} (x_{0,i} - x_{c,i}(0))^2, \tag{4.3}$$

and hence we see again that the control attempts to preserve the relative distance from the centre of the spotlight from one time step to the next.

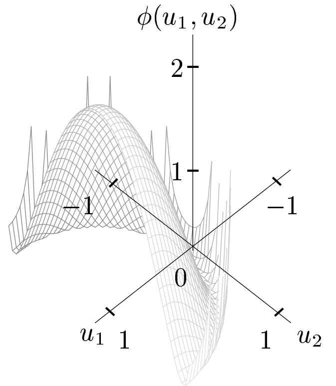

Fig. 4.1. Barrier function for two-dimensional circular spotlight.

To obtain some geometric intuition, consider the case for $n = 2$. Let the (circular) spotlight have unit radius and initial position $(x_{c,1}(0), x_{c,2}(0)) = (0.9, 0)$ and assume its position at the next sampling time Δt is $(x_{c,1}(\Delta t), x_{c,2}(\Delta t)) = (1, 0)$. Let the initial position of the system be $(x_1(0), x_2(0)) = (0, 0)$ and let $u_{\max} = 1$. Figure 4.1 gives a surface plot of the barrier (3.1d) in this case. From this plot, we can see how the set of minimizers satisfies a relation such as equation (4.3). □

[2] This is not a differential game where the motion of the spotlight may be governed by an evil adversary [36], [35], [44].

V. Numerical Experiments

In this section, numerical simulation is employed to illustrate the general applicability of the solution procedure described and analyzed in the previous sections, the types of solutions that are produced, and the relative computational efficiency. We give some results from simulations of a mobile-robot system under the action of various programs. We then proceed to combine the programmed-constraints approach with optimal control, showing the complementarity of the two approaches.

A. Mobile Robot Simulation

We have performed numerical experiments based on a model of the Dynamite multiple mobile robot testbed [47], [48]. Let the robot (which is the size of a small toy car) be located with its centroid at (x, y), oriented at an angle θ measured clockwise from the positive x-axis, and be moving with speed v. The controls are the normalized settings for the acceleration $\tilde{a}$ and the steering ψ, i.e., ψ is the angle that the wheels make with respect to the car's body [1]. With L, β, γ, and κ given parameters representing car length, steering gain, damping factor, and acceleration gain, respectively, let $\alpha = \frac{1}{L}\tan(\beta\psi)$ and $a = \kappa\tilde{a}$ be the new controls[3]. Thus, the state equations of the mobile robot that are used in our simulations are

$$\begin{aligned} \dot{x} &= v\cos\theta && (5.1a)\\ \dot{y} &= v\sin\theta && (5.1b)\\ \dot{\theta} &= v\alpha && (5.1c)\\ \dot{v} &= a - \gamma v. && (5.1d) \end{aligned}$$

This system corresponds to (1.1a). The bounds on the controls, corresponding to (1.1b), are

$$|a| \le 150 \text{ cm s}^{-2}, \quad |\alpha| \le \frac{1}{25} \text{ cm}^{-1}. \tag{5.2}$$

The first bound represents the maximum allowable throttle setting and the second yields a minimum turning radius of 25 cm. We take $\gamma = 1/6$.

For the viability set $K(t)$ defined by (1.1c), we have two types of constraints: static bounds on x, y and v,

$$\begin{gathered} x_{\min} \le x \le x_{\max}, \qquad y_{\min} \le y \le y_{\max}, \\ -v_{\max} \le v \le v_{\max}, \end{gathered} \tag{5.3}$$

and time-dependent, driving constraints, describing the robot's task. The latter type will be specified below, when we consider different special cases.

We utilize two possible choices for the default control, $u_{\text{nat}}(t) \equiv (a_{\text{nat}}(t), \alpha_{\text{nat}}(t))$. In both cases, the natural steering angle is 0, but they differ in the choice of natural acceleration. The choices are classified according to an effective damping factor $\tilde{\gamma}$. The first default control is $u_{\text{nat}}(t) \equiv 0$ and corresponds to damped behaviour with $\tilde{\gamma} = \gamma$. Because $\tilde{\gamma} > 0$, the robot slows down while maintaining the same heading θ when the system is safely inside $K(t)$. The second choice for default control is $a_{\text{nat}}(t) = \gamma v(t)$. This corresponds to a cruising option: The natural motion is with constant velocity and $\tilde{\gamma} = 0$. This is a more aggressive strategy. However, it is no more expensive to compute with in principle, and in fact it may lead to efficient solutions in case that it ultimately produces fewer interactions with the boundary of $K(t)$.

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 step-size is $\Delta t = \frac{1}{60}$ seconds, representing the actual sampling time of the Dynamite system [47]. A first-order integration scheme was chosen due to the low expected accuracy from simplifications made in the model equations.

Case 5.1 (Robot in a Spotlight)

In this example, the objective is to move the robot to a corner of the table while in the neighbourhood of a specified path. Hence, there is a geometric appeal to the programmer, who may have an idea of the desired path but does not wish to be involved in solving inverse dynamics problems. The ensuing DAI is easily specified and efficiently solved by our method.

We use a parabolic spotlight path that passes through $(x(0), y(0)) = (5.27, 50)$, $(x_{\min}, y(0))$ and $(x_{\max}, y_{\max})$. The path is parameterized as

$$x_c(t) = v_w t + x(0), \quad y_c(t) = ax^2 + bx + d, \tag{5.4}$$

where v_w is the x-component of the spotlight velocity and a, b, d are constants. The constraint that complements (5.3) in specifying $K(t)$ is then

$$(x(t) - x_c(t))^2 + (y(t) - y_c(t))^2 \le r_c^2. \tag{5.5}$$

We choose $v_w = 5, r_c = 5$, and $t_f = (x_{\max} - x_{\min})/v_w$. For this example, it was possible to produce qualitatively different solutions, depending on the combination of choices for $u_{\text{nat}}(t)$ and U_{frac}. Typical examples of these are shown in Figures 5.1, 5.2 [4]. The solution in Figure 5.1 looks qualitatively similar to the heavy-looking solutions that are normally produced by an active-set method: the robot slides along the 'trailing' part of the boundary, as if being dragged

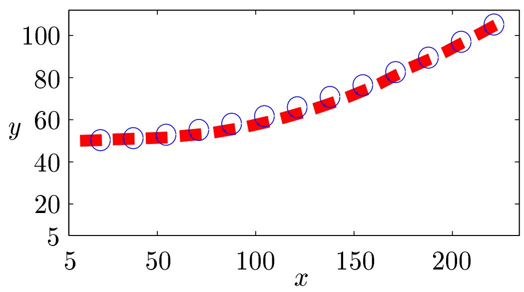

Fig. 5.1. Solution exhibiting heavy behaviour.

[3]Such a redefinition of controls is possible since ψ and α are in a one-to-one correspondence in the allowable range of ψ.

[4]Note that the constraints are defined with respect to the coordinates of the centre. The fact that the outline of the car itself may sometimes fall outside of the spotlight is not relevant.

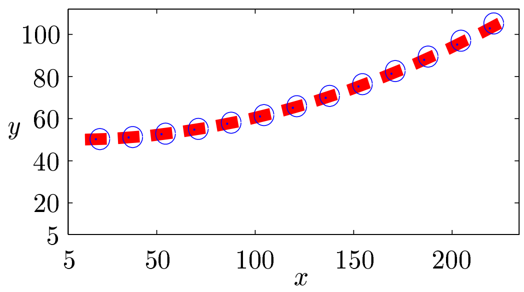

Fig. 5.2. Solution not exhibiting heavy behaviour.

along by the spotlight. On the other hand, in Figure 5.2, the robot stays very close to the centre for most of the time, right up until the final moments of the simulation.

Because the speed of the spotlight is modest, and the starting values of U_{frac} are identical, we can attribute the difference in behaviours to the different choices of $u_{\text{nat}}(t)$: the cruise option invokes the barrier control less often, thus allowing the car centre to approach the boundary of the spotlight more readily.

Case 5.2 (The Parallel-Parking Problem)

We now illustrate another use of DAIs to solve the familiar problem of parallel parking. In this example, the robot must move to a position between two other stationary robots, while not running up the curb. All robots have length 11.5 cm and width 8cm. The initial starting configuration of the parking robot is $(10\sqrt{15}, 10, 0, 0))$, and the parked robots have their centres at positions $(-15.75, 0)$ and $(35.75, 0)$, respectively, also with orientation 0. We note that the constraints defined by the parked cars are not activated until the moving car is somewhere between them. A spotlight path is chosen to connect the starting point with the desired point $(0, 0)$. The final desired orientation of 0 is encouraged by making the path have zero slope at the destination point. The curb is given by the equation $y = -5$. Taking the radius of the spotlight to be 1 cm and its nominal speed along the x-axis to be -1 cm s^{-1}, the trajectory of robot performing the parallel parking maneuver is shown in Figure 5.3.

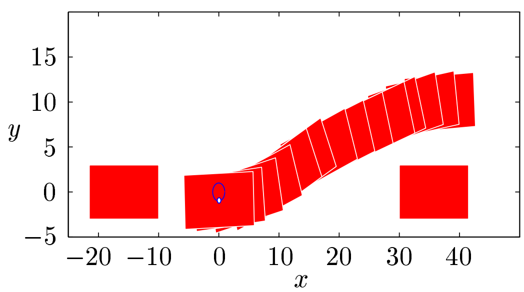

Fig. 5.3. A Parallel-Parking Maneuvre.

Case 5.3 (Connection with Optimal Control)

In this example, we discuss some uses of the programmed-constraints approach in conjunction with an optimal-control solver. We show some complementarity between the two approaches, in particular, that the DAI solver presented in §3 can be a very inexpensive starter for an optimal-control routine.

The optimal-control code used is OCPRSQP by V. Schulz [49], which solves optimal-control problems in boundary-value DAEs with state- and control-dependent constraints of the form:

$$\min_{x,y,u} \int_{t_0}^{t_f} L(x(t), y(t), u(t))\, dt, \tag{5.6a}$$

subject to

$$\begin{pmatrix} \dot{x}(t) \\ 0 \end{pmatrix} = f(x(t), y(t), u(t)), \tag{5.6b}$$

$$u_{\min} \leq u(t) \leq u_{\max}, \tag{5.6c}$$

$$c^i_{\min} \leq c^i(x, y, u) \leq c^i_{\max}. \tag{5.6d}$$

The boundary-value problem is discretized by collocation on a mesh

$$t_0 < t_1 < \ldots < t_N = t_f$$

and the resulting finite-dimensional optimization problem is solved by partially reduced sequential quadratic programming methods [49]. These methods are intended to combine the convenient treatment of inequality constraints afforded by sequential quadratic programming and the smaller quadratic subproblems offered by reduced sequential quadratic programming, allowing the optimization problem (5.6) to be solved very efficiently.

We consider the minimum-time problem

$$\min_{q,u} \int_0^{t_f} dt,$$

subject to (5.1) – (5.3). In order to make the exact solution to the optimization problem more intuitive, only position constraints are imposed (so, $v_{\max} = \infty$). The boundary conditions for the optimal-control programs are

$$\begin{aligned} q(0) &= \left(\tfrac{1}{10}x_{\max}, \tfrac{1}{2}y_{\max}, 0, 0\right)^T, \\ q(t_f) &= \left(\tfrac{9}{10}x_{\max}, \tfrac{1}{2}y_{\max}, 0 \bmod 2\pi, 0\right)^T, \end{aligned}$$

with the values for (5.3)

$$x_{\min} = y_{\min} = 0, \; x_{\max} = 200, \text{ and } y_{\max} = 100.$$

We will refer to the initial (x, y)-coordinates as A and to the final (x, y)-coordinates (x_f, y_f) as B.

The programmed-constraints formulation shares the same initial conditions, but the terminal conditions can only be enforced to within some tolerance. However, this is not a concern when constructing an initial guess because an optimal trajectory is not necessarily 'close' to one that is produced from a DAI. The two trajectories only need to be close enough to enable the nonlinear solver to converge.

We now give a few technical details for the optimal-control simulations. All simulations in this section were run on an SGI Indigo 2 having 64 Mb of RAM and a clock speed of 200 MHz. Three-stage Gauss collocation is employed as the discretization scheme, on an adaptive mesh having 40 subintervals initially. The controls u are taken to be piecewise constant, generally on a coarser mesh than the state variables. Convergence is deemed to have occurred based on an expected decrease, TOL, in the Powell merit function [49]. We pay particular attention to the effects on the numerical solution of decreasing this tolerance from 1.0×10^{-3} to 1.0×10^{-6}. Note that the particular optimal trajectory achieved depends critically on the initial guess. The addition of an objective function to a basic viability problem already adds significantly to the computational effort, and this effort would be severely compounded further if an attempt is made to also switch to a global optimization method, such as simulated annealing [50].

We simulate an experiment involving navigation in the presence of circular obstacles with constraints of the form

$$(x(t) - x_{\text{obs}})^2 + (y(t) - y_{\text{obs}})^2 \geq r_{\text{obs}}^2.$$

This is a prototypical application for a robot in a hazardous environment [38]. Other applications can be found in [31].

We now demonstrate the use of DAI as an initial guess for a problem with multiple obstacles. The arena is occupied by five obstacles of random size and position as depicted in Figure 5.4. In this case, the geometric solution to the optimal-control problem is fairly clear; however, the values for the other unknowns such as control profile or minimum time are less clear. We would like a low-effort way to produce a good initial guess for the optimal-control solver. It should be inexpensive in terms of both computation time and programmer effort. The easiest guess would be to just linearly interpolate between start and end points (despite parts of the path being infeasible) and choose some arbitrary control (for example, zero). This clearly satisfies both criteria of low effort. Unfortunately, the nonlinear equation solver of the optimal-control code was not able to converge from such a poor initial guess.

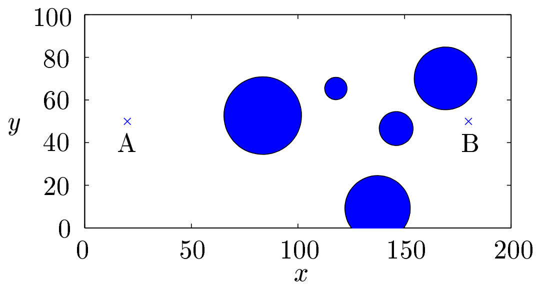

Fig. 5.4. Location of Multiple Obstacles.

An initial guess based on a shooting-type approach with an open-loop control strategy satisfies the criterion of low computational expense because the computational effort boils down to an explicit, initial-value ODE solver. It also has the advantage of being able to produce a consistent, feasible guess. However, the programmer will generally need to perform a large number of iterations on the control profile before a solution that comes reasonably close to satisfying the terminal boundary conditions is found. Moreover, if these terminal boundary conditions are changed, obstacles become non-stationary, or more obstacles are added (or discovered), the researcher would find that there is essentially no return on the time invested in the initial program! The initial guess obtained via shooting that is used in the computations to follow is shown in Figure 5.5. Some effort was put into making this guess reasonably good so that it would not unduly affect the convergence of OCPRSQP.

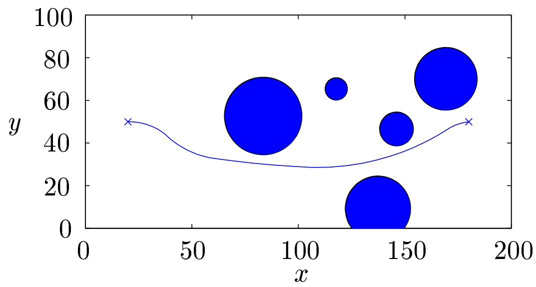

Fig. 5.5. Initial Guess from Shooting for Multi-Obstacle Problem.

A DAI program that forces the robot to end up in a neighbourhood of the desired terminal conditions can be constructed as follows. In addition to the position constraints from (5.3), we introduce two new pairs of constraints (see Figure 5.6):

1. A pair of constraints in the form of two travelling waves that uniformly 'squeeze' the viability region down to a shape with size δ in the x-direction at time $t = t_f$. We take $\delta = 5$ and choose $t_f = 2.25$ to be an estimate of the optimal time. The constraints themselves are then

$$x - v_1 t \geq 0, \quad x - v_2 t \leq x_{\max},$$

where v_1, v_2 are chosen such that the waves are located at $x = x_{\min} = 0$ and $x = x_{\max}$ at $t = 0$, and $x = x_f - \frac{1}{2}\delta$ and $x = x_f + \frac{1}{2}\delta$ at time $t = t_f$.
2. A pair of constraints in the form of artificial, stationary edges, that 'funnel' [26] the solution into a neighbourhood of B. Explicitly, these are

$$\tfrac{1}{2}x - 50 \leq y \leq -\tfrac{1}{2}x + 150.$$

We note that the program is easily modified to accommodate different boundary conditions, hence the time invested in constructing the first program is not entirely lost.

The initial guess produced by this program is depicted in Figure 5.6. Also shown are the optimal-control solutions for two different tolerances.

The following table shows the relative computation times to produce an optimal solution for different tolerances. The

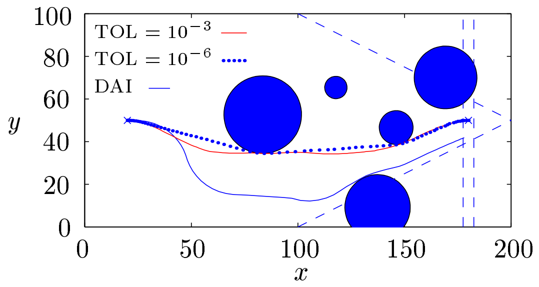

Fig. 5.6. DAI and Optimal Trajectories for Multi-Obstacle Problem.

first uses an initial guess from the DAI program and is listed under the column labelled (CPU1). The second uses an initial guess obtained from shooting (CPU2). The times quoted are in seconds.

TOL	CPU1	CPU2
10^{-3}	52.26	5.57
10^{-4}	122.82	126.97
10^{-5}	118.14	105.75
10^{-6}	109.70	83.69

TABLE 5.1
Optimal-control solutions using initial guesses obtained from shooting and DAI methods.

Comparing the results in this particular example, we see that the savings in computation time are not great when starting from a very accurate initial guess obtained from shooting. However, the savings in terms of programming time and program flexibility are significant. It takes relatively little effort to construct the program and its solution. The initial guess obtained by means of a DAI was produced in 0.51 CPU seconds, or about 0.6% of the time required for the optimal solution with TOL $= 10^{-6}$. Moreover, the DAI approach is more robust with respect to changes in the programming task or environment. In such cases, the programmer is likely not to need to build a new DAI for each small change.

Discussion. It should be pointed out that for a given task, some program formulations will work better than others. For example, if there are non-convex objects (in particular, those that cannot be conveniently enclosed by their convex hull), then it can be that a spotlight formulation will succeed more easily than a formulation using plane waves. We have also mentioned the possibility of complex configuration spaces impairing geometric intuition. In this case, it is possible to use a DAI program in conjunction with a path planner, such as [39], [7], [51], [18]. However, it is not the goal of the programmer to intentionally mislead the robot; so, in many cases, constraint programming provides enough flexibility to ultimately handle most tasks.

VI. Conclusions

In this paper, we have described a numerical method for solving the systems of differential-algebraic inequalities that arise from the programmed-constraints approach to robot programming. The programmed-constraints approach is appealing because programs are often geometrically motivated and intuitive. Moreover, programs can be developed incrementally. However, the range of possible tasks and desiderata for a robot program is so diverse that no single method can be the most appropriate in all cases.

We have also shown how an efficient solution to the DAI formulation of more traditional robot programs can complement existing approaches, in particular as an effective seed for optimal control. Generally, there is no strict connection between the DAI and optimal-control solutions; however, DAIs are easy to program and their solution adds little computational expense to the optimal-control calculation. Thus, they provide a flexible alternative to shooting and a more consistent framework than traditional holonomic motion planners.

The view of constructing a solution to a viability problem has led to a greater understanding of the problem and what it is that a solution method should try to do. We have shown how the algorithm presented is a nonstandard selection method based on a nonstandard optimization criterion where the relative position within the viability region is maintained from one time step to the next.

VII. Acknowledgements

The authors wish to thank Philip Loewen for suggesting the use of a $\mathcal{C}^1$ barrier function and the connection of viability theory with our work. We also wish to thank Georg Bock for noting the connection with differential games, and the reviewers for their thoughtful comments.

References

[1] R. Spiteri, U. Ascher, and D. Pai, "Numerical solution of differential systems with algebraic inequalities arising in robot programming", in *Proceedings of the IEEE Conference on Robotics and Automation*, Nagoya, 1995, pp. 2373–2380.
[2] J.C. Latombe, *Robot Motion Planning*, Kluwer, 1991.
[3] Zexiang Li and J.F. Canny, *Nonholonomic Motion Planning*, Kluwer Academic, Norwell, Massachusetts, 1993.
[4] John Canny, Ashutosh Rege, and John Reif, "An exact algorithm for kinodynamic planning in the plane", *Discrete Comput. Geom.*, vol. 6, no. 5, pp. 461–484, 1991.
[5] Bruce Donald, Patrick Xavier, John Canny, and John Reif, "Kinodynamic motion planning", *J. Assoc. Comput. Mach.*, vol. 40, no. 5, pp. 1048–1066, 1993.
[6] B. R. Donald and P. Xavier, "Provably good approximation algorithms for optimal kinodynamic planning: robots with decoupled dynamics bounds", *Algorithmica*, vol. 14, no. 6, pp. 443–479, 1995.
[7] B. R. Donald and P. G. Xavier, "Provably good approximation algorithms for optimal kinodynamic planning for Cartesian robots and open-chain manipulators", *Algorithmica*, vol. 14, no. 6, pp. 480–530, 1995.
[8] R. Brockett, *Control Theory and singular Riemannian geometry (eds. P. Hilton and G. Young)*, pp. 11–27, Springer Verlag, 1981.
[9] A.M. Bloch and N.H. McClamroch, "Control of mechanical systems with classical nonholonomic constraints", in *IEEE Conference on Decision and Control*, Tampa, 1989, pp. 201–205.
[10] G. Laffarriere and H.J. Sussmann, "Motion planning for completely controllable systems without drift", in *IEEE Conference*

on Robotics and Automation, Sacramento, CA, 1991, pp. 1148–1153.

[11] R.M. Murray and S.S. Sastry, "Steering nonholonomic systems using sinusoids", in *IEEE Conference on Decision and Control*, 1990.

[12] P. Jacobs, J.P. Laumond, and M. Taix, "Efficient motion planners for nonholonomic mobile robots", in *IEEE IROS*, Osaka, November 1991.

[13] Henry Tominaga and Behnam Bavarian, "Solving the moving obstacle path planning problem using embedded variational methods", in *IEEE International Conference on Robotics and Automation*, Sacramento, April 1991, pp. 450–455.

[14] O. Khatib, "Real time obstacle avoidance for manipulators and mobile robots", *The International Journal of Robotics Research*, vol. 5, no. 1, pp. 90–99, 1986.

[15] N. Hogan, "Impedance control: An approach to manipulation", *ASME Journal of Dynamics Systems*, vol. 107, pp. 1–7, March 1965.

[16] W.S. Newman and N. Hogan, "High speed robot control and obstacle avoidance using dynamic potential functions", in *IEEE Int. Conf. Robotics Automat.*, Rayleigh, NC, 1987, pp. 14–24.

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

[18] Christopher I. Connolly and Roderic Grupen, "The applications of harmonic functions to robotics", *Jorunal of Robotic Systems*, vol. 10, no. 7, pp. 931–946, 1993.

[19] Christopher I. Connolly and Roderic Grupen, "Harmonic control", in *The 1992 International Symposium on Intelligent Control*, August 1992.

[20] I. Mantegh, Jenkin M.R.M., and A.A. Goldenburg, "Fast trajectory planning and obstacle avoidance for autonomous robots using nonholonomic modelling", in *IEEE-SMC Computational Engineering in Systems Applications*, 1998.

[21] John S. Zelek, "Complete real-time path planning during sensor-based discovery", in *IEEE Conference on Intelligent Robots and Systems*, 1998, pp. 1399–1404.

[22] R.B. Tilove, "Local obstacle avoidance for mobile robots based on the method of artificial potentials", in *IEEE Int. Conf. Robotics Automat.*, Cincinnati, OH, May 1990, pp. 566–571.

[23] J. Borenstein and Y. Koren, "Real-time obstacle avoidance for fast mobile robots", *IEEE Trans. Syst. Man Cyber.*, vol. 19, no. 5, pp. 1179–1187, 1989.

[24] Y. Zhang and A. Mackworth, "Constraint programming in constraint nets", in *First Workshop in Principles and Practice of Constraint Programming*, 1993.

[25] D.K. Pai, "Programming anthropoid walking: Control and simulation", Tech. Rep. TR90-1178, Cornell University, 1990.

[26] D.K. Pai, "Least Constraint: A framework for the control of complex mechanical systems", in *Proceedings of the American Control Conference*, 1991, pp. 615–621.

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

[28] W. Wannasuphoprasit, R. Brent Gillespie, J. Edward Colgate, and Michael Peshkin, "Cobot control", in *IEEE International Conference on Robotics and Automation*, 1997, pp. 3571–3576.

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

[30] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*, Springer-Verlag, 1991.

[31] R.J. Spiteri, *Solution Methods for Differential Systems subject to Algebraic Inequality Constraints*, PhD thesis, University of British Columbia, Institute of Applied Mathematics and Dept. of Mathematics, September 1997.

[32] J.P. Aubin, *Viability Theory*, Birkhäuser, Berlin, 1991.

[33] J.P. Aubin and A. Cellina, *Differential Inclusions*, Springer Verlag, New York, 1984.

[34] F.H. Clarke, Yu. S. Ledyaev, R.J. Stern, and P.R. Wolenski, "Qualitative properties of trajectories of control systems: A survey", *Journal of Dynamical and Control Systems*, vol. 1, no. 1, pp. 1–48, 1995.

[35] A. Subbotin, *Generalized solutions of first-order PDEs*, Birkhauser, Boston, 1995.

[36] R. Isaacs, *Differential games; a mathematical theory with applications to warfare and pursuit, control and optimization*, J. Wiley and Sons, New York, 1965.

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

[38] R.M. Murray, Z. Li, and S.S. Sastry, *A Mathematical Introduction to Robotic Manipulation*, CRC Press, Boca Raton, Fla., 1994.

[39] John F. Canny and Ming C. Lin, "An opportunistic global path planner", *Algorithmica*, vol. 10, no. 2-4, pp. 102–120, 1993, Computational robotics: the geometric theory of manipulation, planning, and control.

[40] U.M. Ascher, S.J. Ruuth, and R.J. Spiteri, "Implicit-explicit Runge-Kutta methods for time-dependent partial differential equations", *Applied Numerical Math., Special Issue on Innovative Time Integrators*, 1997, to appear.

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

[42] R. Fletcher, "An ideal penalty function for constrained optimization", *J. Inst. Maths. Applns*, vol. 15, pp. 319–342, 1975.

[43] M.R. Hestenes, "Multiplier and gradient methods", *J. Opt. Theo. Applns*, vol. 4, pp. 303–320, 1969.

[44] J. Lewin, *Differential games : theory and methods for solving game problems with singular surfaces*, Springer-Verlag, New York, 1994.

[45] H.P. Huang and N.H. McClamroch, "Time-optimal control for a robotic contour following problem", *IEEE Journal of Robotics and Automation*, vol. 4, no. 2, pp. 140–149, 1988.

[46] A.W. Divelbiss and J. Wen, "Nonholonomic path planning with inequality constraints", *IEEE*, pp. 52–57, 1994.

[47] 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*, Tokyo, Japan, July, 1993, pp. 261–266.

[48] Michael K. Sahota, "Real-Time Intelligent Behaviour in Dynamic Environments: Soccer-playing Robots", Master's thesis, Department of Computer Science, University of British Columbia, Vancouver, Canada, 1991.

[49] V. Schulz, *Reduced SQP Methods for Large-Scale Optimal Control Problems in DAE with Application to Path Planning Problems for Satellite Mounted Robots*, PhD thesis, Interdisziplinäres Zentrum für Wissenschaftliches Rechnen, Univerisität Heidelberg, 1996.

[50] L. Ingber, "Simulated annealing: Practice versus theory", *Mathematical Computer Modelling*, vol. 18, no. 11, pp. 29–57, 1993.

[51] Christopher I. Connolly, J. B. Burns, and R. Weiss, "Path planning using Laplace's equation", in *Proceedings of the 1990 IEEE International Conference on Robautics and Automation*, May 1990, pp. 2102–2106.