Thomas M. Howard
Alonzo Kelly

Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213-3890, USA
{thoward,alonzo}@ri.cmu.edu
http://www.ri.cmu.edu/people/
{howard_thomas,kelly_alonzo}.html

Optimal Rough Terrain Trajectory Generation for Wheeled Mobile Robots

Abstract

An algorithm is presented for wheeled mobile robot trajectory generation that achieves a high degree of generality and efficiency. The generality derives from numerical linearization and inversion of forward models of propulsion, suspension, and motion for any type of vehicle. Efficiency is achieved by using fast numerical optimization techniques and effective initial guesses for the vehicle controls parameters. This approach can accommodate such effects as rough terrain, vehicle dynamics, models of wheel-terrain interaction, and other effects of interest. It can accommodate boundary and internal constraints while optimizing an objective function that might, for example, involve such criteria as obstacle avoidance, cost, risk, time, or energy consumption in any combination. The algorithm is efficient enough to use in real time due to its use of nonlinear programming techniques that involve searching the space of parameterized vehicle controls. Applications of the presented methods are demonstrated for planetary rovers.

KEY WORDS—mobile robots, trajectory generation, rough terrain, constrained optimization, optimal control, path planning

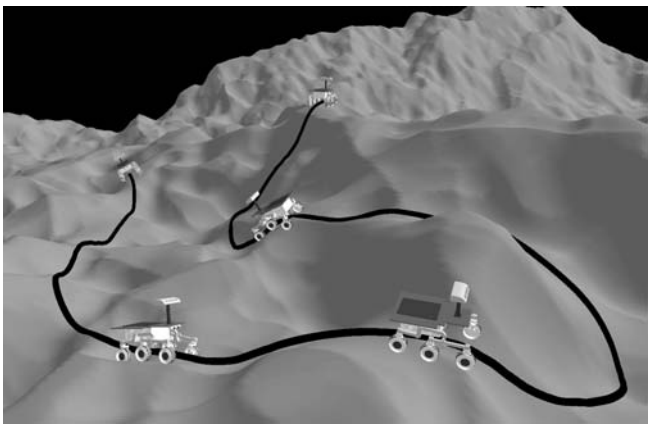

Fig. 1. Motion planning on rough terrain. For competent navigation in challenging environments, the terrain shape must be considered in the generation of continuous motion trajectories. The presented trajectory generation algorithm generates motion plans that account for arbitrary terrain shape, vehicle dynamics models, and wheel/terrain interaction models by linearizing and inverting forward models.

1. Introduction

In order to operate competently in any environment, a mobile robot must understand the effects of its own dynamics and of its interactions with the terrain. It is therefore natural to incorporate models of these effects in a trajectory generator that determines the controls necessary to achieve a prescribed motion.

Trajectory generation is the problem of determining a feasible motion (or a set of feasible motions) that will permit a vehicle to move from an initial state to a final state given some model of the associated dynamics. While this two-point boundary value problem is classical and well studied, it remains quite complex to solve adequately in practice. In order to generate a smooth, continuous path on flat terrain (which satisfies an arbitrary number of constraints involving position, heading, linear and angular velocities, and/or curvature), a nonlinear differential equation must be solved.

The addition of rough terrain to the problem further complicates matters by coupling these nonlinear equations of motion. Numerical methods, such as the ones presented in this article, are required to solve such problems for arbitrary terrain due either to its typically sampled representation, the nonlinearities of the models, or the nonexistence of closed-form integrals of the dynamics.

The International Journal of Robotics Research
Vol. 26, No. 2, February 2007, pp. 141-166
DOI: 10.1177/0278364906075328

1.1. Motivation

While the present generation of mobile robots is content to move from point A to B, and perhaps avoid obstacles along the way, truly useful machines must often interact with the world in ways more general than simply driving over it. For autonomous vehicles, trajectory generation algorithms can form the basis of any capacity to achieve a designated state of motion. Further, real-time algorithms are needed to do so in response to information gathered on the fly by perception.

Continuous motion is the core capacity of contemporary mobile robots. Yet, future machines will be required to address specific places in challenging terrains at specific attitudes and headings (Figure 1). At these places, they will be required to deploy implements to do something useful like measuring the composition of a rock, digging a hole, or placing a load of material on a truck.

Competent operations in cluttered environments require the capacity to understand precisely the entire space of feasible motions and to search it for a (or the best) solution. The trivial solution of line segments joining waypoints is often not feasible due to kinematic or dynamic limitations on curvature—even for vehicles that can nominally turn in place when stopped. Continuous curvature trajectories can have certain advantages over this trivial alternative. For example, the time to complete the mission or the exposure to risks (such as wheel slip) increases when the vehicle must stop and change direction.

In the context of semi-autonomous operations, trajectory generation can be used to drive the vehicle to an operator-designated waypoint or waypose. This point-and-click approach reduces both operator workload and telemetry bandwidth relative to continuous car-like driving. It potentially provides a better solution than might be achieved otherwise because closing the speed loops on the vehicle can mitigate the effects of latency.

For autonomous operations, trajectory generation can be used to acquire specific terminal states when the context is one of acquiring a fixed goal point. When following a path, trajectory generators can correct for path following errors by reacquiring a moving goal point at some forward position on the path.

Trajectory generation can also be cast as a core component of global motion planning. It can be used as a mechanism to encode the connectivity of state space in lattice-like networks as in Pivtoraiko and Kelly (2005). In this context, trajectory generation is the key to encoding a search space that intrinsically meets all mobility constraints.

1.2. Related Work

In the context of robot motion planning, most research in trajectory generation has dealt with finding obstacle-free paths subject to nonholonomic constraints assuming flat terrain and simple vehicle models. Two basic techniques exist. The first is sequential search of a graph whose edges consist of dynamically feasible low-order controls (arcs, clothoids, etc.). This technique produces a solution sequence of these low-order geometric primitives. The second technique is continuum optimization producing a single high-order parameterized geometric primitive. Graph-search methods generate the globally optimal solution in the discretized network, while parametric optimization methods search the continuum of solutions to find a locally optimal solution. The choice is between a sampled global solution and a continuous local one.

Some of the first work in trajectory generation involved composing optimal paths from a sequence of line segments, arcs (Dubins 1957), clothoids (Kanayama and Miyake 1985; Shin and Singh 1990), and cubic spirals (Kanayama and Hartman 1989). The desire for higher-order geometric primitives was intended to enable higher levels of continuity at the boundaries of the primitives. B-splines have been used to meet arbitrary position and heading boundary conditions by defining a sequence of knot points along the path (Komoriya and Tanie 1989). The concept of differential flatness, a property of a class of systems ideally suited to trajectory generation, was introduced by Fliess et al. (1995). Methods based on sinusoidal and Fourier series input functions also appear throughout the literature (Brockett 1981; Tilbury et al. 1992; Murray and Sastry 1998). These methods exploit the geometry of the problem to solve for the unknown path parameters directly. They cannot generally solve for collision-free paths in an obstacle field.

Graph-search techniques have been used for a long time in kinodynamic planning. In the context of robot manipulators, optimal joint trajectories were planned in Heinzinger et al. (1990) using grid-search. These methods also apply to the problem of solving for obstacle-free and minimum-length paths which satisfy nonholonomic and boundary constraints (Canny et al. 1988; Jacobs and Canny 1989; Barraquand and Latombe 1989; Reeds and Shepp 1990; Laumond et al. 1990). The drawback of using graph-search techniques for trajectory generation is the resolution lost due to discretization of state space and/or control space. The only boundary states that can be reached are those that already exist in the network.

Variational (optimization) techniques for trajectory generation, which search the continuum for a locally optimal solution, are as old as optimal control theory and have been used in most fields that employ automatic control (Betts 1998). This approach generally uses numerical methods to satisfy some set of boundary conditions and/or minimize some cost function by searching for the associated parameterized or sampled control. Automatic generation of joint trajectories using optimal control and cubic polynomial primitives were exhibited in (Lin et al. 1983). Minimum-time paths between boundary states are treated as a control problem in Baker (1989). Jackson and Crouch (1991) implemented the shooting method to solve for trajectories using cubic spline primitives.

Energy minimization was used in Delingette et al. (1991) to successively deform a curve until it met the boundary constraints, but it was found to be unsuitable for real-time applications. In Laumond (1995), a holonomic geometric path is found in an obstacle field and path segments are smoothed using optimal control. A near real-time optimal control trajectory generator is presented in Reuter (1998), which solves eleven first-order differential equations subject to the state constraints. A real-time trajectory generation algorithm for differentially flat systems is presented in Faiz et al. (2001), where an approximation of nonlinear constraints are replaced by linear inequality constraints. Kim and Tilbury (2001) used methods based on solving an approximate linearized problem (when systems are input-output linearizable) for UAV trajectory planning. Some of the most recent work in optimal control trajectory generation includes Kalmár-Nagy et al. (2004), where near-optimal paths are constructed for omnidirectional vehicles using bang-bang optimal control methods. Their methods generate minimum-time omnidirectional trajectories subject to complicated dynamics and actuator models. In Nagy and Kelly (2001) and Kelly and Nagy (2003), our group presented a real-time algorithm which solves the planar trajectory generation problem between arbitrary boundary states by linearizing and inverting the equations of motion.

All of the mobile robot trajectory generation methods discussed so far have assumed a flat world. By contrast, at the level of global motion planning where primitive trajectories are sequenced together, the nonflat terrain shape is often known. It is normally considered only in terms of its effect on the overall objective being optimized rather than in terms of its lower level effect on the motion itself.

Some of the first such rough terrain work involved using A* on a search-space based on the isolines of a relief map which incorporated energy costs associated with elevation changes (Gaw and Meystel 1986). More complicated terrain and vehicle models are introduced in Shiller and Chen (1990), Amar et al. (1993), and Bonnafous et al. (2001), which include kinematic and dynamic models. In Shiller and Chen (1990) and Shiller and Gwo (1991), optimal B-spline paths are generated on a B-patch representation of the terrain. Amar et al. (1993) adapt a sub-optimal cubic spline path assuming flat terrain to three-dimensional terrain using a kinematic vehicle model and enforcing terrain contact. A graph-search method using a set of arc primitives was used in Bonnafous et al. (2001) where distance and risk associated with the robot orientation was minimized.

One technique, which does account for the influence of terrain on motion, is the two-level planner is presented in Cherif et al. (1994) and Cherif (1999). This work searches for an optimal global path plan assuming flat terrain but it ensures connectivity between the states using a local trajectory generator that accounts for terrain shape. The local trajectory generator is set up as a graph-search problem and is solved using best-first search.

1.3. Discriminators

The method presented in this article differs from the body of prior work discussed above in several ways. The current state of the art in nonholonomic trajectory generation exhibits two classes. Algorithms in the first class produce smooth motion primitives on assumed flat terrain. The second class produces rough terrain primitives that are generated from a search, often of a graph, over a discretized control space rather than the continuum. Such techniques have not produced continuous motions at the junctions between motion primitives.

Continuum motion generation algorithms to date have not accounted for the effects of rough terrain and models of vehicle dynamics (e.g., delays, gain limits, wheel slip) at the level of primitive motions. Of course, the flat terrain assumption greatly simplifies the problem because it decouples the nonlinear state equations of the system (Howard and Kelly 2005a) but it does so at the expense of introducing model error for which controllers must later compensate. By accounting for terrain shape and models of vehicle dynamics, we eliminate this significant source of error predictively at planning time rather than reactively at execution time (in feedback control). Terrain shape is generally known (to the accuracy of the perception system) and it is already used elsewhere in the controls of most rough terrain autonomous vehicles, so terrain-adaptive lower level controls such as trajectory generation are an inevitable development that we describe in this article.

1.4. Technical Approach

A highly general description of the trajectory generation problem is that of generating a set of controls ($\mathbf{u}$) which satisfy a set of state constraints ($\mathbf{C}$) subject to a set of governing differential equations (f($\mathbf{x}$,$\mathbf{u}$,t)) describing the system dynamics:

$$\dot{\mathbf{x}} = \mathbf{f}\,(\mathbf{x}, \mathbf{u}, t)\,, \tag{1}$$

$$\mathbf{C}(\mathbf{x}, t) = \mathbf{0}. \tag{2}$$

Our formulation is one that transforms the basic optimal control problem into one of nonlinear programming. We assume some parameterized set of controls, and then compute the response to an initial guess for the control. Initial guesses are generated based on lookup tables for flat plane solutions. The mapping from control to response is then linearized numerically with respect to the parameters, and it is inverted numerically to provide the first-order updates to the guessed parameters that explain the deviation of the computed endpoint from the desired endpoint. Continued iteration refines the estimated control until the terminal state error descends below some threshold. The search spaces we consider have been observed to have a large radius of convergence and few local optimums, making such methods efficient and reliable in practice.

The numerical linearization in the above formulation is able to invert a forward model of a vehicle without explic-

itly encoding anything peculiar to the vehicle. However, the forward model does model the vehicle specifics. An arbitrarily complex vehicle model can be used to predict how the vehicle will move over the terrain. In our suspension model, we generally enforce terrain contact, possibly by articulating visco-elastic degrees of freedom, to determine attitude and elevation given position and heading. However, dynamic models admitting ballistic motions are also possible. In our propulsion model, such elements as actuator dynamics, rate limits, wheel slip, and the velocity kinematics are included. More complicated models involving forces, mass properties, and terramechanical models are consistent with our framework.

In summary, this article presents an approach to the problem that uses parameterized controls and nonlinear programming to search the continuum of control space efficiently for an optimum trajectory while preserving the ability to do so in real-time. The numerical approach used also makes the approach applicable to arbitrary terrain shape and arbitrary vehicle models actuated in arbitrary ways.

1.5. Layout

This paper is divided into seven sections. Section 2 describes the trajectory generator system architecture, including details about the boundary state definition, parameterized controls, numerical optimization, integration, and simulation steps in the process. Section 3 discusses initial guesses for the vehicle controls parameters and Sections 4 and 5 detail the experiments used to test the capabilities of the algorithm. Applications, conclusions, and future work are discussed in Sections 6 and 7.

2. Trajectory Generator System Architecture

The trajectory generation algorithm exhibits a three-level architecture which separates the trajectory generation (numerical optimization which minimizes constraint error), motion prediction (numerical integration to predict motion), and the vehicle simulation methods (Figure 2). The initial and final state boundary pair, the parameterization of the controls and the vehicle model comprise the inputs to the trajectory generator. The output is a trajectory, which we define as the union of the path (comprised of a vector of vehicle states) and the correct parameters for the controls. The vehicle model is defined externally to render the approach vehicle independent. Sections 2.1, 2.2, 2.3, 2.4, and 2.5 discuss the boundary state definition, parameterized vehicle controls, trajectory generation (numerical optimization), motion prediction (numerical integration), and the vehicle model respectively.

2.1. State Constraint Definition

One of the simpler forms of trajectory generation problems is a form of the two-point boundary value problem where it is convenient to constrain the vehicle state at the boundaries of the path. The integration of the equations of motion requires the initial state of the vehicle, and the terminal state defines the target state of the vehicle at some forward time t_f (Figure 2).

The most basic types of state constraints include world-frame position (x, y, z) and orientation (ϕ, θ, ψ). However, since roll (ϕ), pitch (θ), and elevation (z) are determined (under assumptions of terrain contact) by the pose and the interaction between the vehicle suspension and the terrain, only the position (x, y) or pose (x, y, ψ) boundary constraints on position and orientation can generally be specified. Typically there are also requirements on the vehicle controls that must be satisfied at the initial and terminal states. Linear and angular velocities $(\mathbf{v})$ and accelerations $(\mathbf{a})$ are often constrained for a dynamically feasible motion plan. A state vector is formed for the initial $(\mathbf{x}_0)$ and terminal states $(\mathbf{x}_f)$ of the robot:

$$\mathbf{x_0} = [x_0, y_0, \psi_0, \mathbf{v}_0, \omega_0, \ldots]^T, \tag{3}$$

$$\mathbf{x_f} = \left[x_f, y_f, \psi_f, \mathbf{v}_f, \omega_f, \ldots\right]^T. \tag{4}$$

The initial state constraint is typically satisfied trivially and its value is used to seed the numerical integration while the terminal state specifies the constraints at the end of the trajectory. We will write the constraints on the system in the following manner:

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \\ \dot{k} \\ \vdots \end{bmatrix} = \begin{bmatrix} f_1(\mathbf{x}, \mathbf{u}, t) \\ f_2(\mathbf{x}, \mathbf{u}, t) \\ f_3(\mathbf{x}, \mathbf{u}, t) \\ f_4(\mathbf{x}, \mathbf{u}, t) \\ \vdots \end{bmatrix}, \mathbf{C}(\mathbf{x}, t) = \begin{bmatrix} x_f - x\left(t_f\right) \\ y_f - y\left(t_f\right) \\ \psi_f - \psi\left(t_f\right) \\ k_f - k\left(t_f\right) \\ \vdots \end{bmatrix} = \mathbf{0}. \tag{5}$$

Extensions to enforce constraints at several different times, or constraints which are arbitrary functions of the states are straightforward.

2.2. Parameterized Vehicle Controls

In addition to the boundary states, the algorithm requires a set of parameterized controls that encode the motion of the vehicle (Figure 2). While the space of parameterized controls may represent only a subspace of all feasible motions, an appropriate choice of parameterization can represent nearly all possible controls. The set of all parameterized controls $(\mathbf{u})$ for the vehicle is defined as a function of the parameter vector $(\mathbf{p})$ and some distinguished independent variable (ζ):

$$\mathbf{u} = \mathbf{f}(\mathbf{p}, \zeta). \tag{6}$$

The free parameters in the parameter vector represent knobs that allow the algorithm to change the shape of the control. The number of free parameters in the parameter vectors minus the number of constraints represents the number of remaining degrees of freedom in the system.

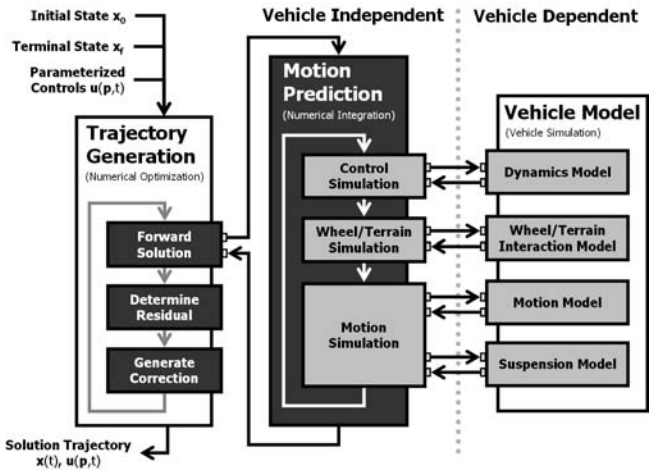

Fig. 2. Trajectory generator system architecture. The trajectory generation algorithm can be conceptualized as a three-level hierarchy. The highest level is a numerical optimization that minimizes the constraint error by adjusting the free parameters in the controls. The next level, motion prediction, is simply the numerical integration of the equations of motion. The lowest level, the vehicle model, simulates the vehicle's dynamics, motion, and suspension. This architecture is formalized and implemented to permit different vehicle models to interface to the same generation algorithm.

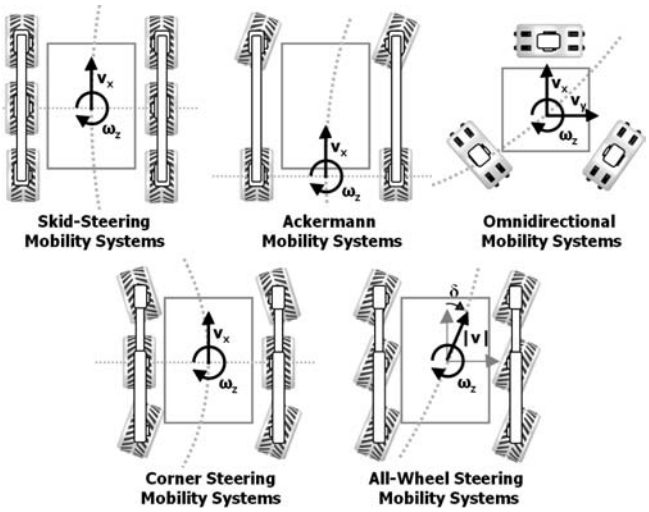

Fig. 3. Mobility system models. The choice of mobility system determines the vehicle's capacity to execute complex motions. Omnidirectional mobility systems are most capable, allowing instantaneous motion in any direction in the local terrain tangent plane. All-wheel steering mobility systems allow motion in any direction in the local tangent plane, but are restricted by nonholonomic constraints (Nesnas 2000). The skid-steering, Ackermann steering and corner steering mobility systems are the least flexible mobility systems because their linear velocity is constrained along the forward x-axis of the vehicle frame.

2.2.1. Body-frame Vehicle Parameterized Controls

Choosing the proper set and parameterization for vehicle controls (**u**) requires an examination of typical wheeled mobile robot mobility systems (Figure 3).

Nonholonomic constraints are often conveniently expressed in the body frame when they can be expressed by simply eliminating degrees of freedom. Neglecting articulations, wheeled mobile robots move rigidly with three degrees of freedom in the local terrain tangent plane. Hence, body-frame linear (v_x, v_y) and angular velocities (ω_z) are natural candidates for the controls because they reduce the dimension of the input space to a minimum. This choice can be made without loss of generality because a specific implementation might elect to include a mapping from wheel level controls onto body motions before the steps discussed below. Wheel velocities and steering angles are found by mapping the body-frame linear and angular velocities through the suspension kinematics.

Skid-steered, Ackermann, and corner-steered mobility systems are the simplest locomotion methods because their linear velocity is constrained to lie along the forward x-axis of the vehicle (v_x). Omnidirectional mobility systems have holonomic wheels which allow translation in any direction in the local tangent plane (v_x, v_y). All-wheel steering mobility systems allow translation in any direction in the local tangent plane so long as the path heading varies continuously.

The parameterization of the controls in terms of the linear velocity in the x and y-directions is not always convenient. When the number of control variables exceeds the number of degrees of freedom in the system, explicit constraints must be formulated. A convenient set of controls for all-wheel steering mobility platforms encodes the independent linear velocities in the local tangent plane in terms of the speed in the local tangent plane $(|v|)$ and the body frame referenced velocity direction (δ) (Howard and Kelly 2006).

$$\begin{aligned} v_x &= |v|\cos(\delta) \\ v_y &= |v|\sin(\delta) \end{aligned} \quad \leftrightarrow \quad \begin{aligned} |v| &= \sqrt{v_x^2 + v_y^2} \\ \delta &= \tan^{-1}\left(v_y / v_x\right). \end{aligned} \tag{7}$$

Notice that the skid-steered, Ackermann steered, and corner-steered mobility systems are simply a special case of the all-wheel steering mobility system definition where the continuous direction function (δ) is zero.

2.2.2. Control Function Parameterizations

Now that the control candidates for specific vehicle mobility systems have been defined, appropriate parameterized functions must be chosen to represent the freedoms in each vehicle control. The typical choices for the independent variable (ζ) in the parameterized controls are time (t) and arc-length (s).

This section details several logical options for the parameterization of the controls. In general, any parameterized function can be applied provided it permits numerical linearization of the solution to the equations of motion.

It is important to remember that these parameterized controls are the inputs to the system, not necessarily the response, so arbitrary parameterizations do not violate vehicle dynamics. Even discontinuous controls will generate a smooth response when passed through the dynamics.

Linear Velocity Profiles Typically, it is convenient for local motion planning algorithms to control the speed and shape of the trajectory independently. The initial speed, traversal speed, and the terminal speed are often known along with the desired linear velocity profile. An archetypal linear velocity profile for vehicles is the trapezoidal profile (Figure 4), defined by an initial velocity (v_0), initial acceleration (a_0), traverse velocity ($v_{traverse}$), terminal deceleration (a_f), terminal velocity (v_f), and the traversal time (Δt). Smoother (where acceleration and its derivative are defined), simpler (linear profile between v_0 and v_f), or more complex profiles (several intermediate velocities) can also be applied in the same framework.

$$\mathbf{p}_{|\mathbf{v}|} = \left[v_0 \; a_0 \; v_{traverse} \; a_f \; v_f \; \Delta t\right]^T . \tag{8}$$

Angular Velocity Profiles The shapes of trajectories are primarily determined by the angular velocity profile, so it is effective to represent the most possible feasible motions with a minimum number of parameters. One effective parameterization of an angular velocity control is a polynomial function of time (Figure 5):

$$\mathbf{p}_{\omega_\mathbf{z}} = [a \; b \; c \; d \; \ldots \; \Delta t]^T , \tag{9}$$

$$\omega_z\left(\mathbf{p}_{\omega_\mathbf{z}}, t\right) = a + bt + ct^2 + dt^3 + \ldots \; . \tag{10}$$

By the Taylor remainder theorem, the polynomial can represent all feasible motions to any desired degree of precision given enough terms. It is also differentiable, providing any desired degree of continuity.

When solving parameterized optimal control problems, the computations are best conditioned if all of the freedoms are of the same scale. Since interpolation schemes often use polynomial functions to join a set of control points, the control points themselves are an equivalent representation. Unlike polynomial coefficients, though, they have roughly equal scale (Figure 6):

$$\mathbf{p}_{\omega_\mathbf{z}} = [\omega_0 \; \omega_1 \; \omega_2 \; \omega_3 \; \ldots \; \omega_n \; \Delta t]^T , \text{ where}$$

$$\omega_0 = \omega\left(t_0\right) , \; \omega_1 = \omega\left(t_0 + \Delta t / n\right) , \cdots \omega_n = \omega\left(t_f\right) , \tag{11}$$

$$\omega_z\left(\mathbf{p}_{\omega_\mathbf{z}}, t\right) = \hat{a} + \hat{b}t + \hat{c}t^2 + \hat{d}t^3 + \ldots , \text{ where}$$

$$\hat{a} = f_1\left(\mathbf{p}_{\omega_\mathbf{z}}\right) , \hat{b} = f_2\left(\mathbf{p}_{\omega_\mathbf{z}}\right) , \hat{c} = f_3\left(\mathbf{p}_{\omega_\mathbf{z}}\right) , \ldots . \tag{12}$$

2.3. Trajectory Generation (Numerical Optimization)

Given the boundary states and the parameterized controls, the method used for finding the correct controls is nonlinear programming (Figure 2). Slightly different solution methods apply depending on the degree of problem constraint. For fully constrained problems, a constraint residual can be defined and when the algorithm is properly seeded, it will quickly converge to an acceptably small residual. Underdetermined problems encode a family of solutions from which one "best" solution must be chosen. The criterion of "best" is encoded by specifying some utility functional to be used to judge alternatives.

2.3.1. Constrained Trajectory Generation

The constrained trajectory generation formulation is the most straightforward approach. The process modifies the variables in the parameterized control ($\mathbf{p}$) until the terminal state of the forward simulated trajectory ($\mathbf{x}(t_f)$) is equal to the defined target terminal state ($\mathbf{x}_f$), thereby satisfying the constraint equations ($\mathbf{C}(\mathbf{x})$). Some of these boundary constraints are trivial to solve because they can be equated directly to certain parameters in the control (e.g., initial velocity). Nontrivial constraints (e.g., position) must be solved numerically using numerical methods.

The numerical method applied is Newton's method, where the Jacobian of the solution to the equations of motion is inverted to find a correction to the parameters in the system needed to minimize the constraint error ($\mathbf{\Delta} x_f(\mathbf{p})$), which we define for the moment to be the difference in the simulated terminal state and the terminal boundary constraints:

$$\left[\frac{\partial \Delta \mathbf{x_f}\left(\mathbf{p}\right)}{\partial \mathbf{p}}\right] \Delta \mathbf{p} = \left(\mathbf{x_f} - \mathbf{x}\left(t_f\right)\right) = -\Delta \mathbf{x_f}\left(\mathbf{p}\right) , \tag{13}$$

$$\Delta \mathbf{p} = -\left[\frac{\partial \Delta \mathbf{x_f}\left(\mathbf{p}\right)}{\partial \mathbf{p}}\right]^{-1} \Delta \mathbf{x_f}\left(\mathbf{p}\right) . \tag{14}$$

If the Jacobian of the constraints is non-square (when the constraint vector and the parameter vector are not of equal length), a pseudo-inverse can be applied to produce the least squares or least norm solution in order to generate a parameter correction vector.

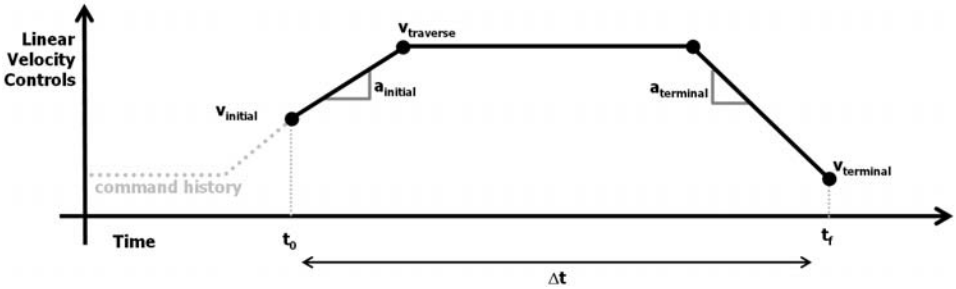

Fig. 4. Trapezoidal linear velocity profile. Typically most of the parameters defining the shape of the linear velocity control are known. For a trapezoidal velocity profile, the parameters in the control are the initial (v_0), intermediate ($v_{traverse}$), and terminal (v_f) velocities, initial (a_0) and terminal (a_f) accelerations, and the duration (Δt).

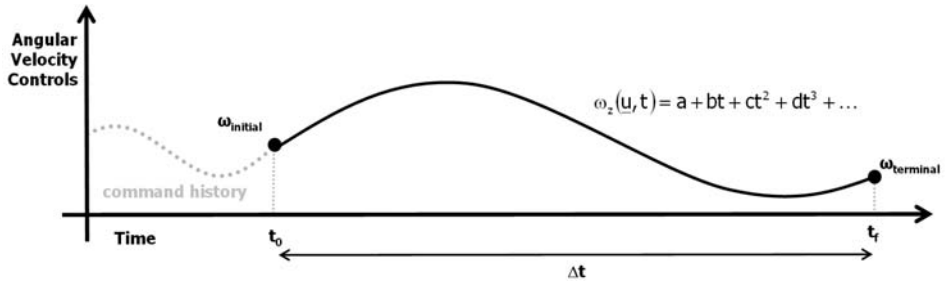

Fig. 5. Polynomial angular velocity profile. A polynomial function of time is an effective parameterization of the angular velocity controls. The parameters in the control are the polynomial coefficients ($a, b, c \ldots$) and the duration (Δt).

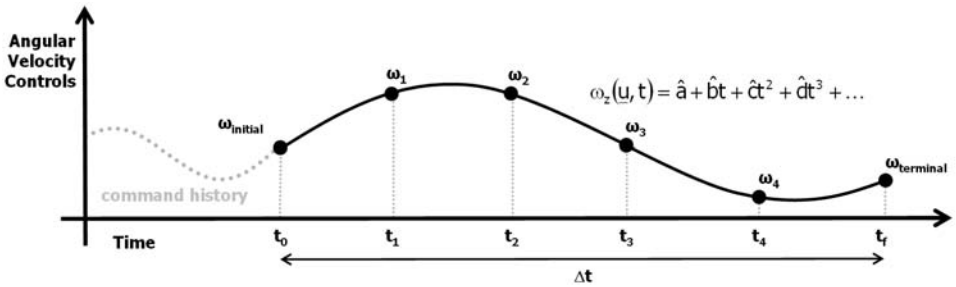

Fig. 6. Spline angular velocity profile. An alternate representation of a polynomial function is a spline function. The parameters in this control are the knot points ($\omega_1, \omega_2, \omega_3, \ldots$) that define the shape of the angular velocity control and the duration (Δt). This representation is beneficial for numerical optimization schemes because all of the knot points have roughly equal scale.

Since the partial derivatives of the integral of the equations of motion cannot generally be found analytically (Howard and Kelly 2005b) estimates must be found numerically. Forward (eq. (15)) or central difference (eq. (16)) linearizations of the forward equations of motion can be used to estimate these partial derivatives. The algorithm derives its vehicle independence from the numerical estimation of all of the partial derivatives.

$$\frac{\partial \Delta \mathbf{x}_{i,j}(\mathbf{p})}{\partial \mathbf{p}_k} = \frac{\Delta \mathbf{x}_{i,j}(\mathbf{p}_k + e, \mathbf{p}) - \Delta \mathbf{x}_{i,j}(\mathbf{p})}{e}, \quad (15)$$

$$\frac{\partial \Delta \mathbf{x}_{i,j}(\mathbf{p})}{\partial \mathbf{p}_k} = \frac{\mathbf{q}_{i,j}(\mathbf{p}_k + e, \mathbf{p}) - \mathbf{q}_{i,j}(\mathbf{p}_k - e, \mathbf{p})}{2e}. \quad (16)$$

2.3.2. Constrained Optimization Trajectory Generation

In addition to the pseudo-inverse, optimal control methods can be used to solve the underconstrained formulation of the problem. In this case, the optimization process creates the extra needed constraints to define a locally unique solution.

In the optimal control formulation of the problem, parameters in the linear and angular velocity controls ($\mathbf{p}$) must be adjusted to satisfy the constraints and minimize some utility functional ($\mathbf{J}(\mathbf{p})$). As in Kelly and Nagy (2003), this is accomplished using the method of Lagrange multipliers The Hamiltonian ($\mathbf{H}$) is defined as the sum of the cost function and the product of the Lagrange multiplier vector (λ) with the constraints:

$$\mathbf{H}(\mathbf{p}, \lambda) = J(\mathbf{p}) + \lambda^T \Delta \mathbf{x_f}(\mathbf{p}) = J(\mathbf{p}) + \lambda^T \left(\mathbf{x_f} - \mathbf{x}\left(t_f\right)\right). \quad (17)$$

The utility functional is a description of what we want to optimize over the path. In general, it takes the form:

$$J(\mathbf{p}) = \int_{t_0}^{t_f} Y(\mathbf{x}, \mathbf{p}, t)\, dt. \quad (18)$$

The utility functional is conceived as a line integral of a potentially time-varying utility function ($Y(\mathbf{x}, \mathbf{p}, t)$) along an unknown path. Equivalently, the problem can be formulated in terms of cost rather than utility. The time-varying utility function can be considered to be a (potentially time varying) field over the state vector. It represents any weighted combination of utilities or costs that are properties of a given state. It may include instantaneous energy consumption, wheel slip, loss of mobility, risk, slope, path smoothness, proximity to an obstacle, or anything else of interest.

$$Y(\mathbf{x}, \mathbf{p}, t) = w_1 Y_{risk}(\mathbf{x}) + w_2 Y_{energy}(\mathbf{x}, t) + w_3 Y_{smoothness}(\mathbf{x}) + \ldots. \quad (19)$$

The weights in the utility functional inevitably represent tradeoffs—like how far the system should be willing to go around an obstacle in order to reduce the risk, at a cost of lengthening the time to the goal. The tuning of these weights is unavoidable and both application and platform specific, and is outside the scope of this article.

The first-order necessary conditions for optimality are well known:

$$\frac{\partial \mathbf{H}(\mathbf{p}, \lambda)}{\partial \mathbf{p}} = \frac{\partial J(\mathbf{p})}{\partial \mathbf{p}} + \lambda^T \frac{\partial \Delta \mathbf{x_f}(\mathbf{p})}{\partial \mathbf{p}} = \mathbf{0}^T. \ (n \text{ eqns}). \quad (20)$$

$$\frac{\partial \mathbf{H}(\mathbf{p}, \lambda)}{\partial \lambda} = \frac{\partial \Delta \mathbf{x_f}(\mathbf{p})}{\partial \lambda} = \mathbf{0}. \ (m \text{ eqns}). \quad (21)$$

There are a total of $n + m$ equations for this system, where n is the length of the parameter vector (and the number of unknown parameters in the system) and m is the length of the Lagrange multiplier vector (and the number of constraints in the system).

This system is solved by linearizing the first-order necessary conditions. The initial guess of free control parameters and the Lagrange multipliers are adjusted at each iteration until a locally optimal solution is found. At this point, the gradient of the Hessian and the error in the boundary state both approach zero. Each iteration of the optimization involves a numerical solution of:

$$\begin{bmatrix} \frac{\partial^2 \mathbf{H}(\mathbf{p},\lambda)}{\partial \mathbf{p}^2} & \frac{\partial \Delta \mathbf{x_f}(\mathbf{p})}{\partial \mathbf{p}}^T \\ \frac{\partial \Delta \mathbf{x_f}(\mathbf{p})}{\partial \mathbf{p}} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \Delta \mathbf{p} \\ \Delta \lambda \end{bmatrix} = - \begin{bmatrix} \frac{\partial \mathbf{H}(\mathbf{p},\lambda)}{\partial \mathbf{p}}^T \\ \Delta \mathbf{x_f}(\mathbf{p}) \end{bmatrix}. \quad (22)$$

A correction factor for the control parameters and Lagrange multiplier vector is found by inverting the relationship in eq. (22):

$$\begin{bmatrix} \Delta \mathbf{p} \\ \Delta \lambda \end{bmatrix} = -\beta \begin{bmatrix} \frac{\partial^2 \mathbf{H}(\mathbf{p},\lambda)}{\partial \mathbf{p}^2} & \frac{\partial \Delta \mathbf{x_f}(\mathbf{p})}{\partial \mathbf{p}}^T \\ \frac{\partial \Delta \mathbf{x_f}(\mathbf{p})}{\partial \mathbf{p}} & \mathbf{0} \end{bmatrix}^{-1} \begin{bmatrix} \frac{\partial \mathbf{H}(\mathbf{p},\lambda)}{\partial \mathbf{p}}^T \\ \Delta \mathbf{x_f}(\mathbf{p}) \end{bmatrix} \quad (23)$$

A step size scaling factor (β) is used to enhance numerical stability when initial guesses are far from the local solution. In place of the scale factor, the optimal step size could be determined by performing a more expensive line search.

The Hessian of the Hamiltonian ($\mathbf{H}$), like the Jacobian of the forward solution, cannot be solved analytically in our case. Again, forward or central differences are used to estimate the partial derivatives:

$$\frac{\partial^2 \mathbf{H}_{i,j}(\mathbf{p})}{\partial \mathbf{p}_k \partial \mathbf{p}_l} = \frac{\mathbf{H}_{i,j}(\mathbf{p}_k + e, \mathbf{p}_l + e, \mathbf{p}) - \mathbf{H}_{i,j}(\mathbf{p}_l + e, \mathbf{p}) - \mathbf{H}_{i,j}(\mathbf{p}_k + e, \mathbf{p}) + \mathbf{H}_{i,j}(\mathbf{p})}{e^2}. \quad (24)$$

2.4. Motion Prediction (Numerical Integration)

The process of numerically integrating the differential equations that govern the motion of the vehicle is essentially one of simulation (Figure 2). Motion simulations are required to generate the trajectory corresponding to the parameterized

controls and consequently the Jacobian and the Hessian of the Hamiltonian (because they are found by numerical linearizations of forward solutions). The process is broken into three distinct steps: control dynamics, modeling of the wheel-terrain interaction, and motion simulation.

The first step, modeling of the control dynamics, is very important for accurately simulating real-world systems. Real systems have motor acceleration and torque limits, latency, and joint limits that must be modeled in order to produce an accurate simulation of how the vehicle will respond to its commands. The control dynamics modeling portion of the motion prediction stage simply determines the vehicle response to the body-frame linear and angular velocity controls:

$$\dot{\mathbf{x}}\left(\mathbf{u}, t\right) = \mathbf{f}_{\text{control dynamics}}\left(\mathbf{x}, \mathbf{u}, t\right). \tag{25}$$

The control dynamics portion of the model determines the forces that the vehicle will exert on the environment, and therefore the resulting response forces from the environment. It is here where models of wheel slip, sliding, and the actions of external forces on the system are applied.

The last portion of the motion prediction is the actual motion simulation, which determines the change in vehicle pose $(\mathbf{r}|_{world}, \mathbf{o}|_{world})$ over a small time step (Δt):

$$\mathbf{x}\left(t + \Delta t\right) = \mathbf{x}\left(t\right) + \dot{\mathbf{x}}\left(\mathbf{x}, \mathbf{u}, t\right)\Delta t. \tag{26}$$

Since ground vehicles generally cannot control their roll, pitch, or elevation rates (they are functions of the interaction between the environment and the suspension), new estimates of these states are computed by the suspension model at the end of each motion prediction step.

2.5. Vehicle Model (Vehicle Simulation)

From a coding perspective, the vehicle model portion of the trajectory generation algorithm is an instance of the abstract vehicle model class that determines how the robot will respond to the commanded body-frame linear and angular velocity controls, change its position and orientation, and interact with its environment (Figure 2). The vehicle model is generally broken down into four elements: a dynamics model, a wheel/terrain interaction model, a motion model, and a suspension model.

2.5.1. Vehicle Dynamics Models

Vehicle dynamics models simulate the response of the body-frame vehicle commands to the torques on and/or velocities of the wheels. By modeling the response to the parameterized controls, the solutions found are dynamically feasible.

A few elements of vehicle dynamics models are rate limits, joint limits, and latency. Rate limits represent constraints on how fast actuators can turn. Modeling these effects can account for drive wheels that move slower than requested or steering servos that lag behind their desired orientation. By contrast, joint limits bound entire regions of control space that contain infeasible motions. An example of such infeasible motions is the impossible "turn-in-place" maneuver in automobiles. The front wheels can never reach the steer angles required for this motion. Accounting for latency in the system is essential for generating correct trajectories when the vehicle state can change dramatically over the scale of the latency.

These types of constraints can be handled in the trajectory generator by using the response to the input controls in the integration of the kinetic motion model instead of the controls themselves. Hard limits can be imposed on steering angles and models of actuator dynamics can simulate delays in the motor controller.

The imposition of such joint and rate limits in the vehicle model requires a method for computing wheel direction and velocities given some body-frame linear and angular velocity. To determine the mapping from body-frame linear and angular velocity to wheel velocities, suspension articulation rates can be temporarily neglected to simplify the computations. For such an assumed rigid body, the velocity of any point on the vehicle can be determined from the linear and angular velocity of the body frame. Specifically at the location of a wheel:

$$\mathbf{v}|_{\text{wheel}} = \mathbf{v}|_{\text{body}} + \omega|_{\text{body}} \times \mathbf{r}|_{\text{wheel}}. \tag{27}$$

In the linear and angular velocity vectors, only the controllable velocities (v_x, v_y, ω_z) need to be considered. The vector $\mathbf{r}|_{wheel}$ is the displacement from the body frame to wheel contact point. The wheel speed and direction can then be determined from the computed wheel velocity vector.

2.5.2. Wheel/Terrain Interaction Models

By predicting the wheel/terrain interaction in the planning stage of the overall autonomy system, rather than accounting for it in the execution stage, more dynamically feasible vehicle motion can be generated. Traversing high wheel slip environments is a major challenge for current planetary robotic systems, for example and the present algorithm promises to help compensate for such slip to the degree that it can be predicted by the specialized perception algorithms that are appearing (Angelova et al. 2006).

2.5.3. Vehicle Motion Models

The vehicle motion model maps body frame linear and angular velocities to world frame position and orientation rates. A body-frame coordinate system is defined with the positive x-axis pointing forward, the positive y-axis pointing to the right, and the positive z-axis pointing down. The mapping of linear velocities from the body-frame $(\mathbf{v}|_{body})$ to world frame $(\mathbf{v}|_{world})$ is completed by rotating the body-frame x, y, and z axes by the Euler angles roll (ϕ), pitch (θ), and yaw (ψ) respectively:

$$\dot{\mathbf{r}}|_{\mathbf{world}} = \mathbf{v}|_{\mathbf{world}} = \mathbf{ROT_z}(\psi)\,\mathbf{ROT_y}(\theta)\,\mathbf{ROT_x}(\phi)\;\mathbf{v}|_{\mathbf{body}}\,, \quad (28)$$

$$\dot{\mathbf{r}}|_{\mathbf{world}} = \begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{z}\end{bmatrix} = \begin{bmatrix} c\psi c\theta & c\psi s\theta s\phi - s\psi c\phi & c\psi s\theta c\phi + s\psi s\phi \\ s\psi c\theta & s\psi s\theta s\phi + c\psi c\phi & s\psi s\theta c\phi - c\psi s\phi \\ -s\theta & c\theta s\phi & c\theta c\phi \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}. \quad (29)$$

This relationship can be inverted to determine the body-frame velocity vectors in terms of the global position rates:

$$\mathbf{v}|_{\mathbf{body}} = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = \begin{bmatrix} c\psi c\theta & s\psi c\theta & -s\theta \\ c\psi s\theta s\phi - s\psi c\phi & s\psi s\theta s\phi + c\psi c\phi & c\theta s\phi \\ c\psi s\theta c\phi + s\psi s\phi & s\psi s\theta c\phi - c\psi s\phi & c\theta c\phi \end{bmatrix} \begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{z}\end{bmatrix}. \quad (30)$$

The mapping between the Euler angle rates and the body-frame angular velocities ($\omega|_{body}$) can also be found by transforming the individual Euler rotation rates from their intermediate frames to the robot-fixed frame:

$$\omega|_{\mathbf{body}} = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + rot_x(\phi) \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + rot_x(\phi)\, rot_y(\theta) \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix}, \quad (31)$$

$$\omega|_{\mathbf{body}} = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} 1 & 0 & s\theta \\ 0 & c\phi & -s\phi\, c\theta \\ 0 & s\phi & c\phi\, c\theta \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}. \quad (32)$$

Just as with global position rates and body-frame linear velocities, this relationship can be inverted to determine the Euler rates in terms of the body-frame angular velocities:

$$\Psi|_{\mathbf{world}} = \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & t\theta\, s\phi & -t\theta\, c\phi \\ 0 & c\phi & s\phi \\ 0 & -\frac{s\phi}{c\theta} & \frac{c\phi}{c\theta} \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}. \quad (33)$$

Equations (29) and (33) form the basis of the time-based velocity kinematics. The rate of change of global position and Euler angles can be determined given a set of controllable body-frame linear and angular velocities:

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} c\psi c\theta & c\psi s\theta s\phi - s\psi c\phi & 0 \\ s\psi c\theta & s\psi s\theta s\phi + c\psi c\phi & 0 \\ 0 & 0 & \frac{c\phi}{c\theta} \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega_z \end{bmatrix}. \quad (34)$$

2.5.4. Vehicle Suspension Models

Vehicle suspension modeling is the problem of properly determining the attitude and elevation of a vehicle given other elements of its current state. This is an embedded optimization problem and it is often highly underdetermined. We solve this problem by conceptually allowing the vehicle to float in altitude and attitude, and articulate in suspension, while minimizing the residual between the wheels and the terrain elevations under (or above) them.

3. Initial Guesses for the Vehicle Controls Parameters

Good initial guesses for the vehicle controls parameters are very important to the efficiency of the trajectory generation algorithm. When initial guesses of parameters are close to the actual solution, fewer iterations of the algorithm are required to satisfy the boundary state constraints. Furthermore, when local minima exist, a good initial guess is insurance against falling into the wrong minimum. Since initial guesses for three-dimensional terrain, arbitrary vehicle dynamics, and arbitrary terrain models are difficult to calculate (because of their dimensionality), flat surface solutions with limited dynamics are considered. Historically, approximations of the solutions have been found by hand-tuning polynomial functions to data from a few dimensions (Nagy and Kelly 2001). This section discusses the problem of generating the dataset for the initial guesses and two methods of storing them: a initial guess lookup table and a neural network.

3.1. Lookup Table

A lookup table is an efficient means of storing initial guesses for the vehicle controls parameters given that the space of solutions is low dimensional and is smooth. This is yet another reason for using parameterized controls because they encode the entire shape of the trajectories in relatively little memory.

Generally the most important boundary constraints which influence the shape of the path are the initial and terminal positions (x, y), headings (ψ), and initial and final curvatures (k_0, k_f), resulting in a five-dimensional lookup table of solution parameter vectors. More dimensions, such as velocities, can be incorporated into the lookup table if they have dramatic effects on the accuracy of the initial guess.

In order to generate the initial guess lookup table efficiently, we use previous solutions to seed neighboring trajectory generation problems in the table. This requires a satisfactory initial guess for the first trajectory generation problem, which is ideally centrally located in the lookup table.

In practice it may be necessary to adjust the order of the progression through each dimension of the initial guess lookup table if convergence problems (due to dynamic infeasibility or sparsity of the discretization) are observed. The resolution and dimensionality of the stored lookup table is a function of the robot's storage and computing power and is therefore platform dependent.

3.2. Neural Network

Another method to generate the initial guess used in the trajectory generator is to use machine-learning algorithms to fit a function to a large training set of trajectory generation examples. The initial guess lookup table from Section 3.1 (or a sampled version of it) can be used as a good training set because the boundary state pairs are regularly separated.

The motivating factor behind using such a representation is the space savings. A high-dimensional initial guess lookup table can require tens or hundreds of megabytes of storage whereas a neural network, which requires that only

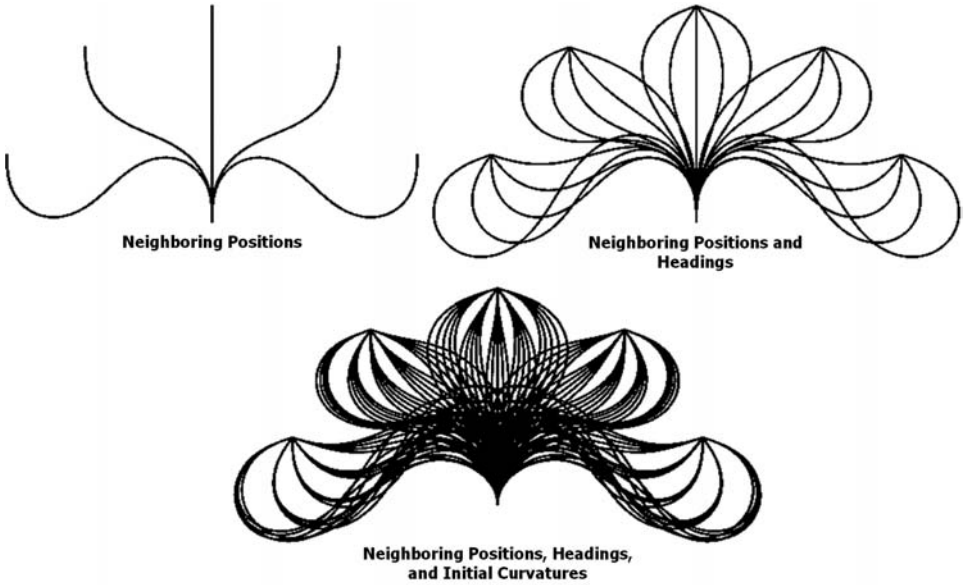

Fig. 7. Neighboring solutions for initial guess lookup table generation. The initial guess lookup table is generated by continually solving neighboring trajectory generation problems in such a way that the last solution is a good initial guess for the next query.

the weights of the learned function be stored, requires only a few kilobytes. This is important for applications with limited storage such as planetary rovers. The solutions are good candidates for machine learning algorithms because the parameters of the solutions are generally smooth and continuous. The downside to applying neural networks for such situations is the upfront cost of learning the entire function.

4. Experimental Setup

Using the methods and algorithms previously described, we have built a continuous primitive trajectory generator for arbitrary vehicles that accounts for terrain geometry in the motion prediction. The experiments and simulations will use a vehicle model based on the Rocky 8 prototype Mars rover (Figure 8). The vehicle has an all-wheel steering mobility system that is artificially constrained in some of the subsequent examples in order to demonstrate trajectory generation for less capable mobility systems. This section will outline the state constraints (Section 4.1), control parameterizations (Section 4.2), trajectory generation algorithm implementation (Section 4.3), suspension models (Section 4.4), vehicle dynamic models (Section 4.5), and the simulated environment used in the experiments.

4.1. State Constraints

For the series of examples in Section 5, we will solve the two-point boundary value problems typically associated with trajectory generation. It is important to note however that in general the algorithm does not require that the state constraints be specified at the boundaries. Any number of constraints can be specified at any number of points in time. The implemented error threshold used to declare convergence of the algorithm is shown in Table 1.

Only the terminal positions, headings, directions, and curvatures must be satisfied by the trajectory generator optimization. The position, heading, direction, and curvature initial state constraints are solved trivially using the current state of the vehicle.

4.2. Control Parameterizations

This implementation employs a fifth-order polynomial spline function in curvature and linear profiles for the linear velocity and direction. A curvature profile is used in the same way that the angular velocity is used and the two are interchangeable. There are artificial constraints on the initial controls—that they be equal to the initial curvature, linear velocity, and direction state constraints for maximum continuity.

Table 1. Convergence Criterion

State Constraint	Required Accuracy
Position (x, y)	0.001 m
Heading (ψ)	0.001 rad
Direction (δ)	0.001 rad
Curvature (ω)	0.001 rad

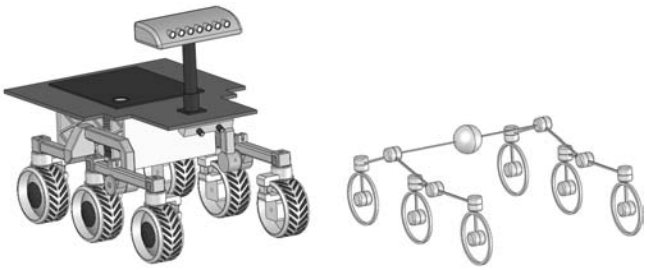

Fig. 8. Kinematic models for Rocky 8. A kinematic model can easily be generated for mobile robots with rocker-bogie suspensions using a series of revolute joints. The kinematic model is used to map body frame velocities to wheel frame velocities and to determine the attitude and elevation of the robot given its position.

4.3. Trajectory Generation Algorithm Implementation

Since the length of the parameter vector (8) exceeds that of the constraint vector (6), the system can be solved using the constrained or constrained optimization trajectory generation techniques. The constrained trajectory generation technique (Section 2.3.1) is used in Sections 5.1–5.5 whereas the constrained optimization technique (Section 2.3.2) is used in Section 5.6.

4.4. Suspension /Kinematic Model

In order to map body-frame velocities to wheel velocities and to determine the orientation and configuration of the vehicle on arbitrary terrain, the suspension of the vehicle must be modeled. The body to wheel kinematics equations were computed in a manner similar to Tarokh and McDermott (2005). The vehicle can be modeled as a series of revolute joints and links as seen in Figure 8.

The suspension model has been implemented as a numerical optimization (a different application of Newton's Method) that minimizes the distance between wheel contact points and the terrain by adjusting the three rocker-bogie freedoms, and the roll, pitch, and elevation. In general, the partial derivatives required by the optimization must be found numerically. However, an estimate of the Jacobian can be found analytically by taking the partial derivatives of the forward kinematics equations of the wheel contact point with respect to the body-frame elevation, attitude, and suspension freedoms. This solution is efficiently computed online in the forward wheels in contact with the terrain is computed efficiently during the forward dynamics solution by using the previous state (suspension, attitude, altitude) as the initial estimate for the suspension optimization of the present step.

4.5. Vehicle Dynamic Models

Varying dynamic models, including models of wheel slip and sliding are demonstrated in Section 5.5. The examples in Sections 5.1–5.4 and 5.6 apply ideal vehicle dynamics models in order to isolate effects of different aspects of modeling. In general, all of these applications can use arbitrarily complex vehicle models (joint limits, motor models, etc.) at the burden of higher computational costs.

4.6. Simulated Environment

The terrain in the experiments is represented as an elevation map generated using fractals. A third-order Lagrangian interpolation scheme was implemented to determine the elevation at any given position in the environment. This method was preferred over simpler and less costly linear interpolation schemes because third-order Lagrangian interpolation provides continuous derivatives at the boundaries between map cells.

5. Experiments and Results

This section demonstrates some uses of the developed trajectory generator. A comparison of paths generated using the rough terrain and traditional trajectory generations is shown first to demonstrate the need to include the terrain geometry in the forward model. Then, the observed rate and behavior of convergence of the algorithm is discussed followed by examples of motion generation for different mobility systems, cases where vehicle dynamics are important, and the generation of optimal trajectories.

Results are presented for a simulated vehicle because this is the best way to test the algorithm on a statistically significant set of cases. The main purpose of the algorithm is to invert a model to produce feasible motions that meet the dynamic constraints encoded in the model and the boundary constraints encoded in the problem specification. The fidelity of the model used is an important but separate and independent question that can only be evaluated on a real vehicle. We assert that, however the parameters of the vehicle model may change in order to calibrate it better to reality; our trajectory generator will still be able to invert it. The algorithm has been integrated into the CLARAty system at the Jet Propulsion Laboratory and successfully field tested on the Rocky 8 prototype mobile robot platform. It has also been used on custom designed unmanned ground vehicles (prototype military robots) and automated automobiles for use in robot races.

Due to space limitations, a relative few examples are presented in detail.

5.1. Rough Terrain Trajectory Generation

Accounting for terrain geometry in trajectory generation is important in rough terrain environments. Figure 9 shows two forward simulations of trajectories generated by alternately assuming and then not assuming flat terrain. The rough terrain solution meets the stringent terminal constraints (in three iterations), whereas in this example the flat terrain solution is off by 24.1% in relative position and 11.5% in relative heading.

There are three important points to take from this figure:

- Neglecting the influence of the terrain geometry in the motion model leads to incorrect trajectories.
- Any path generated assuming flat terrain will be shorter than the real three-dimensional path between two arbitrary states on general surfaces. Trajectories generated by ignoring terrain geometry will make turns too early because the vehicle will not recognize that it has only displaced a fraction of its apparent (x, y) position change.
- The flat terrain solution is close enough to the real solution to provide an initial guess to initialize the rough terrain trajectory generation algorithm. Even in the presence of large terrain undulations like the ones shown in Figure 10, convergence typically requires less than three iterations.

Of course, in modern robotic systems, such errors are typically treated using feedback control and path tracking. However, the main point here is that such errors can be avoided entirely, before the fact, using the methods of the article because perception used in local planning can inform the algorithm on terrain geometry.

5.2. Algorithm Convergence

To demonstrate robustness to poor initial guesses for the vehicle control parameters, the same problem as Section 5.1 was solved using an artificially poor initial guess for the parameters of the control and a smaller step size scaling factor ($\alpha = 0.625$). Figures 10 and 11 show trajectories representing each of the nine iteration steps required by the trajectory generator to converge to a solution that reaches the target terminal state.

Figure 11 demonstrates the convergence property of Newton's Method as the parameter correction is proportional to the current state error. Even when provided with a relatively large error in the initial guess, the algorithm is able to converge to the correct solution because of the relative convexity of the solution space.

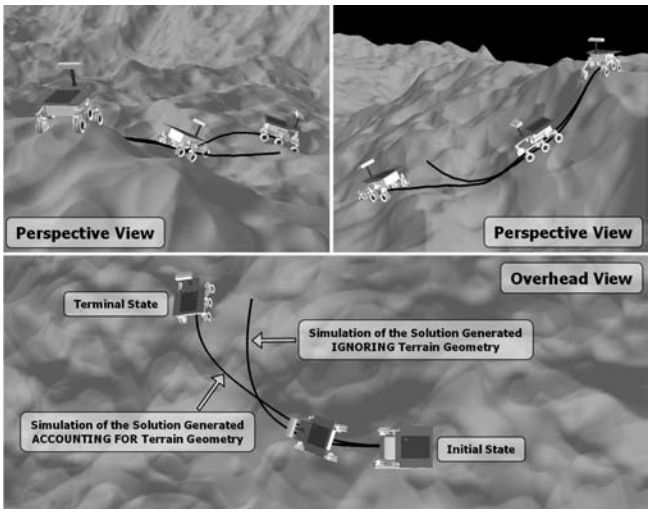

Fig. 9. Trajectory generation compensating for terrain geometry. An example motion plan between a pair of boundary states is shown with and without compensating for terrain geometry. The trajectory plan ignoring the terrain geometry does not reach the target terminal state.

5.3. Rough Terrain Trajectory Generation for Instrument Placement with Restricted Mobility

Instrument placement problems for mobile robots typically require a local planning algorithm to generate a trajectory to a terminal state where its position and heading are defined. The position and heading boundary constraints are typically set such that the target is within the range of motion of scientific instruments or the field of view of sensors or cameras.

To demonstrate a more general example of the presented trajectory generation algorithm, we have used the algorithm to plan trajectories to visit seven sequential scientific targets as seen in Figure 12. Here the simulated mobile robot's all-wheel steering capability is constrained to demonstrate motion planning for skid-steered, Ackermann, and corner-steering mobility systems (linear velocity is restricted to be along the x-axis of the vehicle). The generated motion plan consists of seven forward and five reverse trajectories all subject to the vehicle's nonholonomic constraints. The trajectories used trapezoidal velocity profiles with zero initial and terminal velocity, ± 2.0 m/s^2 acceleration and deceleration, and a traverse speed of ± 1.0 m/s. Figure 12 shows views of the simulated instrument placement scenario and Figure 13 plots the position and heading errors as functions of the number of iterations of the algorithm.

In this situation, the termination conditions are satisfied by all twelve trajectories in fewer than four iterations of the algorithm when considering the terrain geometry. As in the previous examples, this demonstrates fast convergence of the algorithm in relatively difficult terrain geometry while illustrating a typical application for the presented method.

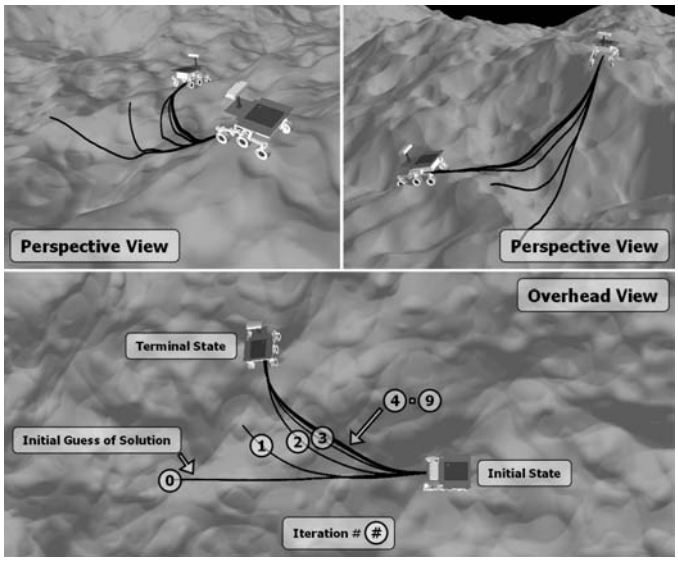

Fig. 10. Trajectory generation convergence. The same problem in Figure 8 is solved using an artificially poor initial guess to demonstrate robustness of the algorithm. As the method progresses, successive iterations of the optimization minimize the distance between the simulated state and the target terminal state until the error reaches an acceptable level.

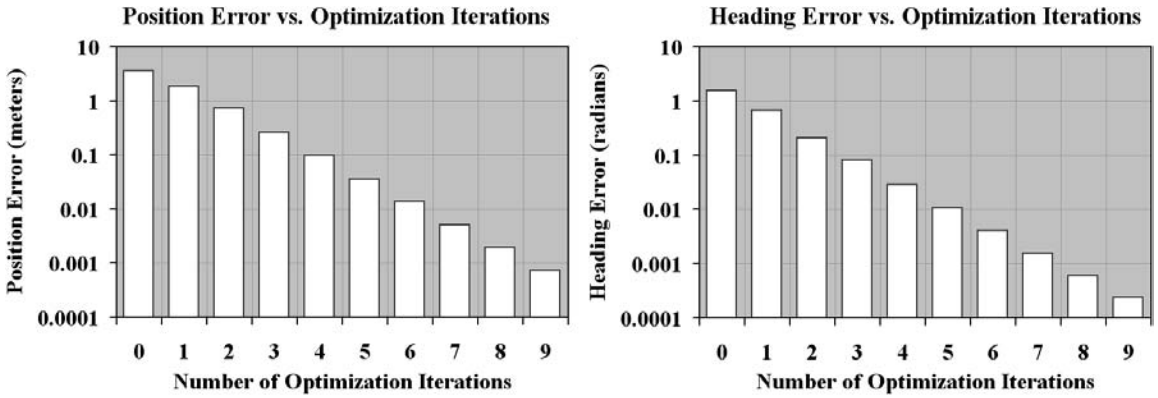

Fig. 11. Trajectory generation convergence. The position and heading errors are shown for each trajectory in Figure 10. Given an artificially poor initial guess of the vehicle control parameters, the trajectory generator determined the correct solution in nine iterations of the optimization.

5.4. Rough Terrain Trajectory Generation for Instrument Placement and All-wheel Steering Mobility Systems

The ability to generate paths in arbitrary terrain for all-wheel steering mobility systems can endow the robot with the capacity to move efficiently through the environment. This is especially important for planetary robotics applications where energy is limited, the environment may be cluttered with obstacles, and the orientation of the robot is important (to deploy scientific instruments). The same instrument placement scenario as in Section 5.3 is presented in Figure 14 except that now the robot can use the all-wheel steering mobility system. The motion plan illustrates how the algorithm can exploit all-wheel steering to generate smoother, more efficient motion plans for multiple sequenced instrument placement problems. Figure 15 plots the position and heading errors as functions of the number of iterations of the algorithm.

In this situation, the termination conditions are satisfied by all seven trajectories in fewer than six iterations when considering the terrain geometry and the all-wheel steering ca-

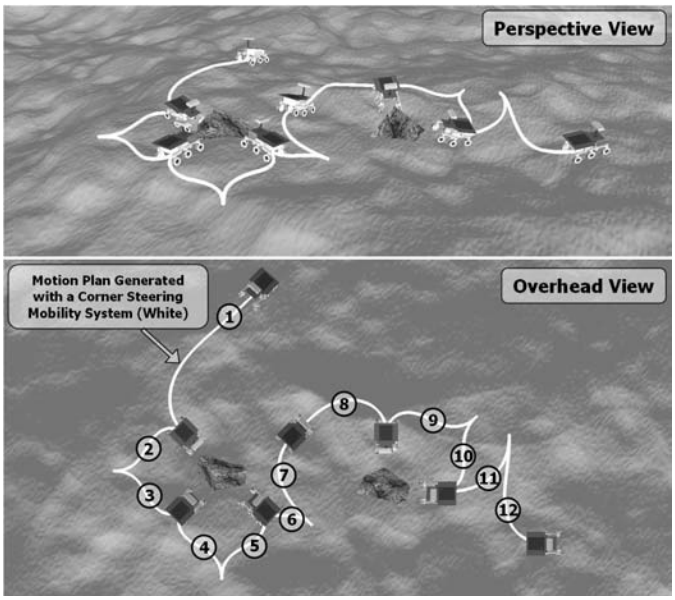

Fig. 12. Instrument placement using corner steering mobility systems. Sequences of trajectories are planned for an example instrument placement task using a corner steering mobility system. The targets must be aligned with the front of the robot chassis because of the location of the scientific instruments. Each of the seven science targets are achieved by planning twelve trajectories that include forward and backward motions.

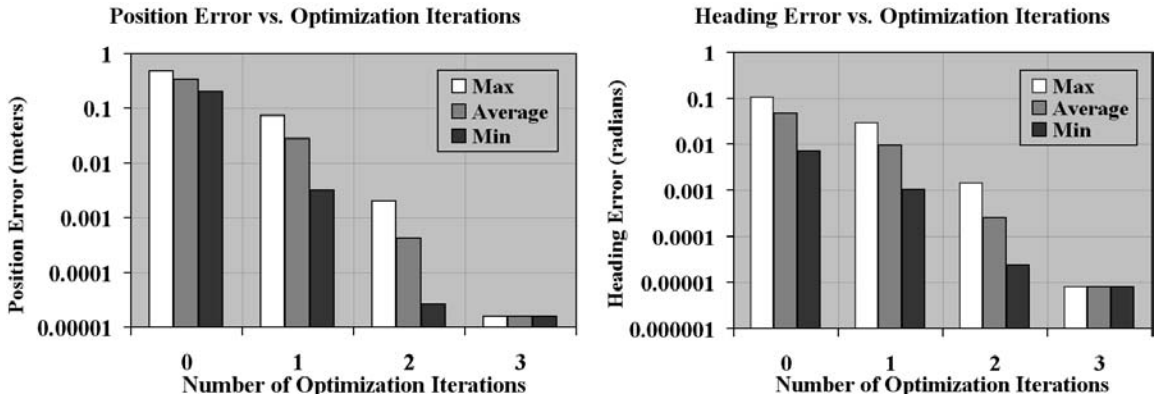

Fig. 13. Trajectory generator convergence for instrument placement tasks using corner steering mobility systems. For the sequence of trajectories shown in Figure 12, the maximum, average, and minimum position and heading errors of the twelve planned trajectories are shown as a function of the number of iterations executed by the trajectory generation algorithm.

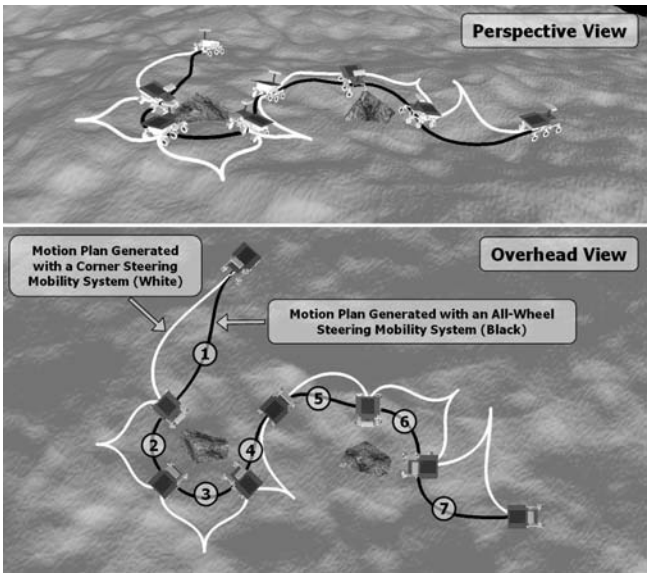

Fig. 14. Instrument placement using all-wheel steering mobility systems. The same instrument placement problem in Figure 12 is solved using an all-wheel steering mobility system, allowing the vehicle path heading to change independently from the body yaw. Each of the seven science targets are achieved by planning only seven motions. The overall motion plan (black path) is more efficient than the corner steering motion plan (white path) by effectively exploiting the all-wheel steering mobility.

pability of the vehicle. The constraint error is larger because the initial guess used does not account for the all-wheel steering capability of the vehicle. In practice, such solutions could be encoded into the initial guesses for the vehicle controls parameters.

In comparing the motion plans for the instrument placement scenarios presented in Sections 5.3 and 5.4, the all-wheel steering motion plan is clearly shorter and more efficient. The planned all-wheel steering mobility system path is 42.3% shorter (24.33 s vs. 42.15 s) than the corner steering mobility system path. Of particular interest are the all-wheel steering trajectory sequences (2–3) and (6) in Figure 14, which allow the mobile robot to circle a target while continuously pointing its instruments towards the target. Trajectory generation algorithms that can exploit all-wheel steering systems can enable far more capable and efficient local motion planning algorithms.

5.5. Trajectory Generation Considering Vehicle Dynamics

One advantage of the numerical linearization is that arbitrary models of vehicle dynamics can be incorporated. Often, dynamic effects are neglected at the local motion planning level and dealt with at a lower level through feedback control. Our formulation allows dynamic models of wheel–terrain interaction and other vehicle dynamics to be treated beforehand in the trajectory generation solution.

5.5.1. Trajectory Generation Considering Wheel Slip Models

Compensating for wheel slip is presently one of the most important mobility problems for planetary mobile robots (Biesiadeki et al. 2005). Algorithm convergence is not guaranteed for any wheel slip model since a reasonable initial guess of the solution is required. However, if a reasonable initial guess is provided and convergence is still not achieved, then there is likely no dynamically feasible motion.

Since the trajectory generator uses the response of body-frame velocity controls, it is necessary to first determine individual wheel velocities, apply a slip model on each wheel, and then invert the body kinematics to estimate of the body-frame velocity response. While this approach is not as principled as force-based models of wheel slip, this approximation is often used in practice, so we will use it here.

A representative problem involving wheel slip is planning trajectories for planetary rovers while climbing hills, as seen in Figure 16. Typical wheel slip models for such applications, respond with an attitude dependent percent of the commanded velocity. When wheel slip is not modeled in the trajectory generator, paths come up short. We can, however, use the presented trajectory generation algorithm to compute paths that meet the target state constraints subject to these wheel slip models.

The wheel slip model used here calculates percentage wheel slip for each wheel based on the vehicle attitude. The model parameters used were consistent with observations from field experiments. In general, any wheel slip model can be implemented in the vehicle model for such situations.

Figure 17 shows the commanded and response linear and angular velocities of the two paths shown in Figure 16. Notice that the path generated by incorporating the wheel slip model takes longer to execute (9.21 s vs. 6.16 s) because the net velocity response of the wheel slip model is always lower than the commanded velocity. Also, note that the response velocities are not particularly smooth functions despite the smoothness of the commands.

5.5.2. Trajectory Generation Considering Vehicle Dynamics Models

Wheel slip is not the only form of dynamics that can be accommodated. We will use the term "sliding" to refer to unintended sideways motion of the wheels. Sliding or slipping dynamics can be incorporated in the control dynamics model in order to generate trajectories that compensate for these motions automatically. For example, the planetary robot in Figure 18 is attempting to follow a provided trajectory while crossing a slope. If the plan is generated and followed without modeling the sliding effects of the slope, the robot will slide down

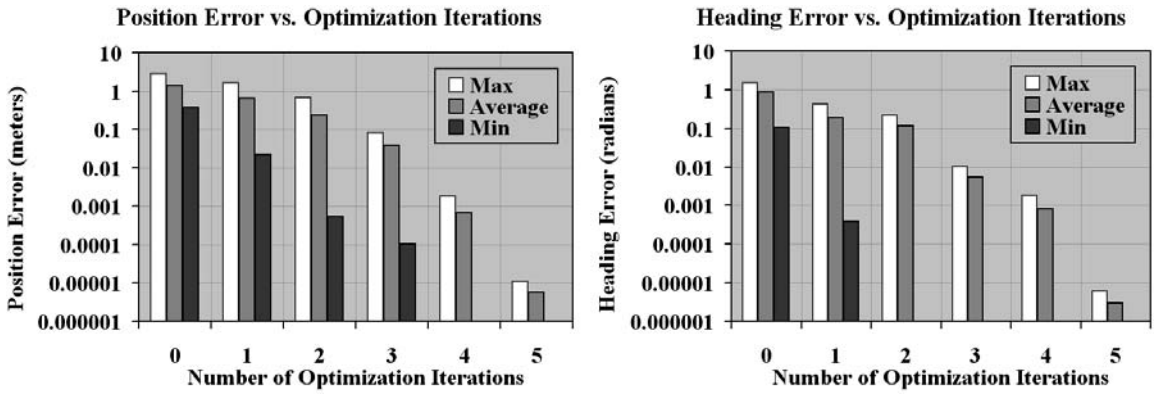

Fig. 15. Trajectory generator convergence for instrument placement tasks using all-wheel steering mobility systems. For the sequence of trajectories shown in Figure 14, the maximum, average, and minimum position and heading errors of the seven planned trajectories are shown as a function of the number of iterations executed by the trajectory generation algorithm.

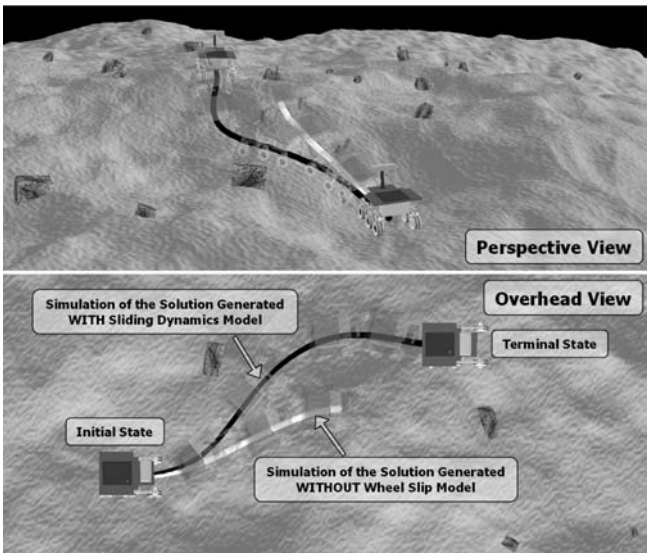

Fig. 16. Compensating for wheel slip in trajectory generation. Wheel slip models can be included in the vehicle model used by the trajectory generator to plan motions. Notice that the forward simulation of the solution generated by the motion without a wheel slip model slides moves significantly less than predicted and it does not reach the target terminal state. Improved performance of path tracking algorithms can be expected by modeling these effects instead of relying on feedback control to account for such errors.

the hill. This effect can easily be modeled as a velocity proportional to the gradient of the terrain, resulting in a planned trajectory that drives up the hill to compensate for the downward sliding effects.

In this example, the trajectory generator was able to meet the boundary state constraints in five iterations of the algorithm. Figure 19 shows the commanded and response velocities for the modeled and unmodeled vehicle dynamics solutions. Notice that the unmodeled solution does nothing to account for sliding down the hill, so the angular velocity remains constantly at zero (the terminal state is straight ahead of the initial state). The solution that incorporates these effects understands that to compensate for these effects, it must initially turn up against the slope, and hence it has a non-zero angular velocity profile.

5.6. Rough Terrain Trajectory Generation using Constrained Optimization

The optimal control formulation is applicable whenever there are sufficient degrees of freedom to optimize something. We illustrate two different cases of trajectory optimization, specifically using minimum-cost and minimum-slope-dwell utility functionals. It is important to note that these trajectories are optimal only over the space of feasible motions spanned by the polynomial control set. However, we have also argued that this space is a very good approximation to the continuous function space of arbitrary controls.

5.6.1. Minimum-cost Performance Index

In this type of formulation, obstacles are represented as costs in a field that the robot traverses. If a high-cost obstacle is observed to be in the planned trajectory, typically the vehicle must either replan at the global level or temporarily swerve

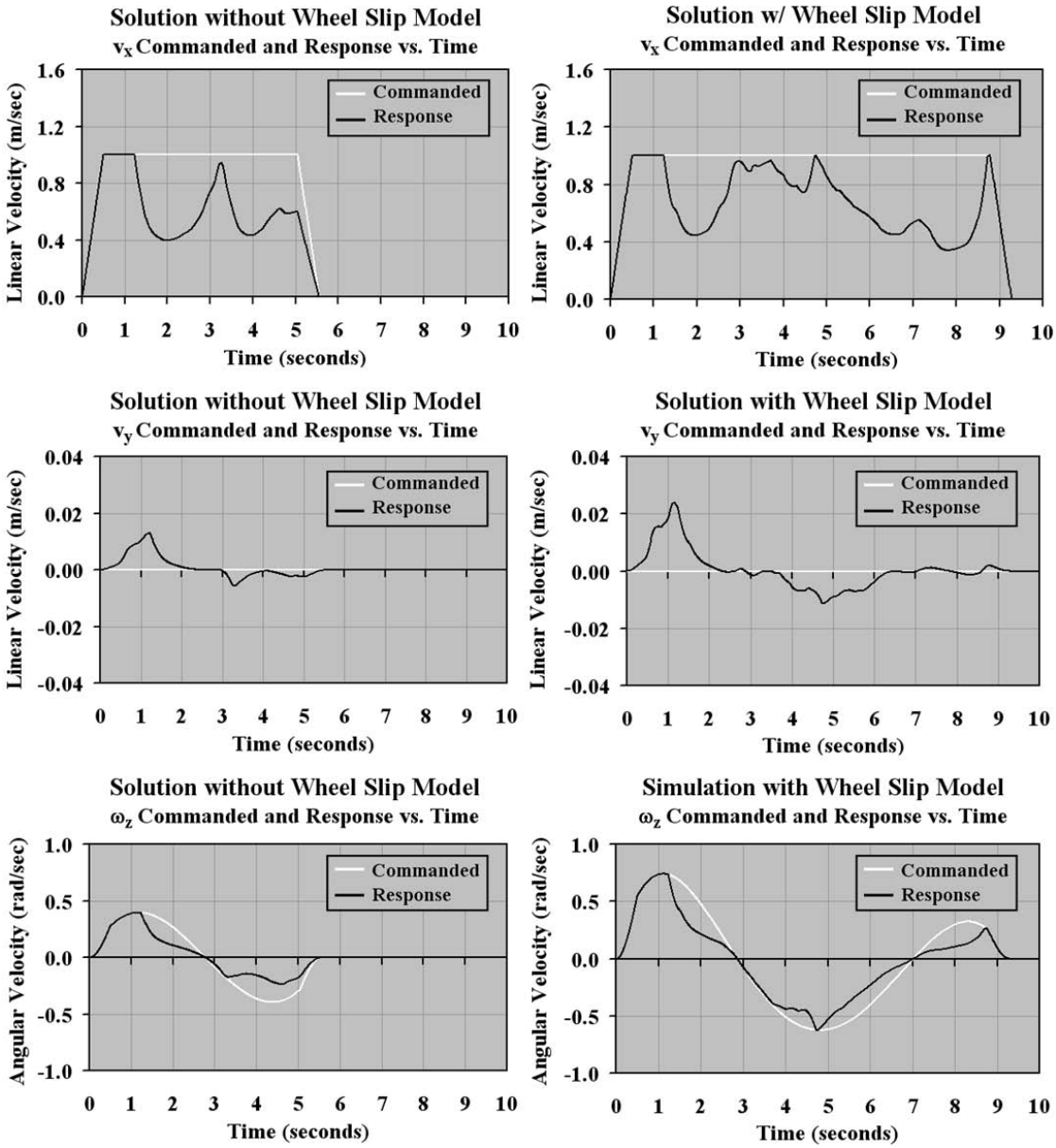

Fig. 17. Compensating for wheel slip in trajectory generation. Wheel slip approximations can be used in the forward model to enable predictive compensation for these effects in the trajectory generator. The plots on the left show the commanded and response linear and angular velocities of the unmodelled solution. The plots on the right show the commanded and response linear and angular velocities of the solution that models these effects. Notice that the solution that accounts for wheel slip takes longer to execute (9.21 s vs. 6.16 s) because the response net velocities on the slopes are always less than the commanded net velocities.

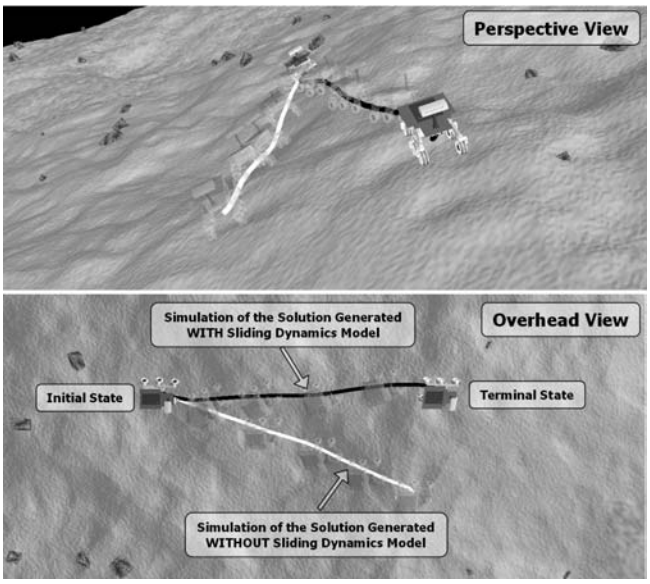

Fig. 18. Compensating for sliding dynamics in trajectory generation. Sliding dynamics models can be included in the vehicle model used by the trajectory generator to plan motions. Notice that the forward simulation of the solution generated by the motion without a sliding dynamics model slides down the slope and does not reach the target terminal state. Improved performance of path tracking algorithms can be expected by modeling these effects instead of relying on feedback control to compensate for such errors.

from the target path and reacquire it behind the obstacle. The swerve motion can be generated dynamically in the optimal control formulation of trajectory generator, where a tradeoff between minimum-cost and minimum-time is encoded in the overall cost functional.

To find a best compromise, the utility functional $(Y(\mathbf{x}, \mathbf{p}, t))$ is defined as the sum of 1 and the weighted value of the cost map at the current position (x, y).

$$J(\mathbf{u}) = \int_0^{t_f} Y(\mathbf{x}, \mathbf{u}, t)\, dt = \int_0^{t_f} 1 + \alpha \,\mathrm{cost}(x, y)\, dt \quad (35)$$

The "1" term in the integral represents the time that it takes to traverse a cell and the "cost(x, y)" is the cost of its traversal. The parameter α controls the trade-off between finding the shortest path and the one that minimizes the cost line integral. As α approaches zero, the minimum-time path will be found. Conversely as α approaches infinity, the locally minimum cost path will be found.

A straightforward example demonstrating the effectiveness of this utility function is that of avoiding a simulated localized obstacle. A cost field can be generated by penalizing proximity to the obstacle as shown in Figure 20.

The solution for zero α is trivial, there is no cost contribution from the simulated obstacle and the minimum-time solution (a straight-line path directly to the terminal state) is found. As α is increased, the paths deflect away from the high cost regions at the cost of longer path lengths. This example is applicable, for example, to operator interfaces where the operator might designate the obstacle by clicking at its location.

5.6.2. Minimum-slope-dwell Performance Index

For planetary rovers operating in rough terrain, it is sometimes necessary to minimize the amount of time spent on slopes to reduce risk. We can therefore define a utility function that penalizes high roll and pitch values along the path to avoid slopes:

$$J(\mathbf{u}) = \int_0^{t_f} Y(\mathbf{x}, \mathbf{u}, t)\, dt = \int_0^{t_f} 1 + \alpha\left(\phi^2 + \theta^2\right) dt \quad (36)$$

Just as with the minimum-cost example from Section 5.6.1, the "α" term represents the trade-off between the minimum-time and minimum-slope-dwell solutions. Since the numerical techniques that we use are descent algorithms, we will find only local optima so the technique cannot be used without modification when many such minima are prevalent.

To demonstrate the use of this utility functional, we try to find the shortest path that drives over a large pyramid-shaped hill. On flat terrain, the trajectory is trivial—it is a straight line between the two states. However, driving on the side of a hill can be dangerous (tip-over, sliding, loss of traction, etc.). We get a straight-line motion when our weight (α) is equal to zero because it is the minimum-time solution to the problem. As the weight (α) is increased toward unity, the algorithm converges to a path that moves around most of the hill. It does not plan entirely around the hill because it is still looking for the shortest solution subject to this penalty for high attitude. An increase of the weight (α) towards two causes the path to avoid more of the hill at the cost of increasing the time to the goal. Figure 21 shows how the paths differ from one another given the different weights in this formulation of the problem.

5.7. Runtime Performance

The complexity of the vehicle model and the terrain shape used in trajectory generation has significant effects on the expected runtime of the algorithm. This section benchmarks the expected performance with the presented rough terrain trajectory generation algorithm with a real vehicle model. All tests were run on a 1.866 GHz Pentium M notebook computer with 1 GB of RAM.

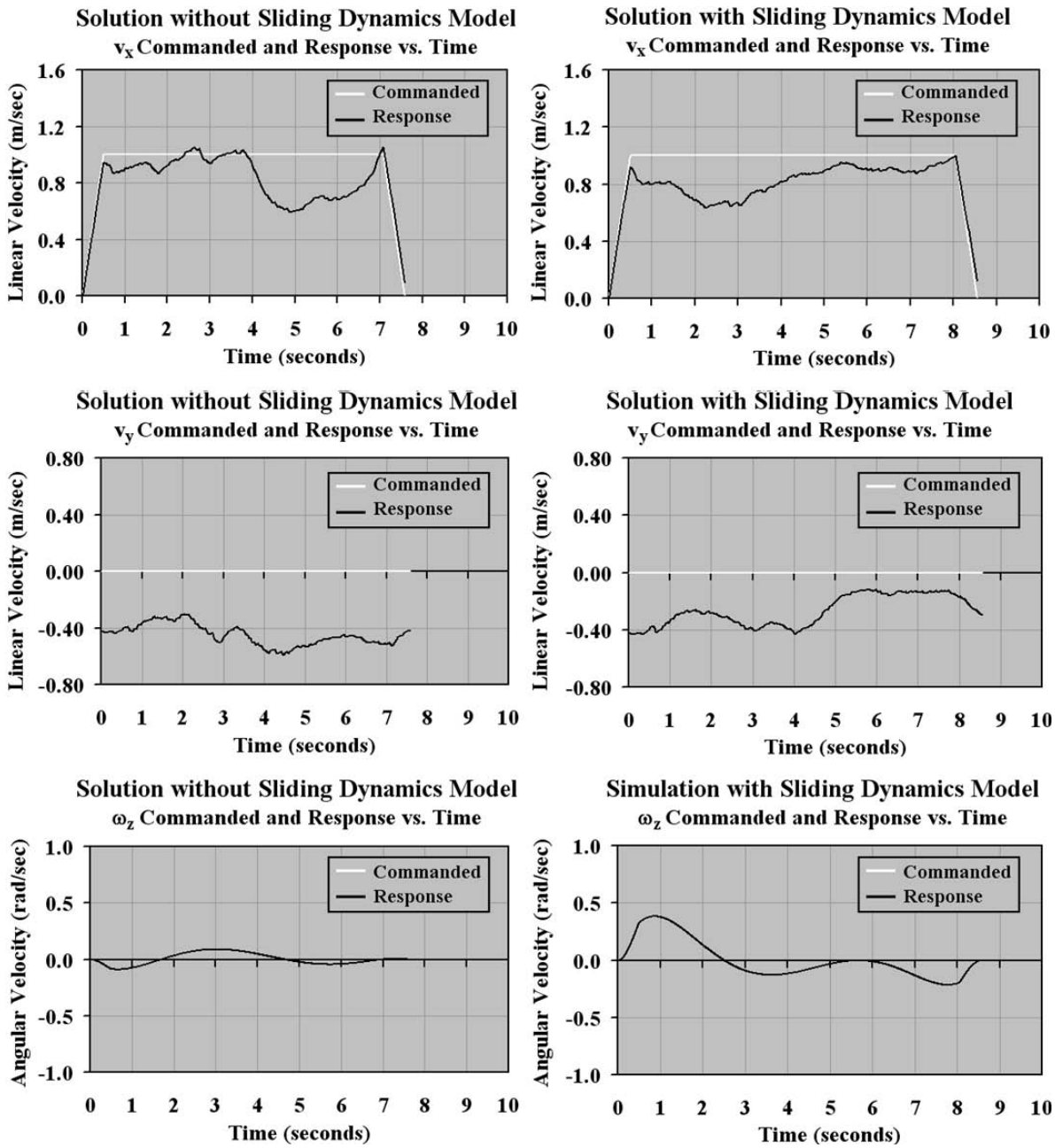

Fig. 19. Compensating for vehicle sliding dynamics in trajectory generation. Vehicle sliding dynamics models can be incorporated in the forward model of the vehicle to predictably compensate for these effects in trajectory generation.

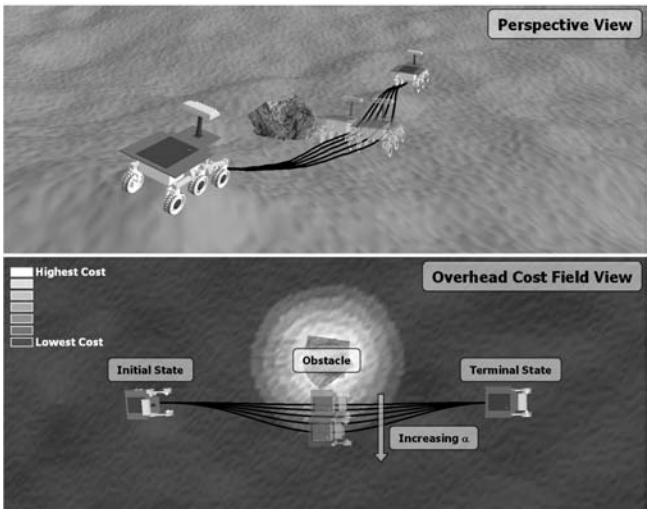

Fig. 20. Minimizing path cost. The optimal control formulation of the trajectory generator can be used to minimize an arbitrary path cost function over the course of a trajectory. This example demonstrates avoidance of high cost regions (proximity to a simulated obstacle) at the expense of longer path lengths.

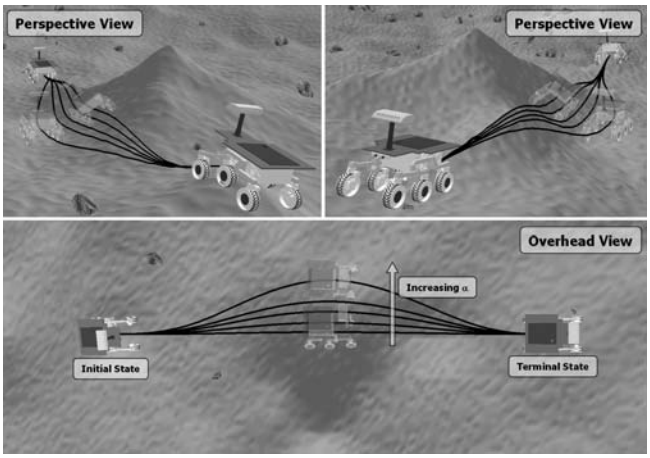

Fig. 21. Minimizing slope dwell. Just as in the minimum path cost example from Section 5.6.1, the optimal control formulation of the trajectory generation algorithm can be used to mitigate risk involving high slopes when navigating rough terrain. This example demonstrates how increasing the slope weight (a) causes the trajectory to avoid the hill at the cost of longer path lengths.

Runtime can be very dependent on the terrain roughness because of the complexity of the suspension model used. At each step of the numerical integration, the elevation, attitude, and the new suspension angles must all be computed via nonlinear optimization. In order to investigate the effect of terrain roughness on runtime, we ran 4,096 trajectory generation queries using the Rocky 8 vehicle model on a series of fifty different terrains of increasing roughness. Each world was generated by scaling a single elevation map by roughness index. Views of the different height maps used throughout the tests are shown in Figure 22 and the average runtime vs. terrain roughness plot is shown in Figure 23.

The average runtime of the algorithm is observed to increase proportionally with terrain roughness. This is expected because the suspension optimization must perform more iterations to adapt the vehicle to rough terrain and the initial guess of the control parameters (which are based on the flat terrain solutions) are going to be worse on rougher terrain.

It is important to note that the complexity can be scaled appropriately depending on the computational capabilities of the platform.

6. Applications

Our algorithm can endow a mobile robot with an unprecedented capability to predict the consequences of its own actions in relatively challenging environments. Using it, we can expect intelligent behaviors in applications such as local planning and obstacle avoidance, path following, and global planning as outlined in this section.

6.1. Local Motion Planning

The most obvious application for the developed trajectory generation method is local motion planning in complex environments.

One such approach to local motion planning is ego-graphs, which are body-centered search spaces often used for obstacle avoidance (Lacaze et al. 1998). The presented algorithm is highly effective for generating ego-graphs (Figure 24) because it is efficient and it can encode vehicle dynamics constraints.

Ego-graphs can be initially generated with a flat terrain assumption and subsequently adapted to arbitrary rough terrain and dynamics models in order to better evaluate feasibility and true path cost (Figure 25). Since the ego-graphs generated using this algorithm are simply a collection of paths solved between pairs of boundary states, the algorithm can efficiently adapt the solutions to rough terrain using the flat terrain solution as its initial guess.

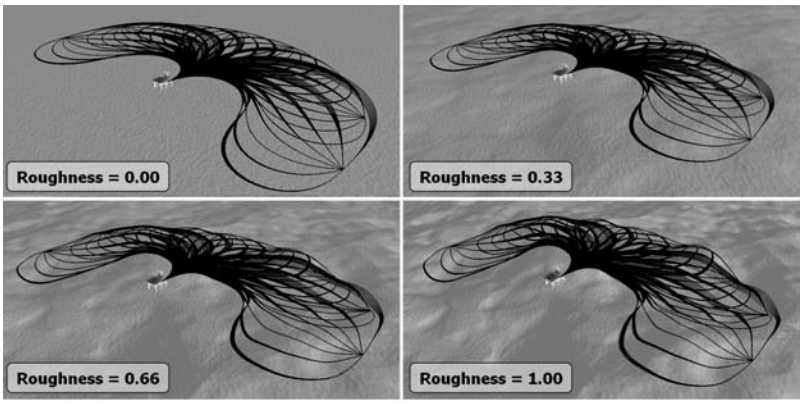

Fig. 22. Terrain roughness tests. In order to conduct a fair series of runtime vs. terrain roughness tests, motions plans are generated on a series of terrains of increasing roughness ranging from a roughness index of 0.0 (nominally flat terrain) to 1.0 (very rough terrain).

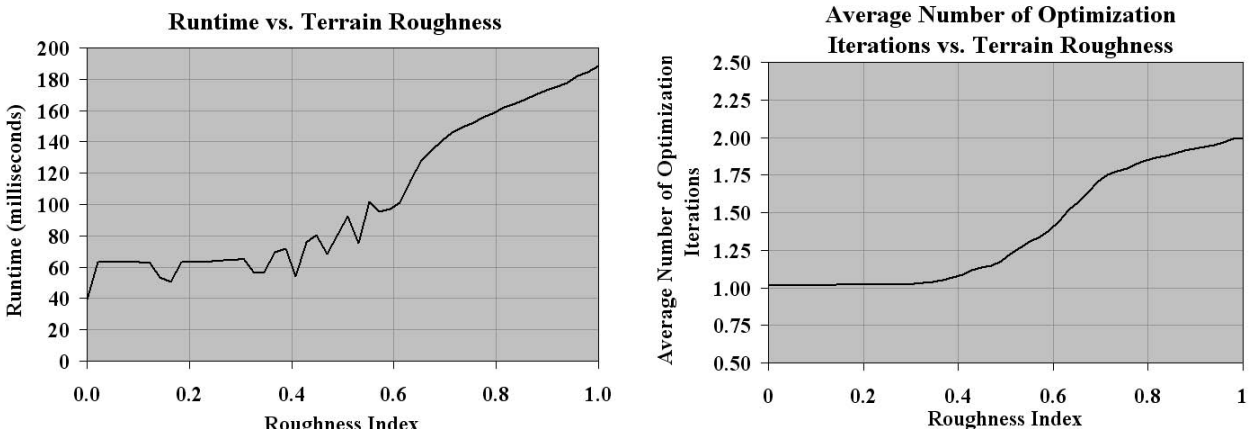

Fig. 23. Runtime vs. terrain roughness. A series of tests were conducted to determine the effect of terrain roughness on the runtime of the algorithm. The number of optimization steps required by the algorithm and hence the runtime of the algorithm is observed to increase proportionally with respect to the terrain roughness.

6.2. Path Following

Path following is the problem of finding vehicle controls which will allow the mobile robot to track its target path. Typically path following algorithms rely on a relatively fast feedback control loop using a low-order motion primitive that reacquires the target path at some forward point (e.g., pure pursuit). This method has been implemented successfully for many years because a fast update rate can compensate for unmodeled errors.

A more robust path tracking algorithm can generate vehicle controls based on realistic models of dynamics and wheel-terrain interaction as in Howard et al. (2006). In this approach, a set of candidate path following motions are generated by the optimal rough terrain trajectory generation algorithm that reacquire the path at some forward vehicle posture, ensuring position, heading, and curvature continuity (Figure 26). An optimal selection of the corrective trajectory is chosen that minimizes some utility function based on cross-track error, smoothness, and any other arbitrary factors (Figure 27).

6.3. Global Motion Planning

Trajectory generation can be used to create an inherently feasible search space for global motion planning. One technique, illustrated in Figure 28, is to create a regular lattice of states

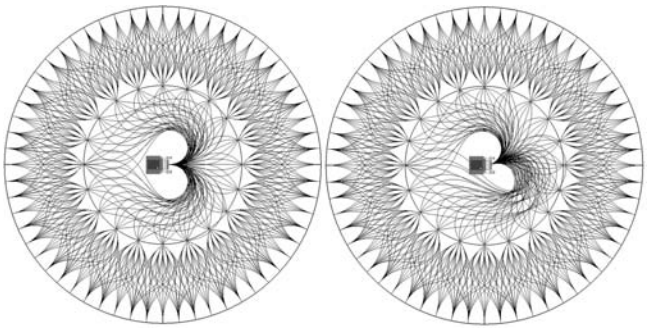

Fig. 24. Ego-graph generation. The presented trajectory generation algorithm can be used to generate ego-graphs, which are body-frame fixed obstacle avoidance and local navigation search spaces. The above two ego-graphs, comprised of 630 individual trajectories each, were generated with different initial states using the present algorithm. Note that only the first level of the search space is dependent on the current vehicle state.

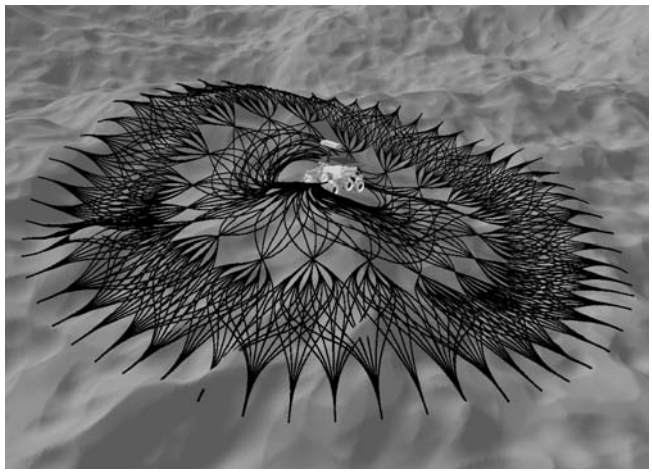

Fig. 25. Terrain-adaptive ego-graphs. The ego-graphs generated in Figure 21 can be adapted to rough terrain using the developed trajectory generation algorithm in order to better evaluate dynamic feasibility and path cost. The fact that individual trajectories all pass exactly through the intended terminal states indicates compensation for terrain shape.

and to connect them with feasible motions that serve as the edges used to transition between states. Initially, the edges can be generated based on a flat terrain lookup table and very few distinct shapes are needed since the node relationships are symmetric under translations and rotations whose magnitudes are consistent with the cell sizes.

Thereafter, the edges can be adapted individually to terrain shape to enforce continuity once they are actually traversed in search, and potentially once again when perception information refines the terrain shape during plan execution.

7. Conclusions

This article has presented a highly generic approach to trajectory generation for mobile robots of somewhat arbitrary mobility characteristics. Such a general formulation is proposed because it permits the predictive elimination of model errors at planning time rather than reactive elimination at execution time. To the degree that models of terrain following, vehicle dynamics, and wheel/terrain interaction have any utility at all, our approach can extract and exploit the signal in such models while leaving the remaining unpredictable components to be compensated during control execution.

This algorithm can endow a mobile robot with an unprecedented capacity to predict the consequences of its own actions and to take corrective actions in relatively challenging environments. Using it, we can expect such intelligent behaviors as the following:

An automated passenger bus can start turning early by precisely the correct amount in order to change lanes in traffic because the actuator dynamic models model the sluggishness of the steering system.

A planetary rover that can exploit its predictions of wheel slip by approaching a hillside goal on the high side in anticipation of sliding into the goal as the trajectory executes.

An unmanned ground vehicle executing a sweep for buried mines can alter its trajectory in response to real-time high-resolution terrain information. Such alteration will ensure that the present sensor swath is precisely parallel to the last, regardless of how poor the aerial terrain map generated by the deployment aircraft may have been.

Performance in real applications is of course subject to the fidelity of the models being inverted. Issues of model fidelity are outside the scope of this work. However, even imperfect nominal models are better that the complete lack of a model which characterizes the state of the contemporary art with respect to the elements introduced here.

Current and future work includes the use of this algorithm in multiple contexts. First, it is being applied in the generation of corrective maneuvers in rough terrain path following applications. Second, it is being used to generate well-separated trajectory sets for obstacle avoidance computations on rough terrain vehicles. In addition, it is being evaluated and refined as the search space generation component of a larger scale nonholonomic motion planner.

Acknowledgments

This research was conducted at the Robotics Institute of Carnegie Mellon University under contract to NASA/JPL as part of the Mars Technology Program.

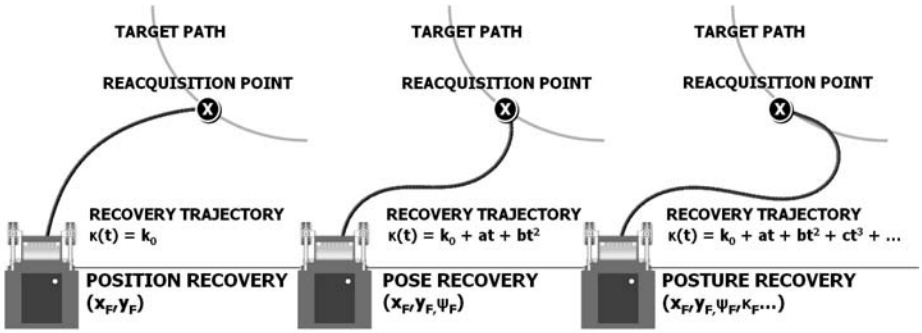

Fig. 26. Corrective trajectory continuity. This figure demonstrates several different trajectory generation boundary conditions that lead to increasingly higher levels of continuity for corrective trajectories. The corrective trajectory becomes increasingly complicated at the benefit of higher degrees of terminal state continuity.

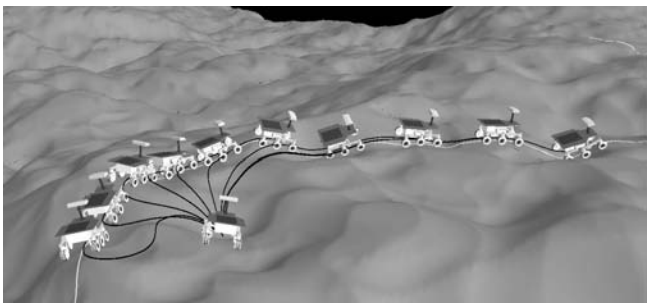

Fig. 27. Reacquiring the target path. The rough terrain trajectory generation algorithm can be employed in a constrained optimization sense to determine the optimal corrective path to reacquire the target path. A cost function based on cross-track error, smoothness, and other factors is minimized in order to determine the optimal corrective trajectory.

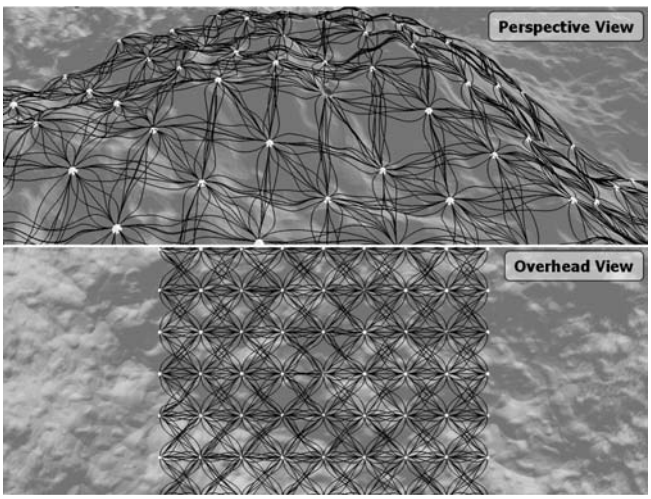

Fig. 28. Connectivity of a state space lattice in rough terrain. The states of a lattice which is equally distributed in (x, y) can be connected with minimum-time paths in general rough terrain using our trajectory generator.

References

Amar, F. B., Bidaud, Ph., and Ouezdou, F. B. (1993). On modeling and motion planning of planetary vehicles. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vol. 2, pp. 1381–1386.

Angelova, A., Matthies, L., Helmick, D., and Perona, P. (2006). Slip prediction using visual information. In *Proceedings of Robotics Science and Systems*, Philadelphia, USA.

Baker, D. (1989). Exact solutions to some minimum-time problems and their behavior near inequality state constraints. *IEEE Transactions on Automatic Control*, **34**(1): 103–106.

Barraquand, J. and Latombe, J. C. (1989). On nonholonomic mobile robots and optimal maneuvering. *Revue d'Intelligence Artificielle*, **3**(2): 77–103.

Betts, J. T. (1998). Survey of numerical methods for trajectory optimization. *Journal of Guidance, Control, and Dynamics*, **21**(2): 193–207.

Biesiadeki, J., Leger, C., and Maimone, M. (2005). Tradeoffs between directed and autonomous driving on the mars exploration rovers. *Proceedings of the International Symposium on Robotics Research*, San Francisco, USA.

Bonnafous, D., Lacroix, S., and Simeon, T. (2001). Motion generation for a rover on rough terrain. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robotics and Systems*, Vol. 2, pp.784–789.

Brockett, R. W. (1981). Control theory and singular riemann geometry. In *New Directions in Applied Mathematics*. New York: Springer-Verlag, pp. 11–27.

Canny, J., Donald, B., Reif, J., and Xaiver, P. (1988). On the complexity of kinodynamic planning. *Proceedings of the 29th IEEE Symposium on Foundations of Computer Science*, pp. 306–318.

Cherif, M. (1999). Motion planning for all-terrain vehicles: a physical modeling approach for coping with dynamic and contact interaction constraints. *IEEE Transaction on Robotics and Automation*, **15**(2): 202–218.

Cherif, M., Laugier, Ch., Milesi-Bellier, Ch., and Faverjon, B. (1994). Planning the motions of an all-terrain vehicle by using geometric and physical models. *Proceedings of IEEE International Conference on Robotics and Automation*, Vol. 3, pp. 2050–2056.

Delingette, H., Herbert, M., and Ikeuchi, K. (1991). Trajectory generation with curvature constraint based on energy minimization. *Proceedings of the IEE-RSJ International Conference on Intelligent Robots and Systems*, Vol.1, pp. 206–211.

Dubins, L. E. (1957). On curves of minimal length with a constraint on average curvature and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, **79**: 497–516.

Faiz, N., Agrawal, S., and Murray, R. (2001). Differentially flat systems with inequality constraints: an approach to real-time feasible trajectory generation. *Journal of Guidance, Control, and Dynamics*, **24**(2): 219–227.

Fliess, M., Levine, J., Martin, Ph. and Rouchon, P. (1995). Flatness and defect of nonlinear systems: introductory theory and examples. International. *Journal of Control*, **61**(6): 1327–1361.

Gaw, D. and Meystel, A. (1986). Minimum-time navigation of an unmanned mobile robot in a 2½D world with obstacles. *Proceedings of the IEEE International Conference on Robotics and Automation*, Vol. 3, pp. 1670–1677.

Heinzinger, G., Jacobs, P., Canny, J., and Paden, B. (1990). Time-optimal trajectories for a robot manipulator: A provably good approximation algorithm. In *Automatic Control Conference*.

Howard, T. M. and Kelly, A. (2006). Trajectory and spline generation for all-wheel steering mobile robots. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robotics and Systems*, Beijing, China, October.

Howard, T. M. and Kelly, A. (2005a). Trajectory generation on rough terrain considering actuator dynamics. *Field and Service Robotics 2005,* Port Douglas, Australia, July.

Howard, T. M. and Kelly, A. (2005b). Terrain-adaptive generation of optimal continuous trajectories for mobile robots. *International Symposium on Artificial Intelligence, Robotics, and Automation in Space 2005*, Munich, Germany, October.

Howard, T. M. Knepper, R. A., and Kelly, A. (2006). Constrained optimization path following of wheeled mobile robots in natural terrain. *International Symposium on Experimental Robotics 2006*, Rio de Janeiro, Brazil, July.

Jacobs, P. and Canny, J. (1989). Planning smooth paths for mobile robots. *Proceedings of the IEEE International Conference. on Robotics and Automation*, pp. 2–7.

Jackson, J. W. and Crouch, P. E. (1991). Curved path approaches and dynamic interpolation. *IEEE Aerospace and Electronic Systems Magazine*, **6**(2): 8–13.

Kalmár-Nagy, T., D'Andrea, R., and Ganguly, P. (2004). Near-optimal dynamic trajectory generation and control of an omnidirectional vehicle. *Robotics and Autonomous Systems*, **46**: 47–64.

Kanayama, Y. and Miyake, N. (1985). Trajectory generation for mobile robots. *Proceedings of the International Symposium on Robotics Research*, pp. 16–23.

Kanayama, Y. and Hartman, B. I. (1989). Smooth local path planning for autonomous vehicles. *Proceedings of the International Conference on Robotics and Automation*, Vol. 3, pp. 1265–1270.

Kelly, A. and Nagy, B. (2003). Reactive nonholonomic trajectory generation via parametric optimal control. *International Journal of Robotics Research*, **22**(7-8): 583–601.

Kim, S. K. and Tilbury, D. M. (2001). Trajectory generation for a class of nonlinear systems with input and state constraints. *Proceedings of the American Control Conference*, Vol. 6, pp. 4908–4913.

Komoriya, K. and Tanie, K. (1989). Trajectory design and control of a wheel-type mobile robot using B-spline curve. In *Proceedings of the IEE-RSJ International Conference on Intelligent Robots and Systems*, pp. 398–405.

Lacze, A., Moscovitz, Y., DeClaris, N. and Murphy, K. (1998). Path planning for autonomous vehicles driving over rough terrain. *Proceedings of the 1998 IEEE ISIC/CIRA/ISAS Joint Conference.* More detail required

Laumond, J. P., Taix, M., and Jacobs, P. (1990). A motion planner for car-like robots based on a mixed global/local approach. *IEEE International Workshop on Intelligent Robots and Systems*, Tsuchira, Japan, pp. 765–773.

Laumond, J. P. (1995). Nonholonomic motion planning via optimal control. *Proceedings of the Workshop on Algorithmic Foundations of Robotics*, San Francisco, USA, pp. 227–238.

Lin, C. S., Chang, P. R. and Luh, J. Y. S. (1983). Formulation and optimization of cubic polynomial joint trajectories for industrial robots. *IEEE Transactions on Automatic Control*, **AC-23**: 1066–1074.

Murray, R. and Sastry, S. (1998). Nonholonomic motion planning: steering using sinusoids. *IEEE Transactions on Automatic Control*, **38**: 700–716.

Nagy, B. and Kelly, A. (2001). Trajectory generation for car-like robots using cubic curvature polynomials. *Field and Service Robotics 2001* (FSR '01), June.

Nesnas, I., Maimone, M. and Das, H. (2000). Rover maneuvering for autonomous vision-based dexterous manipulation. *IEEE Conference on Robotics and Automation*, San Francisco, CA, Vol. 3, pp. 2296–2301.

Pivtoraiko, M. and Kelly, A. (2005). Efficient contstrained path planning via search in state lattices. *International Symposium on Artificial Intelligence, Robotics, and Automation in Space*, Munich, Germany.

Reeds, J. A. and Shepp, L. A. (1990). Optimal paths for a car that goes both forward and backward. *Pacific Journal of Mathematics*, **145**(2): 367–393.

Reuter, J. (1998). Mobile robot trajectories with continuously differentiable curvature: an optimal control approach. *Proceedings of the IEEE/RSI Conference on Intelligent Robots and Systems*, Victoria, BC, October, Vol. 3, pp. 38–43.

Shiller, Z. and Chen, J. C. (1990). Optimal motion planning of autonomous vehicles in three-dimensional terrains. *Proceedings of the IEEE International Conference on Robotics and Automation*, Cincinnati, Vol. 1., pp. 198–203.

Shiller, Z. and Gwo, Y. R. (1991). Dynamic motion planning of autonomous vehicles. *IEEE Transactions on Robotics and Automation*, **7**(2): 241–249.

Shin, D. H. and Singh, S. (1990). Path generation for robot vehicles using composite clothoid segments. Tech Report, CMU-RI-TR-90-31, The Robotics Institute, Carnegie Mellon University.

Tarokh, M. and McDermott, G. (2005). Kinematics modeling and analyses of articulated rovers. Technical Report No. CS/10/2005.

Tilbury, D., Laumond, J. P., Murray, R., Sastry, S. and Walsh, G. (1992). Steering car-like systems with trailers using sinusoids. *Proceedings of the IEEE International Conference. on Robotics and Automation*, Vol. 3, pp. 1993–1998.