JOURNAL OF AEROSPACE COMPUTING, INFORMATION, AND COMMUNICATION
Vol. 6, March 2009

Preference-Based Trajectory Generation

Jamie A. Lennon*
Naval Research Laboratory, Washington, DC 20735

and

Ella M. Atkins†
University of Michigan, Ann Arbor, MI 48109

DOI: 10.2514/1.36214

Numerous techniques exist to optimize aircraft and spacecraft trajectories over cost functions that include terms such as fuel, time, and separation from obstacles. Relative weighting factors can dramatically alter solution characteristics, and engineers often must manually adjust either cost weights or the trajectory itself to obtain desirable solutions. Further, when humans and robots work together, or when humans task robots, they may express their performance expectations in a "fuzzy" natural language fashion, or else as an uncertain range of more-or-less acceptable values. This work describes a software architecture which accepts both fuzzy linguistic and hard numeric constraints on trajectory performance and, using a trajectory generator provided by the user, automatically constructs trajectories to meet these specifications as closely as possible. The system respects hard constraints imposed by system dynamics or by the user, and will not let the user's preferences interfere with the system and user needs. The architecture's evaluation agent translates these requirements into cost-functional weights expected to produce the desired motion characteristics. The quality of the resulting full-state trajectory is then evaluated based on a set of computed trajectory features compared to the specified constraints. If constraints are not met, the cost-functional weights are adjusted according to precomputed heuristic equations. Heuristics are not generated in an ad hoc fashion, but are instead the result of a systematic testing of the simulated system under a range of simple conditions. The system is tested in a two degree of freedom (2DOF) linear and a 6DOF nonlinear domain with a variety of constraints and in the presence of obstacles. Results show that the system consistently meets all hard numeric constraints placed on the trajectory. Desired characteristics are often attainable or, in those cases where they are discounted in favor of the hard constraints, fail by small margins. Results are discussed as a function of obstacles and of constraints.

Nomenclature

$\boldsymbol{a}$	dynamic equations for physical system
bc	boundary conditions on trajectory-optimization problem
D	domain of planning problem
F^i	trajectory feature vector for planning state s_i

Received 14 December 2007; revision received 1 October 2008; accepted for publication 24 November 2008.

* Aerospace Engineer, Naval Research Laboratory, Code 8231, 4555 Overlook Ave. SW, Washington, DC 20735, jamie.lennon@nrl.navy.mil, Member.

† Associate Professor, Aerospace Engineering, 3009 FXB Building, 1320 Beal Avenue, University of Michigan, Ann Arbor, MI 48109, ematkins@umich.edu, Associate Fellow.

g	cost functional for optimization
$\mathcal{H}$	Hamiltonian of the optimization problem
$\mathbf{H}^0$	set of user-imposed hard constraints on planning problem
$\mathbf{HIST}^i$	eecord of variables used to generate solution to optimization problem
$J(\boldsymbol{x}, \boldsymbol{u}, t, \{\boldsymbol{\Omega}^i\})$	domain-dependent multi-objective cost function with weights $\boldsymbol{\Omega}^i$
J^i	integrated cost over trajectory $(\boldsymbol{x}^i, \boldsymbol{u}^i, \boldsymbol{t}^i)$
$\mathbf{L}^i$	feature vector limits (constraints) for planning state s_i ($\mathbf{L}^0$ = initial/default limit set)
$\boldsymbol{\lambda}$	costate of the dynamic system
$\{\mathbf{O}\}$	set of k obstacles $\{o_1, o_2, \ldots, o_k\}$
$\mathbf{P}_0$	trajectory planning problem $\langle \mathbf{bc}, \boldsymbol{\Omega}^0, \mathbf{L}^0 \rangle$ with boundary conditions $\mathbf{bc} = \langle t_0, \boldsymbol{x}_0, \boldsymbol{x}_f \rangle$
$\mathbf{S}^0$	set of user-imposed soft constraints on planning problem
t	time
$\boldsymbol{t}^i$	vector of trajectory time points $\{t_1, \ldots, t_m\}$ for planning state s_i
$\boldsymbol{u}$	control actuation vector at single time t
$\boldsymbol{u}^i$	control actuation vector over the m trajectory time points for planning state s_i
$\mathbf{V}$	fuzzy language database which converts words into trajectory features
$\boldsymbol{\Omega}^i$	cost function weighting factor vector used in planning state s_i
$\boldsymbol{x}$	position/velocity state vector at a single time t
$\boldsymbol{x}^i$	position/velocity state vector over the m trajectory time points for planning state s_i
$\mathbf{X}$	solution $\langle J^n, \mathbf{L}^n, \boldsymbol{t}^n, \boldsymbol{x}^n, \boldsymbol{u}^n \rangle$ returned for planning problem $\mathbf{P}_0$
$\mathbf{Z}$	set of fuzzy logic rules

I. Introduction

INTELLIGENT robotic systems will play an important role in future space and planetary surface operations. Whether exploring on their own or accompanying and supporting human pioneers, they will need the capability to reason, plan ahead, and make decisions based on goals, the environment, and the desires of human or robotic teammates. Embodied robots must also translate mission goals into appropriate physical responses.

Balancing competing costs, while satisfying certain hard constraints, is an important component of "appropriateness." In space exploration problems, fuel and power conservation are dominant issues, whether the agent under discussion has a limited tank of fuel for positioning or, despite recharging capability, has a limited power budget constrained by battery weight. Timeliness is also a concern, as many scientists may wish to use a vehicle's capabilities for a variety of projects before its life span ends. Preserving vehicle health is another priority, and all of this must be done while respecting the dynamic constraints of the vehicle, and the dynamic properties of its environment.

Human users, whether on-site astronauts or ground-based controllers, might express their desired balance of these costs linguistically: "get this done quickly," "be very careful," "maximize your range." When communicating with other humans, this works very well—often better than attempting to fix numerical values to desired characteristics. As humans, we are very adept at giving and understanding these linguistic expressions even if we cannot precisely describe them in objective terms.

We can, for example, easily identify "aggressive driving" when we see it on the roads. An aggressive driver's behavior is marked by high traveling speeds, frequent lane changes, sudden accelerations, and the maintenance of slim safety margins to other vehicles. What is a "high traveling speed?" Even once the context is fixed (e.g., interstate vs in-town), the linguistic term has some fuzziness to it. Certainly, it implies a speed higher than the legal, posted speed limit. It probably means a speed higher than the average speed of the other drivers. But is someone driving 5 mph faster than road speed an aggressive driver? And on the other extreme, is there a speed so high that we can say that we have gone past "aggressive driving" and are into a region of "reckless driving?" When, exactly, is that line crossed? The answers to these questions are easy for humans to intuit, but difficult to formalize.

These concerns follow us into the realm of trajectory optimization. Robot trajectories need not be optimal. In some domains, we may accept satisficing trajectories that are adequate but not optimal. But in the space domain in particular, we will always be concerned with conserving fuel and power. Even if we want an "aggressive, fast"

trajectory, we will still want it to be the most fuel-efficient aggressive trajectory. We are concerned with fuel even if the result is not the fuel-optimal solution.

Trajectory optimization, at its most general, will have multiple objectives and constraints. Multiple objectives in particular give rise to multiple possible solutions. Consider a two-objective case: we desire to save both time and fuel. These objectives compete. The most fuel-efficient trajectory is rarely the most time-efficient, and visa versa. There might be several solutions that take the same minimum time, and we would be interested in the most fuel-efficient one. Or, we might examine all of the minimum fuel solutions and pick the fastest of those. Or, we might want an intermediate solution—one that is neither the fastest nor the most fuel-efficient, but that balances the two objectives. We may have an initial estimate of relative importance in saving time or fuel. How can we communicate such preferences to the numeric trajectory solver so that it will find the "right" optimal?

In addition to these fuzzy preference ideas, we may also have trajectory constraints, typically imposed to meet dynamic performance limits and to ensure safety (collision avoidance). Beyond that, we might further impose limits on either control inputs (e.g., actuator travel limits or thruster saturation) and on system state. Some of these limits could be "soft," like a posted speed limit that can be exceeded given appropriate circumstances (e.g., to avoid an erratic driver). Other limits are "hard," such as acceleration limits imposed to avoid pilot/astronaut blackout.

There are many approaches to solving the constrained multi-objective optimization problem [1], including evolutionary algorithms, mixed integer linear programming (MILP), and optimal control theory. To a greater or lesser extent, each handles soft and hard constraints. All, however, include an iterative refinement loop to find solutions that best match user preferences: that is, to find the "optimal optimal" solution. As a common thread throughout the multi-objective optimization literature, it is tacitly understood that a human user interacts directly with the optimization algorithm, injecting preference information to focus the optimization on areas of interest to the user.

This work seeks to take the human user out of that loop as much as possible. Before optimization ever begins, classes of motion are typified with linguistic expressions: aggressive, curious, careful. Fuzzy logic [2] is an appropriate tool for approaching the problem of translating natural language utterances into numeric terms [3]. Words are correlated to fuzzy state values the system believes best represent user preferences for the resulting trajectory: the numbers we need to take the human supervisor out of the optimization loop. Some iteration may be required to ensure that the user's expectation matches the fuzzy definition of the linguistic expressions. However, once that process is complete, the autonomous "supervisor" can balance optimization objectives without user input.

Here we consider a planetary rover and an Earth-observing satellite as motivating examples. The planetary rover example is a very simplified case with two degree-of-freedom (2DOF) and linear dynamics that provides initial insight into the algorithms required to inject preferences into the trajectory-generation process. The satellite case has more realistic six degree-of-freedom (6DOF) dynamics. The satellite's hypothetical job is to provide imaging to support ground-based decision making. It can execute fuel burns to change its orbit in response to user demands on the ground. These user demands may have varying levels of urgency. Some may be matters of curiosity, but no urgency at all, in which case the satellite is free to execute maneuvers whenever it is most fuel-efficient to do so. It may also need to maneuver around other space objects (perhaps other satellites that require observation).

This paper proposes an architecture to compute preference-optimal trajectories. A cognitively inspired expert system moderates the trajectory generation and optimization process. At initialization, a solution technique is selected based on problem characteristics. If an initial trajectory estimate is required for the solution technique, one is generated, again with consideration for the problem characteristics. Expressed user preferences are transformed via fuzzy methods into an initial set of weights or other parameters, and the selected solution technique is run.

The expert system also considers the results generated by the chosen algorithm. Often in these problems, one or more user-defined constraints or preferences will not be met after the first iteration. Making changes to the weight vector or to other parameters may solve the problem; so may a different initial trajectory estimate or the use of a different solution technique (e.g., if the problem will not solve using the first technique). Given its knowledge base and the current history of repair attempts for this problem, the expert system continues to search for an appropriate trajectory. Below, related literature is introduced to motivate the posed problem and algorithm choices made. Next, the preference-optimal trajectory generator is introduced at the system architecture and algorithm level. Planar and 6DOF systems are introduced, with results used to evaluate the fuzzy logic and iterative weight-adjustment strategies over a series of simple and complex domain examples.

II. Related Work

Incorporating user preferences into a trajectory-generation feedback loop requires assimilation of representations and techniques across multiple disciplines. Below, we motivate our work through connection to the literature in natural language, motion planning, multi-objective optimization, and fuzzy set theory. Emphasis is placed on defining the technological needs for insertion of preferences into trajectory-generation processes and identification of gaps in existing techniques.

A. Natural Language

Verbal or written instructions are one possible mode of interaction between a human and a robotic vehicle [4]. With respect to motion words, descriptive verbs such as "swagger," "slink," "slide," and "sway" effectively convey the nature of motion, but the specific choice of verb is open to interpretation [5]. Translated to our problem, the speaker has numerous choices to describe how a route from a location A to B is traversed. What verbs the speaker selects indicate to some extent the "manner of motion" required for the robot.

There is no appreciable literature dealing with the transformation of verbs to numbers. There is, however, a literature on assigning numeric values to spatial expressions such as "near" or "in front of" [6–8]. Researchers use, among other techniques, a potential field (first developed for robotic path planning [9]) as a membership function in the fuzzy set theory sense; fuzzy terms like "crisp" and "scruffy" appear frequently. Essentially, one point or line is selected (by the researchers) to represent the ideal of "near," "along," or "in front of" some object in the space. This becomes the minimum for the potential field, which can be visualized as a bowl. The minimum of the potential field "bowl" is located at the ideal value set for "near," "along," etc. This paper extends this idea to motion words. First, we define a "state feature space" composed of state features such as average forward velocity and maximum acceleration. A collection of points in state feature space is taken as the ideal representation of a verb or adverb/verb pair, like "jog" or "move stealthily." Fuzzy membership functions are then defined around these areas, so that similar but not identical kinds of motion can still be included in these classifications. This provides flexibility when trying to satisfy the multiple constraints and maximize the combination of objectives such terms imply, while maintaining the user's preference for motion type.

B. Path and Kinodynamic Planning

Path planning focuses on finding a path through free space from an initial to a final location [10]. The robot's dynamics are not generally considered for holonomic robots. For nonholonomic robots, dynamic constraints that directly affect path, such as a turning radius, are used to reject infeasible paths. When following the path, the robot is typically pre-programmed with a simple trapezoidal velocity profile that ramps up to a constant velocity, then decelerates to the final zero-velocity state. For slow, wheeled robots, especially those in a laboratory or office environment, this model usually suffices to move the robot around. *Behavior-based* motion control [11] is the next step, in which environmental cues trigger pre-programmed responses. The resulting actions can be sophisticated or even appear emotional [12]. Although research has begun to make parameters that define these behaviors adaptive to environmental stimuli, they are still reactive, favoring simplicity and real-time response over optimization [13,14].

While most behavior-based protocols can be represented with a Markov decision process (MDP), another branch of research looks at hybrid dynamic systems [15,16]. In such systems, discrete events trigger shifts between different continuous dynamics. In the cited work, a simulated mouse agent adjusts its trajectory in response to the environment by changing weights. The weights, however, correspond to repelling and attracting potential functions for local navigation, not useful for global or multi-objective optimization. "Programming by reward" is a technique that elicits different dynamic behaviors from a system [17]. Like our research, it uses preference information to create these differences. Unlike our research, it injects the preference information into machine learning algorithms for the development of motion behaviors. These behaviors can form an optimal policy, given preferences. However, the "interiors" of the behaviors are still black boxes. The number of lane changes in the authors' driving example can be optimized for a safe driver and for a reckless driver, but the dynamics of those lane changes are unexamined. Our multi-objective trajectory-optimization application requires control of the low-level inputs that result in the desired behaviors, rather than assembling pre-typed behaviors into a policy.

Trajectory or kinodynamic planning incorporates velocity (dynamics) into planning processes. Methods based on velocity obstacles [18,19] are conceptually similar to path planning "roadmap" methods, with infeasible velocities

modeled as velocity space obstacles. Spline methods [20,21] decouple path and velocity planning; once a clear path through space is found, interpolating splines find smooth trajectories along them. Randomized kinodynamic planning [22] explores the state space in a random fashion, working forward from the start state and backward from the goal state until the search trees meet. All these methods search for dynamically feasible trajectories, but have a single parameter or fixed function to define "optimality." Velocity obstacle and spline methods have been used to generate time-optimal trajectories [23,24] and randomized methods are often used as starting points for linear programming methods to generate fuel-optimal trajectories.

C. Multi-objective Optimization

Trajectory planning is a multi-objective problem with constraints, typically including fuel use, traversal time, and obstacle clearance as often-competing parameters to balance. While different approaches have been developed, they have one common feature: a parameter set that can be adjusted to reflect user priorities. Genetic and evolutionary algorithms (GAs and EAs) have become popular search and optimization tools [25]. A population of potential solutions is generated and encoded, and then evaluated, with the most promising solutions modified or maintained over multiple generations that ultimately yield one or more solutions. Since there is rarely a single point where all objectives are simultaneously maximized or minimized, evolutionary multi-objective optimization (EMO) frequently makes use of *Pareto optimality* [26,27] in which a set of nondominated solution vectors are identified. These Pareto-optimal solutions are defined as the *Pareto front*. Recent GA/EA research has concentrated on ways to encourage population diversity to enable exploration of the entire Pareto front, and the addition of elitism, which stores nondominated individuals so that they will not be lost. Most work to date has studied two or three objectives, for which Pareto-front graphs are examined to select a solution from the nondominated set. Open questions remain for "many-objective optimization." It has been shown that, as the number of objectives increases, Pareto dominance becomes nearly useless in ranking individuals [28]. Also, for deployed applications it is infeasible to evaluate a comprehensive nondominated solution set, and thus incorporating user preference into EAs is an emerging area of study.

There are two additional considerations to note when deliberating the use of EAs for trajectory optimization. First, EAs do not explicitly calculate gradients along the solution set. A good fitness function and the judicious use of crossover and mutation ensures that the solutions will tend to follow the gradient down to the minimum, but this is accomplished by selecting evermore-fit individuals, not by taking advantage of trends. In some cases, this is a strength—EAs are robust to discontinuities in the solution space, particularly prevalent at constraint boundaries. However, when gradients are available, their use typically speeds convergence. Second, EAs are an unconstrained optimization technique. Sometimes, constraints can be recast as objectives. An assortment of penalty functions can be invoked, penalizing the fitness of solutions that do not meet the constraints, although this increases the risk of degenerating into a random walk. Another approach iteratively shrinks the search space to focus on zones where the constraints are met [29]. These approaches work, some faster than others, and most have a set of parameters (such as the "rate of shrink" [29]) that must be set and then tuned by the user.

More deterministic methods have also been employed to develop a Pareto front. Isoperformance [30,31] sets a required performance level, indicated by a fixed value for the cost function. This can be the single output of a complex system, such as the displacement of a space telescope subjected to disturbance forces. Design-variable sets that give the desired performance level are recorded. From one of these sets, a nondominated front is further selected. A user would then select a single solution from among the nondominated solutions. Adaptive weighted-sum methods [32] can be used when the cost function is a weighted sum of terms. Traditional Pareto-front exploration samples weights at constant fixed intervals, potentially missing important front features. The adaptive weighted-sum approach begins with a constant-interval weight mesh, then refines the mesh in areas with large gaps in cost. Inequality constraints are also added to restrict calculations to areas where the Pareto front is believed to exist.

These techniques are promising for multi-objective design optimization (MDO), or more generally in any application where a user wants to obtain an entire nondominated set of solutions for consideration. If we were interested in only planning a route between two specific points, the time required to form the Pareto front of trajectories using one of these techniques might be worthwhile. However, our interest is in calculating many trajectories in similar, but not identical, environments. Since the trajectory-generation process is itself computationally intensive, we avoid computing a full Pareto front for any one set of boundary conditions.

To this end, MILP has become increasingly popular as a relatively fast way to generate and optimize trajectories [33]. Derived from the operations research field, recent increases in computing speed have allowed MILP to be considered for real-time applications. Equality and inequality constraints can all be handled robustly. MILP can approximate nonconvex and logical constraints. Obstacle avoidance is achieved by placing inequality constraints directly on the path space, forcing the trajectory outside the region of the obstacles; penalty functions are not used. The cost functional could, in principle, include many terms, although applications typically focus on either fuel- or time-optimal trajectories. MILP has been applied to aerospace applications such as spacecraft rendezvous and multisatellite reconfiguration [34,35]. However, as its name suggests, it is suitable only for problems with linear constraints, including dynamic constraints. For space-based planning, researchers typically employ simplified models such as the linear Hill's/Clohessy–Wiltshire equations [34] and gravity-free (flat space) dynamic system formulations [35]. MILP does not handle nonlinear constraints, except by linearizing them. This is appropriate for some domains, but not for all. For example, trajectory planning for a satellite (or satellite formation) over a highly elliptical orbit or over more complex gravity fields is not amenable to linearization. More fundamentally, as with EA, MILP does not support user preferences except through cost functions and constraints. As will be shown, although we adopt an optimal control trajectory planner to handle nonlinear dynamic systems, our architecture could also adopt MILP as its trajectory planner upon which our preference-based deliberation structure is placed.

D. Optimal Control Trajectory Planning

Optimal control methods [36,37] have been extensively used to solve complex trajectory planning problems, and are applicable to linear or nonlinear dynamic systems. The calculus of variations is used to frame the problem as a system of differential equations subject to conditions imposed at the initial and final time. Generally, a cost functional is of the form:

$$J = \int_{t_0}^{t_f} g(\boldsymbol{x}(t), \boldsymbol{x}'(t), t)\mathrm{d}t \tag{1}$$

where $\boldsymbol{x}(t)$ is the state vector and $\boldsymbol{x}'(t)$ is its derivative. A variation in the functional, δJ, can be defined for small changes of $g(\boldsymbol{x}(t), \boldsymbol{x}'(t), t)$. If a relative minimum for J exists, it is necessary that δJ be zero at that point, yielding the Euler equation. The problem is then to find an admissible input (or control) vector $\boldsymbol{u}^*(t)$ that causes a system described by the differential equations in Eq. (2) to follow an admissible trajectory $\boldsymbol{x}^*(t)$ that minimizes Eq. (3) cost.

$$\boldsymbol{x}'(t) = \boldsymbol{a}(\boldsymbol{x}(t), \boldsymbol{u}(t), t) \tag{2}$$

$$J(\boldsymbol{u}) = \int_{t_0}^{t_f} g(x, a(\boldsymbol{x}, \boldsymbol{u}, t), t)\mathrm{d}t \tag{3}$$

At all points along an admissible trajectory, Eq. (2) holds and can be rewritten:

$$\boldsymbol{a}(\boldsymbol{x}(t), \boldsymbol{u}(t), t) - \boldsymbol{x}'(t) = 0$$

This dynamic constraint set with Lagrange multipliers $\boldsymbol{\lambda}$ forms an augmented cost functional:

$$J_a(\boldsymbol{u}) = \int_{t_0}^{t_f} g(\boldsymbol{x}(t), \boldsymbol{u}(t), t) + \boldsymbol{\lambda}^T \left[\boldsymbol{a}(\boldsymbol{x}(t), \boldsymbol{u}(t), t) - \boldsymbol{x}'(t)\right] \mathrm{d}t \tag{4}$$

The extremals of the functional are where δJ_a is zero. Finding δJ_a and setting it to zero results in three necessary equations. They are most commonly expressed in terms of the Hamiltonian $\mathcal{H}$, defined as:

$$\mathcal{H}(\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{\lambda}(t), t) = g(\boldsymbol{x}(t), \boldsymbol{u}(t), t) + \boldsymbol{\lambda}^T [\boldsymbol{a}(\boldsymbol{x}(t), \boldsymbol{u}(t), t)] \tag{5}$$

The necessary conditions are then:

$$\boldsymbol{x}^{*'}(t) = \frac{\partial \mathcal{H}^T}{\partial \boldsymbol{\lambda}}(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), \boldsymbol{\lambda}^*(t), t)$$
$$\boldsymbol{\lambda}^{*'}(t) = -\frac{\partial \mathcal{H}^T}{\partial \boldsymbol{x}}(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), \boldsymbol{\lambda}^*(t), t)$$
$$0 = \frac{\partial \mathcal{H}}{\partial \boldsymbol{u}}(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), \boldsymbol{\lambda}^*(t), t) \tag{6}$$

for all $t \in [t_0, t_f]$. For a fixed final time and a fixed final state, we have boundary conditions $\boldsymbol{x}(t_0) = \boldsymbol{x}_0, \boldsymbol{x}(t_f) = \boldsymbol{x}_f$, which enables constants of integration to be determined. Solving the system of equations returns the function (trajectory) that minimizes the cost functional.

If the final time and final state are free, a new boundary condition called the transversality condition is produced:

$$\mathcal{H}(\boldsymbol{x}^*(t_f), \boldsymbol{u}^*(t_f), \boldsymbol{\lambda}(t_f), t_f)\delta t_f - \boldsymbol{\lambda}^T(t_f)\delta \boldsymbol{x_f} = 0 \tag{7}$$

When $\boldsymbol{x_f}$ is fixed and t_f is free, as in this work, $\delta \boldsymbol{x_f} = 0$, so $\mathcal{H}(t_f) = 0$. Except for certain special cases, there is no way to analytically solve the optimal control problem. A variety of numerical methods have been employed, including shooting methods [38] and collocation. The shooting method uses initial value problem (IVP) solutions as a starting point to "shoot" towards the solution of the optimal control boundary value problem (BVP). A stable BVP may require the integration of unstable IVPs (ones highly sensitive to changes in boundary values), a drawback that led to the development of the collocation approach. In collocation, the actual solution to differential equations (6) is approximated over a mesh, defined by "knot points." The approximation is made to satisfy the boundary constraints at t_0 and t_f, and further to satisfy Eq. (6) at each knot point and at the midpoint of each interval between them. An initial guess for the solution must be provided; the solution technique will alter the current solution estimate to bring its residual (a measure of error) to within acceptable bounds.

There are many ways to solve a collocation problem. Solution methods fall in general into two classes: direct and indirect. Direct methods [39] model the approximate solution as composed of basis functions; the solution is improved by altering a vector containing the coefficients for the basis functions. This allows vector optimization algorithms such as sequential quadratic programming, Newton–Gauss, or Levenberg–Marquardt to be applied [40]. Direct methods are considered faster and more robust than the indirect methods. Indirect methods [41] link knot points with continuous approximating functions (e.g., splines) over each subinterval. The coefficients of each of these functions must then be solved. This makes indirect methods more computationally intensive than direct methods. Their advantage is in flexibility; the basis functions in the direct methods must be chosen such that every function could be a feasible trajectory. The indirect method has no such constraint.

The optimal control problem's solution is governed by a single cost functional. Multiple objectives can only be optimized via an aggregation method. Since some constraints are likely to be nonconvex, this means that certain solutions along the Pareto-optimal front may be missed. Typically, a sufficient number of other solutions that also satisfy the user's preferences also exist where they can be detected. Optimal controls problems can incorporate constraints and discontinuities. Equality constraints on the state (such as satisfying system dynamics) are adjoined to the cost functional via Lagrangian multipliers, as discussed above. Constraints on the control inputs can be handled via Pontryagin's Minimum Principle and the resulting switching curves. Inequality constraints can be handled by the introduction of a function of a dummy variable, x_{n+1}, whose derivative is defined as:

$$\boldsymbol{x}'_{n+1}(t) \equiv [f_1(\boldsymbol{x}(t), t)]^2 \Uparrow (-f_1) + [f_2(\boldsymbol{x}(t), t)]^2 \Uparrow (-f_2) + \cdots + [f_l(\boldsymbol{x}(t), t)]^2 \Uparrow (-f_l) \tag{8}$$

where $\Uparrow (-f_i)$ is a unit Heaviside function defined by:

$$\Uparrow (-f_i) = \begin{cases} 0, & f_i(x(t), t) \geqslant 0 \\ 1, & f_i(x(t), t) < 0 \end{cases} \tag{9}$$

for $i = 1, 2, \ldots, l$ (where $l \leqslant m$, the size of the control vector). This derivative is always positive or zero: the f_i terms are squared, and the unit Heaviside function is either 0 or 1. x_{n+1} can then be defined as:

$$\boldsymbol{x}_{n+1}(t) = \int_{t_0}^{t} \dot{\boldsymbol{x}}_{n+1}(t)dt + \boldsymbol{x}_{n+1}(t_0) \tag{10}$$

We require boundary conditions $\boldsymbol{x}_{n+1}(t_0) = 0$ and $\boldsymbol{x}_{n+1}(t_f) = 0$. Since the derivative is never less than zero, $\boldsymbol{x}_{n+1}(t)$ must be zero for all $t-$ if $\boldsymbol{x}_{n+1}(t)$ were to become greater than zero, by its definition there is no way to reduce it back to zero to meet the boundary condition at t_f. This is a constraint of the form $f(\boldsymbol{x}(t), t) = 0$ treated by the method of Lagrange multipliers.

The switching curves and Heaviside function are discontinuous, making them problematic for many numeric solvers. They can be approximated by a series of increasingly steep polynomials. However, the unchanging nature of $\boldsymbol{x}_{n+1}$ presents a further problem. Collocation solvers require gradient information to reduce the error between the current approximate solution and the true solution. Since $\boldsymbol{x}_{n+1}$ is identically zero for the entire trajectory, it provides no gradient information. In our particular case, the collocation solver required a Jacobian matrix which, when the Heaviside approximation was added, contained a full row of zeroes and so would not solve. Adjusting the Heaviside approximation further to provide gradient information in the allowable solution region amounted to instituting a penalty function, which is another way to treat state-inequality constraints.

Penalty functions are often used in path and trajectory planning for obstacle avoidance [9–12,18,19]. Often cubic in form, these penalty functions are centered over an obstacle and monotonically decrease as they move away from its center. Typically, they go to zero at some influence limit away from the obstacle, but this is not required. For our work, penalty functions assume a fixed value at an obstacle's center, at the edge of the object, and at a fixed distance from the edge of the object. These constant values are then connected through smooth cubic functions, the coefficients of which can be varied to achieve these conditions for obstacles of different sizes. With an optimal control approach, the value of the penalty function is added to the cost functional. As cost is minimized, the trajectory will move away from obstacles. However, if other costs are sufficiently great, it may be numerically less expensive to accept the penalty—which means planning a path through the obstacle. Penalty functions do not offer guarantees on constraint satisfaction, which means solutions generated via optimal control methods must be validated for obstacle clearance.

E. Fuzzy Set Theory

Above we reviewed available techniques for trajectory optimization with multiple objectives and nonlinear dynamics. The primary element missing from these methods is the ability to inject user preferences initially and at intermediate points during trajectory-optimization processes. Because a human user will expresses preferences through natural language adverb/verb expressions as discussed above, we require a connection between these words and the mathematical cost and constraint formulations common to all trajectory-optimization protocols. We employ fuzzy set theory to perform this translation. The main idea behind fuzzy set theory is that a member of a set may belong only partly to that set [2]. Classically, individuals either are or are not contained in a set. An individual may be 50% hot and 50% not hot, or 30% hot and 50% warm, for example. Complements, like "hot" and "not hot" must sum to 100% but noncomplementary attributes may not. For example, the vertical line in Fig. 1 indicates the generic feature value F is about 45% "low," about 60% "medium," and 0% "high." The triangles in Fig. 1 are membership functions. They correlate "crisp" numeric values, as measured in the real world, to these fuzzy levels. A fuzzy rules set then acts on these "fuzzified" inputs. For example, "If air temperature is LOW, turn heater fan to "HIGH" and "If air temperature is MEDIUM, turn heater fan to LOW." The fans speeds will have similar fuzzy membership function correlating speeds like "HIGH" and "LOW" to revolutions per minute. These outputs are scaled by the membership function of the inputs.

Natural language, while a desirable input modality, is inherently ambiguous. From interpreting sounds into words to parsing the words into sentences to interpreting the possible shades of meaning of a sentence, there are ambiguities. Classical mathematics does not manage ambiguities well. Fuzzy techniques, on the other hand, deal with them substantially better [3]. Fuzzy optimization applies fuzzy set theory to optimization problems. Fuzzy techniques are not themselves used to solve the problem, but are rather applied to candidate solutions to rank them. They are

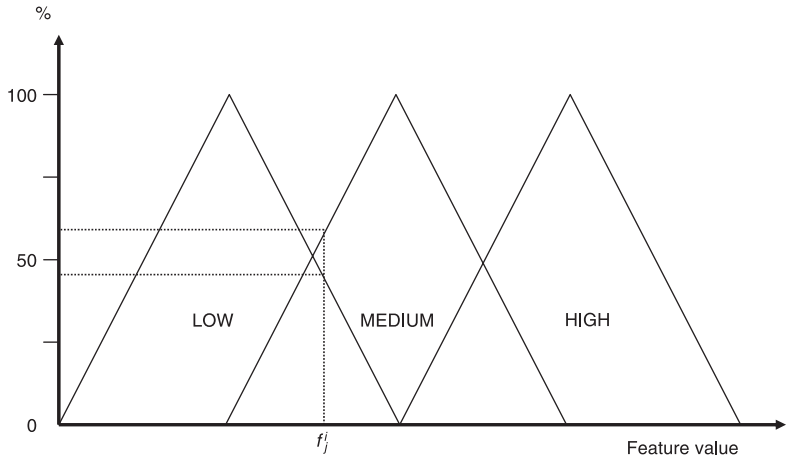

Fig. 1 A fuzzy membership function.

often used in conjunction with EAs, where the EAs generate candidate solutions and the fuzzy methods rank them before selection and breeding occur. Recent work [6] has shown that an expanded and fuzzified notion of Pareto dominance seems to perform more in accord with common sense than strict Pareto dominance, and should not have the same problem as Pareto dominance (e.g., that all solutions become equally good) as the number of objective functions increases to infinity. We combine fuzzy sets with optimal control trajectory planning to gain the benefits of gradient-based numerical optimization while employing fuzzification to translate "common sense" user preferences into numerical objective weights and constraints.

III. Architecture

Figure 2 shows an outline of the agent's processes [42]. At the center sits the evaluation model, overseeing all activities. The human user interacts with this module, monitoring events rather than directly participating in trajectory-generation processes. The evaluation module, *EVAL*, accepts a planning problem, $\mathbf{P}_0$, which can be posed by the user or by any suitable high-level planner that builds task-level actions to achieve its goals, some of which may require vehicle motions.

A trajectory planning problem $\mathbf{P}_0$ is defined as $\{\mathbf{D}, \{\mathbf{O}\}, \mathbf{H}^0, \mathbf{S}^0, \mathbf{bc}\}$. Domain $\mathbf{D}$ describes system dynamics and the parameterized cost functional J to be minimized. $\{\mathbf{O}\}$ represents the set of obstacles in the environment. $\mathbf{H}^0$ describes the hard constraints (limits on state space values) to be met, whereas $\mathbf{S}_0$ is a set of soft constraints that indicate user preference, but are ultimately flexible. $\mathbf{H}^0$ are numeric; $\mathbf{S}^0$ may be numeric or fuzzy linguistic terms. Fuzzy terms must eventually be converted into soft numeric limits; $\mathbf{L}$, the set of all limits, includes $\mathbf{H}^0$ and the extended $\mathbf{S}^0$. Members of $\mathbf{L}$ may be upper limits, lower limits, or range limits (when we want the state feature to be within an upper and a lower limit). The boundary conditions $\mathbf{bc} = \{t_0, \boldsymbol{x}_0, \boldsymbol{x}_f\}$ are split and can include all of the usual optimal controls cases (e.g., fixed or free final time or state, final state constrained to a fixed or moving surface). The goal is to return feasible and optimal solution $\mathbf{X} = \{J^n, \mathbf{L}^n, \boldsymbol{t}^n, \boldsymbol{x}^n, \boldsymbol{u}^n\}$, where J^n and $\mathbf{L}^n$ summarize solution cost and the feature limits/constraints, respectively, of the nth iteration and the set $\{\boldsymbol{t}^n, \boldsymbol{x}^n, \boldsymbol{u}^n\}$ specifies the full-state trajectory (i.e., time sequence $\boldsymbol{t}^n$, position/velocity vector sequence $\boldsymbol{x}^n$, control inputs $\boldsymbol{u}^n$) to be executed. This goal is achieved through intelligent selection of a trajectory planning function and selection and adjustment of a weight vector $\boldsymbol{\Omega}^i$ that influences the relative importance of terms in the cost functional J. *EVAL* incrementally builds a history of activities $\{\mathbf{HIST}\} = \{\mathbf{HIST}^1, \mathbf{HIST}^2, \ldots\}$ with $\mathbf{HIST}^i$ including a record of the function used by *TPLAN*, the initial solution estimate, and the weight vector $\boldsymbol{\Omega}^i$ used for the ith iteration. *EVAL* can then use $\{\mathbf{HIST}\}$ to identify which weight-adjustment strategies it has already employed, to avoid infinite loops. Fig. 3 shows the possible paths through the architecture. Initialization combines the hard and soft constraints and selects initial weights which are expected to meet them [43]. The system loops until all $\mathbf{H}^0$ are met or a loop is detected in the weight vector. If $\mathbf{H}^0$ and $\mathbf{S}^0$ are both met, the successful trajectory is returned. If $\mathbf{H}^0$ are met but $\mathbf{S}^0$ are not met, a second loop is used to

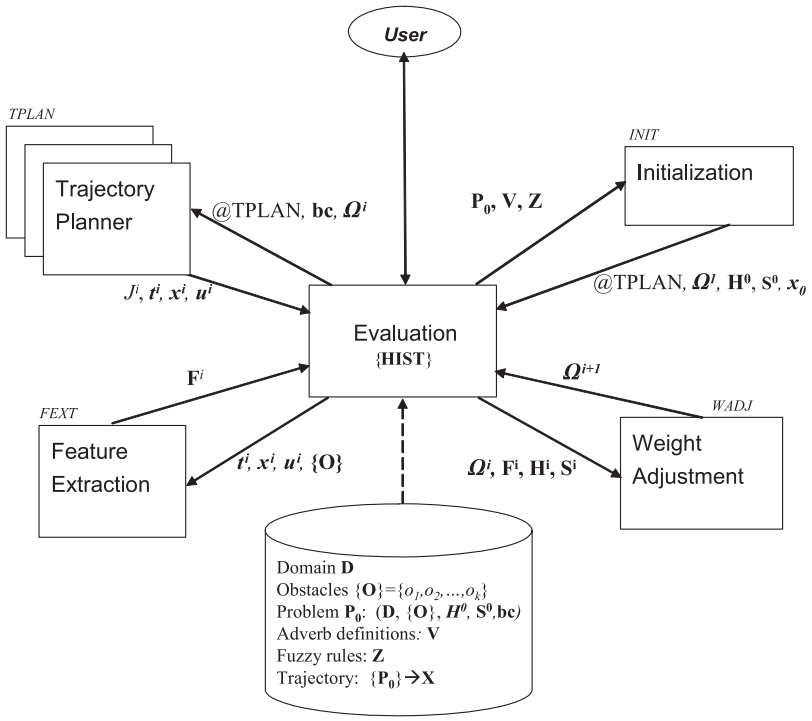

Fig. 2 System architecture.

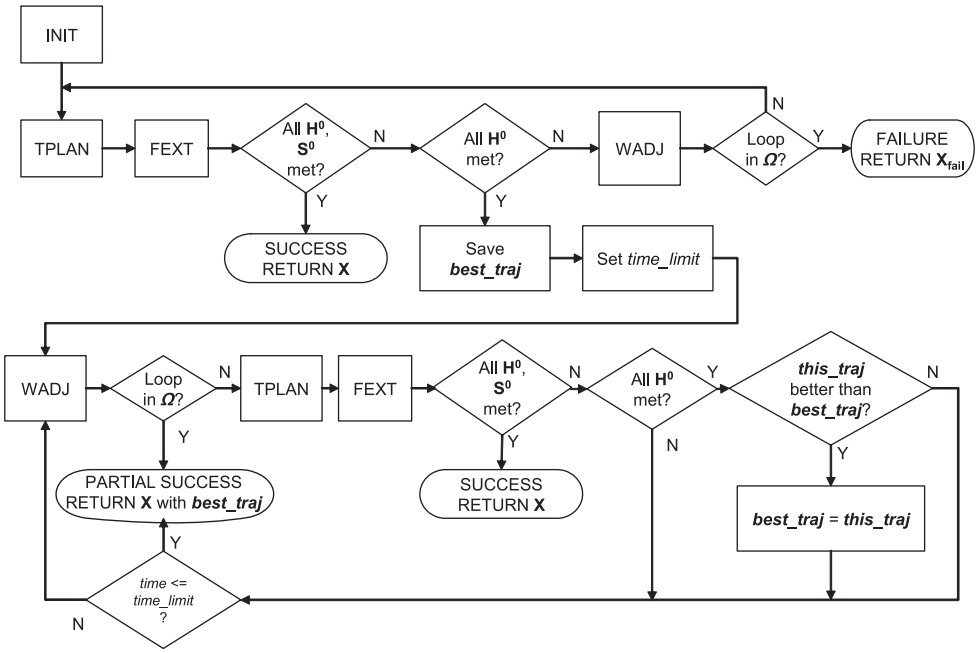

Fig. 3 Paths through the architecture.

attempt meeting $\mathbf{S}^0$. New trajectories are compared to the best found so far, using the 2-norm of the error vector as a measure of quality. In any case, the trajectory which satisfied the hard limits but not the soft limits is stored, so that a legal trajectory is guaranteed to be returned. The process continues as for the hard-limit case, except that if a trajectory is found which again satisfies the hard limits but not the soft ones, this solution is compared to the prior best trajectory. If it is a "better" solution, it replaces the former best trajectory in memory. Currently, the 2-norms of the respective error margin vectors, $\boldsymbol{margin}_{\boldsymbol{error}}$, are compared to determine which solution is "better," and the solution with the smaller overall error margin norm is kept. The error margins are defined as:

$$margin^{i}_{error,\,j} = \begin{cases} F^i_j - \mathbf{L}^0_j, & F^i_j \neg \circ\, \mathbf{L}^0_j \\ 0, & F^i_j \circ \mathbf{L}^0_j \end{cases} \tag{11}$$

where the operator $\circ$ indicates that a feature meets its corresponding limit, whether they are below an upper bound, above a lower bound, or within a range. For the ith trajectory planner iteration, the jth feature is compared to the jth element of the vector of limits. An element of $\boldsymbol{margin}_{\boldsymbol{error}}$ is negative when a lower limit is violated and positive when an upper limit is violated. The sign of $\boldsymbol{margin}_{\boldsymbol{error},\,j}$ is used by *WADJ* to determine the direction of the weight change in subsequent iterations, although the 2-norm ensures positive error margins for trajectory-quality estimates.

We now examine each of the architecture components in more detail.

A. Initialization

To guide the search toward an acceptable solution, the initialization routine *INIT* (Figs. 2 and 3) translates knowledge about the domain $\mathbf{D}$, constraints $\mathbf{L}^0$, and obstacles $\{\mathbf{O}\}$ into choices for the trajectory-generation routine *TPLAN*, any seed information $\boldsymbol{x}_0$ required by *TPLAN*, and an initial weight vector $\boldsymbol{\Omega}^1$.

Almost any trajectory-generation tool set that optimizes over a weighted cost function can be incorporated into the architecture. With multiple tools in place, information for choosing between them must be made available. Figure 4 illustrates choices between a MILP module [35], a receding horizon planner (RHP) [40], and MATLAB's collocation-based boundary value solver BVP4C [41]. User-provided information as well as domain information guides the choices, although making a choice among multiple solvers is beyond the scope of this work, which relies strictly on collocation, a strategy we consider more flexible than MILP to handle nonlinear dynamics and more mature than receding horizon algorithms. Depending on the choice of trajectory planner *TPLAN*, some initial guess may need to be supplied to the trajectory generator (e.g., for collocation). We use a cubic spline which satisfies boundary conditions $\boldsymbol{x}_0$ and $\boldsymbol{x}_f$. In the future, it may be desirable to use a rapidly-expanding random rree (RRT) [22] if the area in which the robot moves is cluttered with obstacles. The RRT can find a dynamically plausible (but nonoptimal) trajectory through the space. This solution can then be used as an initial guess for one of the optimization routines. Once *TPLAN* and any inputs it requires are chosen, the limits $\mathbf{L}^0$ are used to compute the initial weight vector $\boldsymbol{\Omega}^1$ and an extended version of the limits themselves, as described below. When no hard or soft constraints are specified, *INIT* defaults to equal weights ($\boldsymbol{\Omega}^1_{\boldsymbol{default}}$) for all terms of the cost functional.

This computation of $\boldsymbol{\Omega}^1$, shown in Figure 5, accepts limits in either the form of qualitative adverbs (e.g., "quickly" or "safely") with optional adverbial modifiers (e.g., "very") or numeric constraints on a trajectory feature (e.g., "maximum speed $\leqslant$5 m/s"). The numeric constraints may be either hard or soft. We deal firstly with the adverb constraints, all of which are considered soft constraints. Each adverb our system understands is defined in terms of the trajectory features we can extract. For example, "quickly" involves maximum forward speed (*max-speed*) and average forward speed (*avg-speed*). We further define fuzzy levels for each feature. "Quickly" involves "high" *max-speed* and "high" *avg-speed*. These definitions are stored in the fuzzy language database $\mathbf{V}$. The ranges for these values are defined based on vehicle-performance constraints and a typical definition of each adverb relative to these constraints.

We used the same data that generated the weight-adjustment curves to correlate feature levels to weight levels (see Sec. III. D), resulting in our fuzzy rule set $\mathbf{Z}$. We cannot use the weight-adjustment equations themselves, as they require constants that cannot be calculated until the first set of trajectory data is available. But we can say that "low" values of a given weight produce "high" values of *avg-speed*, for example. We define ranges for the weight levels as well. Since all of our fuzzy levels are defined geometrically as triangles with fixed endpoints, their centroids are fixed along the weight axis. Defuzzification for each weight correlated with each feature is the process of looking up

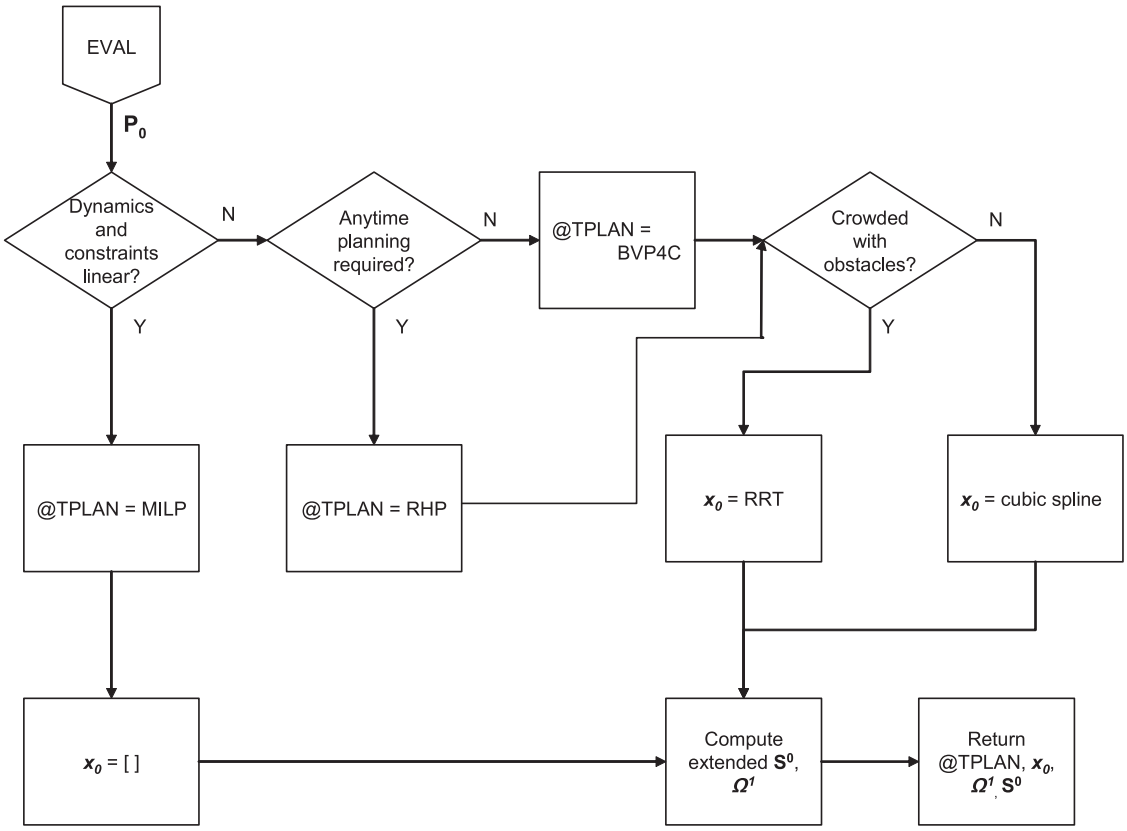

Fig. 4 Initialization procedure.

this centroid value. This numeric information is also contained within the fuzzy rule set $\mathbf{Z}$ and is represented by the generic fuzzy membership function depicted in the upper right of Figure 5.

Numeric soft constraints in $\mathbf{S}^0$ are then also fuzzified (so that "4 m/s $\leqslant$ *max-speed* $\leqslant$ 5 m/s" becomes "*max-speed* high") and appropriate fuzzy weight levels found by referencing $\mathbf{Z}$. This procedure circumvents the necessity for quantitative equations which could directly correlate hard or soft numeric constraints to weight values. We have a general understanding that "*max-speed* high" requires "time weight high," and fuzzy estimates of what values constitute "high" for both parameters. After our first trajectory has been computed, we can estimate adjustments on weights given constraint error margins, but *INIT* must estimate weights to compute the first trajectory from which error margins are extracted. Abstractly, $\mathbf{S}^0$ represents the user's *preferences* for vehicle's behavior. These preferences must be combined into a single weight vector, and then reconciled with $\mathbf{H}^0$, the user's *requirements* or hard constraints on the vehicle's behavior.

$\mathbf{S}^0$ yields weights that are combined into a centroid weight vector $\boldsymbol{\Omega}^1$ that represents an average over each feature's "ideal" weight vector. The "mass" used for this centroid computation is an optional adverbial qualifier that can be stated in the adverb constraint. So "safely but *a little* quickly," where "little" is the adverbial qualifier, places more emphasis on weights resulting from "safely" than weights computed from the "quickly" term. If there are additional hard numeric constraints on features in $\mathbf{H}^0$, we fuzzify those constraints as we did for the numeric soft range constraints. Then we check if the current weight vector $\boldsymbol{\Omega}^1$, when fuzzified via the rules in $\mathbf{Z}$, correlates to that fuzzy level. For example, if we require a hard limit "*max-speed* $\leqslant$ high" and the relevant weight in $\boldsymbol{\Omega}^1$ is "low," are we likely to generate an acceptable behavior? If we are not, the weight value in the required fuzzy range closest to the current weight value is selected. If we require a "high" weight to elicit "high" *max-speed*, but our current weight is "low," the algorithm will select the value in the "high" fuzzy weight triangle closest to the "low" triangle. Since

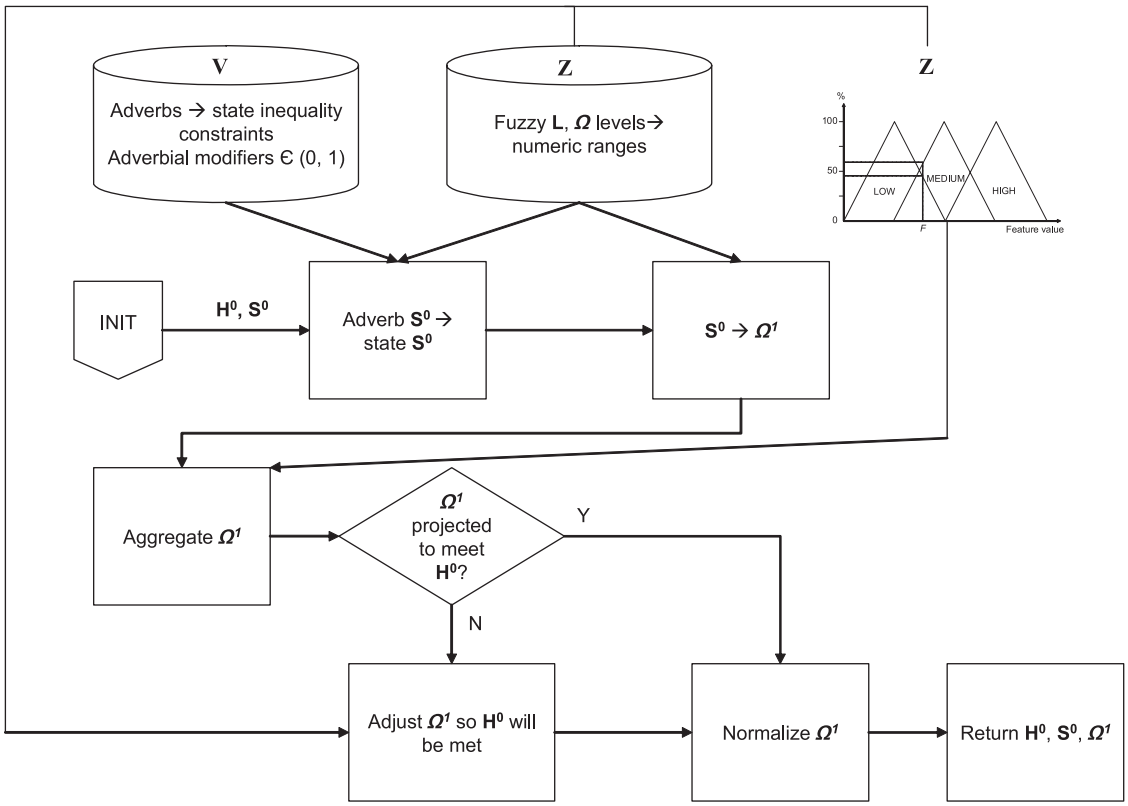

Fig. 5 Calculating Ω^1 and extended S^0.

this overrides the centroid calculation that was built on user preference, it is done only for hard limits $\mathbf{H}^0$, which we assume the user *needs* rather than *wants*.

B. Trajectory Generation

The *TPLAN* trajectory planning module takes as input the INIT weight vector $\boldsymbol{\Omega}^1$ as well as a model of the domain and cost-functional terms. It returns a full state trajectory, including position, velocity, and control inputs at each time step. Users can incorporate their preferred trajectory generator (*TPLAN*) into this architecture. In our experimental domains, we have used the collocation-based BVP4C solver [41] and an extension BVP4C2 [44] for the split BVP. Limited modifications were required for our adaptation. First, although BVP4C and BVP4C2 can theoretically solve for a free end time, to make a problem with free end time converge to a solution requires a very good initial guess, both as to the shape and the duration of the trajectory. We instead provided the solver with an array of possible final times and made the assumption of smoothness between them. The solver iterated over each possible final time in the array and the costs for each resulting trajectory were compared. If the lowest-cost trajectory was in the middle of the array, the times on either side of the lowest-cost trajectory were taken as new upper and lower time limits for a new array with smaller steps between final times. If the lowest-cost trajectory was at either end of the array, the current time step was preserved and the lowest-cost trajectory was used as either the new high or low end of the array. Second, although the collocation routines are fairly robust, they are still sensitive to certain numeric artifacts. We discovered, for instance, poor convergence for certain final times. The algorithm converged well for some $t_f + dt$ and for $t_f - dt$, but for t_f itself, no good answer would be found. Other variables, including continuation schedules (where obstacle potential functions or vehicle step-response outputs are slowly brought from some smooth approximation to their sharper, final shape) and obstacle placement (obstacles symmetric with respect to the initial path guess especially) could also cause problems. Our automated data generation scheme does not easily detect such

cases, not a prohibitive limitation in our experiments, however. In many of these cases, the nonconvergence was not pathological; desired error bounds of 0.1 m might be exceeded by errors of 0.2 m, for instance. Further, these cases were rarely the lowest-cost trajectories, and thus we assumed we could select the lowest-cost convergent trajectory in the presence of nonconvergent trajectories.

C. Feature Extraction

Feature extraction (*FEXT)* is a computational process that takes as an input the generated trajectory and extracts from it certain gestalt properties useful in evaluating the trajectory. Total battery power expended, total time, maximum speed, average speed, and maximum acceleration during the traverse are typical features. Some are maximum or minimum values which are straightforward to express in $\mathbf{L}^0$; others are averages or percentile values that give an overall impression of the trajectory. The "percent plateau" values, for example, are the output of a routine which checks the velocity and acceleration profiles for significant periods of time (at least 10% of the total duration) during which the relevant value fluctuates no more than 1% of its total range. This was intended to give a numerical approximation to the human technique of looking at a trajectory profile and estimating how "flat" it is.

The features can be defined independent of a domain, but may not apply to all domains. An average rotational rate is meaningless in a 2DOF simulation that has only linear motion in the x–y plane. Average vehicle separation is inapplicable to a single vehicle domain. Likewise, certain adverbs or verbs may not be relevant to all domains, even though they exist outside of them. We might prefer an Army field vehicle to move "stealthily," for example, but there is little call to require a space robot to behave in such a fashion.

D. Weight Adjustment

The development of good weight-adjustment (*WADJ*) heuristics was a key part of this work [42,43,45]. Our goal was to automate the process by which cost-functional weights are tuned. This is typically done by hand, by a domain expert, until the desired results are achieved. We have attempted to encode these desired results into the limits $\mathbf{L}^0$, as functions of the features defined above. What remains is to extract domain expert knowledge and techniques and automate the adjustment process. Many of the features in our set can be expressed as functions of the weights used in the cost functional. Despite different dimensionalities, cost functionals, and system dynamics in the 2DOF and 6DOF systems we simulated, we were able to generate *WADJ* heuristics with a common procedure in both cases.

Each test matrix covered a combinatorial set of cost-functional term weights, $\boldsymbol{\Omega}$. Since a cost functional can always be normalized, we examined relative weights rather than absolute magnitudes. Early experimentation led us to conclude that a range of two orders of magnitude in relative weights, from 0.1 to 10, would be sufficient to see a broad range of dynamic behaviors. We varied the magnitude of the commanded motions and also the number of obstacles in the field to ensure answers were not specific to a single problem instance. Once we had collected the trajectories, we used the feature extractor *FEXT* to compute overall trajectory features of interest. For features relating to time (e.g., velocity, acceleration, power), strong power relationships between the feature values and the *ratio* of the energy- or fuel-term weight and the time-term weight (W_1/W_2) emerged:

$$time\_feature = c_1 \left(W_1 / W_2 \right)^{-\alpha} \tag{12}$$

The exponent $-\alpha$ in these equations was approximately constant across field sizes, although the coefficient c_1 varied. Similarly, for path-based features, such as minimum separation from obstacles (*min-sep),* there was a linear relationship between the feature and the influence limit (*LIM*) in the obstacle penalty function:

$$path\_feature = c_2 LIM \tag{13}$$

LIM is simply the distance over which the obstacle penalty function goes to zero, as measured from the edge of the obstacle. Rather than attempt to calculate tables for all possible constant coefficients c_1 and c_2 of these equations for all possible field sizes, they are computed online, using the current weight and feature values to back out the coefficient value. The coefficient, together with the *desired* feature value (e.g., the limit, if it was passed), are then used to recompute the weights. As more obstacles are added to the field, the rules' accuracy is affected. They remain, however, useful rules-of-thumb for guiding weight adjustment, as our results show.

The effects of changing the fuel/time weight ratio or the energy/time weight ratio and *LIM* were largely independent. For example, our data showed little or no change in proximity to obstacles as a function of changing the W_1/W_2 ratio. This allowed us to decouple them, an important and useful assumption. They are not, however, entirely independent. As *LIM* decreases, for example, more direct paths which save both fuel and time can be found. The effect is not dramatic, but can mean the difference between a successful and unsuccessful solution. If the standard *WADJ* rules have failed to find a solution that mediates between competing time and fuel goals, a secondary *WADJ* rule will change *LIM* to take advantage of this secondary effect and find a successful solution.

We were concerned that the 6DOF spacecraft domain with nonlinear dynamics would not be amenable to this *WADJ* rule-generation process. Results for the 6DOF domain were, in fact, very similar to those for the 2DOF domain. A notable difference was the torque weight term, which is unsurprising given the coupled nature of the rotational and translational mechanics. Our torque heuristic is discussed below.

IV. Two Degree-of-Freedom Point Rover

A simplified two-dimensional (2D) domain model was developed as an intuitive baseline case for our architecture and as a method of developing initial modules to populate the Fig. 2 architecture. We began our investigations with a 2DOF point-robot model, imagining a rover-like robot traveling in a plane, using electric motors for propulsion. We used this highly simplified domain to gain an intuition into the process of adjusting the cost-functional weights and computing, then evaluating, the resulting trajectories. The model has simple linear dynamics:

$$\begin{bmatrix} \boldsymbol{x}'(t) \\ \boldsymbol{x}''(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -c_s/m \end{bmatrix} \begin{bmatrix} \boldsymbol{x}(t) \\ \boldsymbol{x}'(t) \end{bmatrix} + \begin{bmatrix} 0 \\ \boldsymbol{u}(t)/m \end{bmatrix} \tag{14}$$

where m is object mass and c_s is the coefficient of sliding friction. We assume an idealized system without motor saturation and perfect trajectory tracking. Our cost functional was an aggregated, weighted sum which penalized electrical energy use, time, and nearness to obstacles. To evaluate the performance of our system for the 2D robot domain, five different logical sets of constraints $\mathbf{L}^0$ of varied complexity were enforced on four different obstacle fields $\{\mathbf{O}\}$ for a total of 20 trials. The field extended from (x, y) coordinates $(-5\,\text{m}, -5\,\text{m})$ to $(5\,\text{m}, 5\,\text{m})$ with zero velocity required at start and end. The obstacles in $\{\mathbf{O}\}$ are denoted as a set containing their center in the (x, y) plane and their radius r, all in meters:

$$\begin{aligned}\{\mathbf{O}\} = \{&\{\{(0.5, -0.5), 1\}, \{-4.0, -2.5\}, 0.5\}, \{(4.0, 3.0), 0.5\}\},\\ &\{\{(1.5, 2.5), 2.0\}, \{(-4.2, -3.5), 0.5\}, \{(-3.2, -4.0), 0.5\}\},\\ &\{\{(1, 1.2), 3\}\},\\ &\{\{(0.0, -3), 0.5\}, \{(4.0, -3.5), 0.5\}, \{(-3.5, 0.5), 1.0\}, \{(-1.0, 0.5), 0.5\}, \{(1.0, 3.7), 0.5\}, \{(4.0, 3.0),\\ &0.5\}, \{(0.5, -0.5), 1.0\}\}\}\end{aligned}$$

We defined 28 possible hard limits ($\mathbf{H}^0$) and 72 soft limits ($\mathbf{S}^0$). The simplest constraint set enforced one hard and two soft constraints; the most extensive (Constraint Set 5) had two hard constraints, two explicit numeric soft constraints, and four soft constraints arising from the fuzzy constraint to move "moderately safely:"

$$\begin{aligned}\mathbf{L}^0 = \{&\{\mathbf{H}^0 = \{\textit{max-speed} \leqslant 4.2\}, \quad \mathbf{S}^0 = \{\text{somewhat quickly}\}\},\\ &\{\mathbf{H}^0 = \{\textit{max-acc} \leqslant 1.0, \textit{min-sep} \geqslant 1.7\}, \mathbf{S}^0 = \{\text{safely}\}\},\\ &\{\mathbf{S}^0 = \{\text{a little quickly, exceedingly inquisitively}\}\},\\ &\{\mathbf{H}^0 = \{\textit{max-acc} \leqslant 1.0, \textit{max-speed} \geqslant 4.0\}, \mathbf{S}^0 = \{10 <= \textit{energy} \leqslant 15, 1.0 \leqslant \textit{avg-speed} \leqslant 2.0\}\},\\ &\{\mathbf{H}^0 = \{\textit{maxacc} \leqslant 1.0, \textit{max-speed} \leqslant 4.0\}, \mathbf{S}^0 = \{10 \leqslant \textit{energy} \leqslant 15, 1.0 \leqslant \textit{avg-speed} \leqslant 2.0,\\ &\qquad \text{moderately safely}\}\}\}\end{aligned}$$

Each of the 20 test cases was run from a default weight vector $\boldsymbol{\Omega}^1_{\textit{default}} = [1,\ 1,\ 1,\ 1]$ and from a $\boldsymbol{\Omega}^1$ provided by *INIT*. The first three of these weights were applied to cost-functional terms minimizing time, energy, and the obstacle

penalty function [43]. The fourth "weight" was the *LIM* used to parameterize the fall-off radius on the obstacle penalty function. Only the collocation *TPLAN* was used for trajectory generation. Illuminating examples from our set of 20 tests are presented in Table 1. Our second $\mathbf{L}^0$ constraint set had $\mathbf{S}^0$ of "safely," which included constraints on *avg-speed*, *max-speed*, maximum acceleration (*max-acc*) and minimum separation between the robot and obstacles (*min-sep*). We added two further numeric constraints, *max-acc* $\leqslant$1.0 and *min-sep* $\geqslant$1.7. We iterated from the base case and from an initial trajectory whose weights were selected by *INIT*. Results for Obstacle Set 1 are summarized in Table 1.

This time, the base case required three additional iterations to converge to an acceptable solution. The base case itself was too fast for "safely," so weights affecting speed and acceleration features were changed. The trajectory resulting from these $\boldsymbol{\Omega}^1$ passed too closely to obstacles, so *LIM* was increased in $\boldsymbol{\Omega}^2$. Forcing the path out and around the obstacles without changing the time-affecting weights resulted in a trajectory that was again too fast, and these weights were adjusted in $\boldsymbol{\Omega}^3$. The trajectory generated using these parameters (Figs. 6 and 7, dashed lines) satisfied all constraints. *INIT*, in this example, provided a starting point that resulted in a final weight set/trajectory much different from those from the base case. Because of the fuzzy level "medium high" required for *min-sep* in "safely," *LIM* starts out twice as large as in the base case. However, the time-affecting weights were still at levels that made it undesirable to plot a path between two obstacles to save time due to the *min-sep* requirements. So the final iteration has weights requiring a longer *and* slower traverse (Figs. 6 and 7, solid lines). The trajectories are quite different given the different weight sets. Both, however, satisfy the required constraints.

The results after one iteration (labeled *Default*^{1} and *INIT*^{1}) and after completion (*Default*^{n} and *INIT*^{n}) were examined for both starting weight vectors. Figure 8 shows the total number of failures for each of the four cases *Default*^{1}, *INIT*^{1}, *Default*^{n}, and *INIT*^{n}. *INIT*^{1} shows a clear advantage over *Default*^{1}, with both fewer failures to meet

Table 1 Summary for 2DOF example

	$\boldsymbol{\Omega}^i$	*max-speed*	*avg-speed*	*max-acc*	*min-sep*
Base case ($\boldsymbol{\Omega}^0$)	1, 1, 1, 2	1.3	1.1	1.3	1.7
$\boldsymbol{\Omega}^1$	1.69, 1, 1, 2	1.0	1.0	1.1	1.6
$\boldsymbol{\Omega}^2$	1.69, 1, 1, 2.13	1.1	0.9	1.1	1.7
$\boldsymbol{\Omega}^3$	***2.04, 1, 1, 2.13***	***0.9***	***0.8***	***0.9***	***1.7***
INIT($\boldsymbol{\Omega}^0$)	14, 3, 1, 4	1.7	0.9	1.3	1.6
$\boldsymbol{\Omega}^1$	***9.15, 1, 1, 4.25***	***0.7***	***0.5***	***1.0***	***1.7***

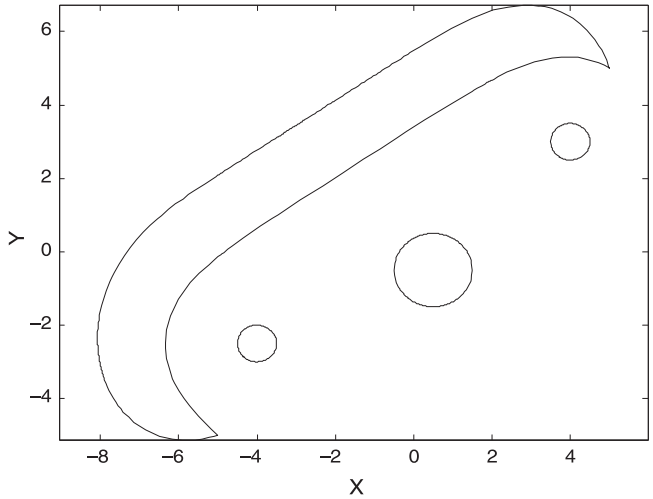

Fig. 6 Paths for 2DOF example.

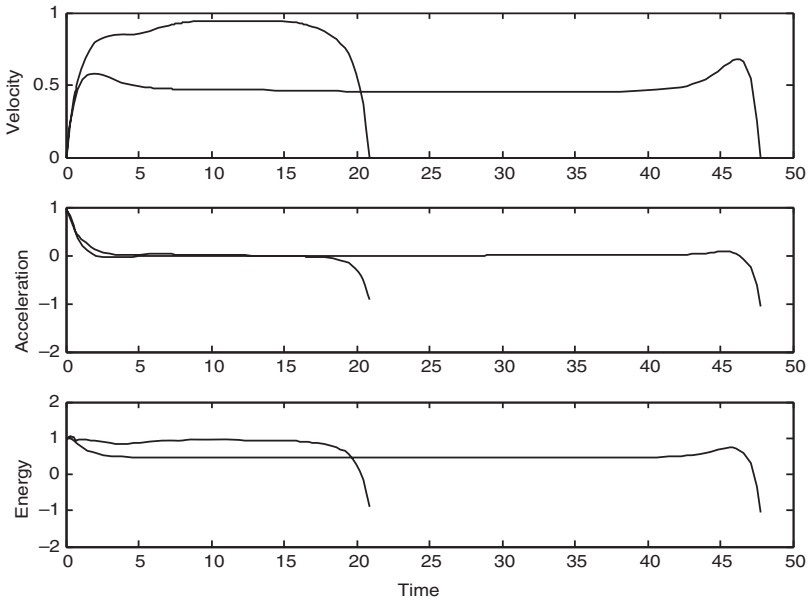

Fig. 7 Resource use for 2DOF example.

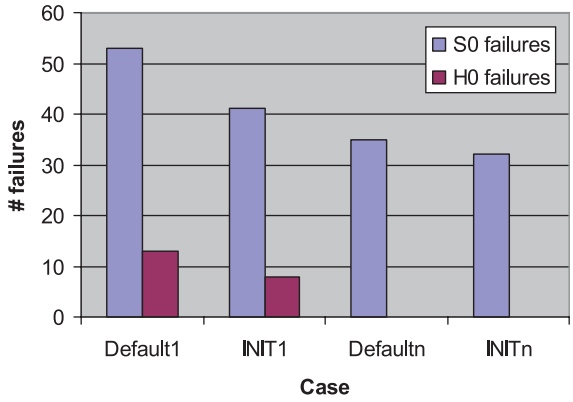

Fig. 8 Failures for each 2DOF case out of 28 $\mathbf{H}^0$ and 72 $\mathbf{S}^0$.

$\mathbf{H}^0$ and $\mathbf{S}^0$. We were not formally working within an anytime planning framework, but this significant improvement in solution quality for the first iteration would be of benefit should we extend the work in that direction.

Final results are not nearly as dramatically different as initial results. Our different starting points in these cases did not, after repeated applications of *WADJ*, result in significant differences in final solution. (We believe that this demonstrates the robustness of our approach.) However, $INIT^n$ converged to an acceptable solution in, on average, 5.25 iterations, whereas $Default^n$ required on average 6.10 iterations, and it never found an acceptable trajectory on the first try. Figure 9 shows a histogram of number of iterations each solution required before returning. Overall, $INIT^n$ has more valid solutions with fewer iterations than $Default^n$. Although the sample is not large enough to analyze statistically, it seems that problems were of two main types: either the process converged quickly, either after just one iteration ($INIT_n$) or after four iterations ($Default_n$), or else it required many more iterations (six for $INIT_n$, seven for $Default_n$) to trade back and forth between competing requirements. The effects of obstacle arrangement did not have a strong affect on the number of $\mathbf{S}^0$ failures, as shown in Fig. 10. Failure rates in $\mathbf{S}^0$ were impacted by the constraint set, as might be expected. Figure 11 gives the failure rates. Constraint Sets 1 and 4 had the overall

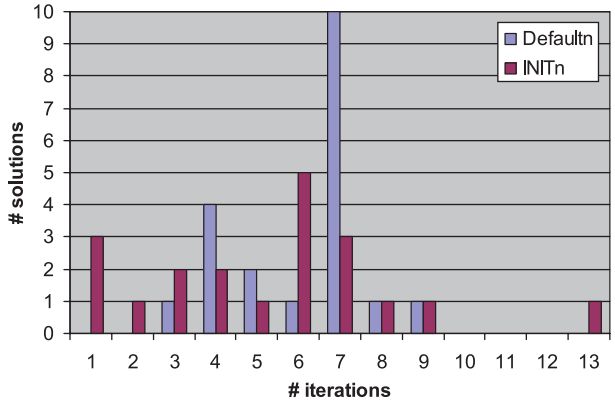

Fig. 9 2DOF iterations through the architecture.

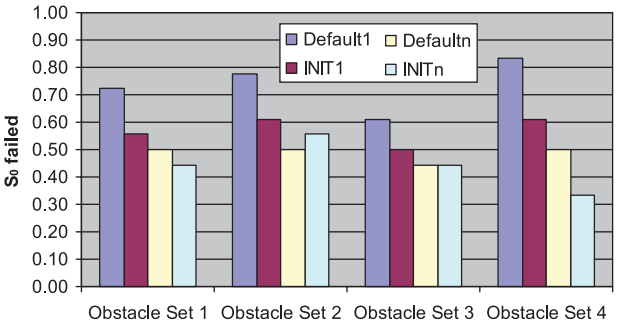

Fig. 10 2DOF S^0 failure rates by obstacle set.

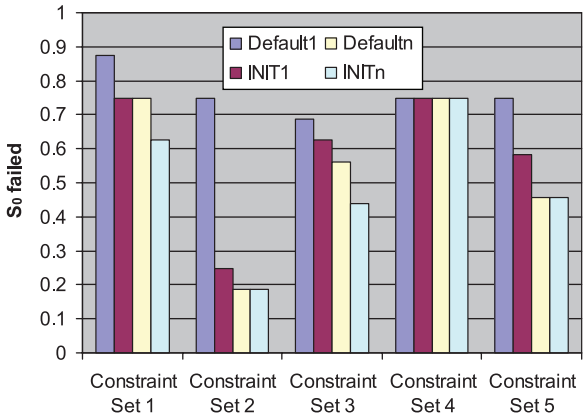

Fig. 11 2DOF S^0 failure rates by constraint set.

highest rates. In Constraint Set 1, competing hard and soft constraints forced the desired *max-speed* into a 0.2 m/s window, a difficult value to achieve.

The Constraint Set 4 failure rates reflect a weakness in the *WADJ* rule-generation method. Constraint Set 4 included two $\mathbf{H}^0$ upper constraints on *max-acc* and *max-speed*, and then $\mathbf{S}^0$ numeric range constraints on *energy* and *avg-speed*. Except for Obstacle Set 4 (the easier obstacle set in Fig. 10), the *energy* constraint was uniformly

violated. Maneuvering around obstacles requires more energy than predicted by the fuzzy rules obtained from the *WADJ* curve data generated in empty space or in a field with one obstacle.

Out of all 40 test cases run to completion, five were able to meet all $\mathbf{H}^0$ and $\mathbf{S}^0$. In matters of solution quality, we found that initialization made little difference when meeting $\mathbf{H}^0$. By the final iteration, the margins by which the $\mathbf{H}^0$ had been met were similar, regardless of whether or not the system had started from initialized weights. The margin by which $\mathbf{S}^0$ of the returned trajectory were made or failed also did not appear to depend on initialization. However, if this architecture were used to support anytime planning, the initialization had clear benefits. The first trajectory returned using initialized weights tended to meet more constraints overall and meet them by better margins. When it failed a constraint, the initialized solution tended to fail it by less than the solution returned by the default weight set.

V. Six Degree of Freedom Deep Space Satellite

The 2DOF experiments showed our architecture could be useful, but the 2DOF point rover problem was highly simplified and linear. As a next-step extension, we adopted a 3D, 6DOF (space) domain example with flat space (gravity-free) dynamic properties that were nonlinear in attitude, but less complex than orbital motion. We assumed a spacecraft with impulsive thrusters for translation and reaction wheels for torque generation, following the modeling described in Henshaw [44]. The general state-space form of a rigid body in deep space is given by:

$$\boldsymbol{x}'(t) = \begin{bmatrix} \boldsymbol{v}' \\ \boldsymbol{p} \\ \boldsymbol{\omega}' \\ \boldsymbol{\sigma}' \end{bmatrix} = \begin{bmatrix} 0 \\ \boldsymbol{v} \\ -\mathrm{H}^{-1}\mathrm{S}(\mathrm{H}\boldsymbol{\omega})\boldsymbol{\omega} \\ \mathcal{G}_\sigma(\boldsymbol{\sigma})\boldsymbol{\omega} \end{bmatrix} + \begin{bmatrix} \boldsymbol{u}(t)/m \\ 0 \\ \mathrm{H}^{-1}\boldsymbol{\tau}(t) \\ 0 \end{bmatrix} \tag{15}$$

where $\boldsymbol{p}$ is the spacecraft position vector in the inertial reference frame, $\boldsymbol{v}$ is its inertial velocity vector, $\boldsymbol{\omega}$ is the angular velocity vector expressed in the body frame, and $\boldsymbol{\sigma}$ is a representation of attitude (a modified Rodrigues vector [46]). H is a matrix of moments of inertia, S is the matrix representation of the cross product, $\boldsymbol{u}$ is the translational control input (force) vector in the body frame, m is the spacecraft mass (presumed constant over each trajectory planning problem), $\boldsymbol{\tau}$ is the vector of rotational control input (torque), and $\mathcal{G}_\sigma$ is an expression which, when multiplied by ω, gives the rate of change in σ:

$$\dot{\sigma} = \mathcal{G}_\sigma(\sigma)\omega = \frac{1}{2}\left(\mathrm{I} - \mathrm{S}(\sigma) + \sigma\sigma^T - \frac{1+\sigma^T\sigma}{2}\mathrm{I}\right)\omega \tag{16}$$

The mass used was 1 kg and $I_{xx} = I_{yy} = I_{zz} = 1$ N-m/s^2, simulating a "microsatellite" rather than a full-scale platform. Maximum thruster output was $\pm$30,000 N in each axis. Maximum torque output about each axis was smooth until a saturation value of $\pm$1000 N-m. We have the same concerns for 6DOF as for 2DOF: fuel and power must be conserved, goals must be accomplished in a timely fashion, and obstacles must be avoided. Our inputs are different in this case. Rather than a continuous electrically powered motor, we have saturating thrusters for 3DOF translation and an electrically powered reaction wheel for 3DOF attitude control. As a result, the cost functional has the form:

$$J = \int_{t_o}^{t_f} \left(W_1 \left\|\boldsymbol{u}(t)\right\|_1 + \boldsymbol{\tau}(t)^T W_2 \boldsymbol{\tau}(t) + W_3 + W_4 \max_{i\in\{O\}}(o_i(r_i))\right) \mathrm{d}t \tag{17}$$

The first term, the one-norm of the thruster force, results in a minimum-fuel control law. This control law is, however, discontinuous (which violates the assumptions of our numeric solver) and thus is approximated with steep but continuous functions, such as a cubic spline made to increasingly approximate a step function [44]. The second term represents electrically powered rotational actuators. This form of the cost functional is an energy-minimizing term, a standard cost representation for electrically powered systems. The third term minimizes time over the integral. The fourth term is the obstacle penalty function. As in the 2DOF case, the obstacle penalty function contains a cubic spline term that penalizes proximity to obstacles. This function also includes a velocity-based component that penalizes speed near the obstacle. Since the cost functional is an integral over time, a penalty based purely on clearance to the obstacle can be minimized by being very close to the obstacle but going very fast, so that the sum over time is less—not a vehicle behavior we would typically wish to encourage.

A. Development of Weight-Adjustment Heuristics and Fuzzy Rules

For the 6DOF weights, the *TPLAN* code BVP4C2 assumed that the force weight W_1 was normalized to one, and all other weights were relative to this. As a result, it was more intuitive to work with the W_1 as the denominator in the weight ratios for our 6DOF spacecraft domain. All of the code written to implement these 6DOF heuristics uses torque/force and time/force weights, rather than the inverse as in the 2DOF case. A subset of the *WADJ* heuristics and fuzzy rules reflects this inversion and the heuristics are labeled W_2/W_1 and W_3/W_1 as implemented. Our weight vector $\boldsymbol{\Omega}$ included W_1 the force weight, W_2 the torque weight, W_3 the time weight, and *LIM*. Since the obstacle penalty function weight W_4 is never adjusted, we do not include it in our weight vector representation.

To develop *WADJ* rules, we followed the general procedure outlined in Sec. again in 6DOF. We did not test different field sizes for 6DOF, however, as we were confident from the 2DOF case that results would scale well. (This confidence was well-placed; our *WADJ* curves were generated at a scale of 50 m while our test cases were on the order of 10 to 20 m.) We tested pure translation, translation plus rotation, and a translation in the presence of obstacles. For the translation in the presence of obstacles, the final state orientation was identical to the initial state orientation; rotation was not required, but it was not forbidden, either. Following the insights gained in the 2DOF domain, we plotted "per second" features (speeds and accelerations) versus the ratio of the weight of the translational inputs (here, thruster force) W_1 to the time weight W_3. The results for the feature *avg-speed* are shown in Fig. 12. Once again, there is the power relationship between speed and the force/time weight ratio. We found this to be the case for the other force- and time-based quantities as well. The path-based features (e.g., *min-sep*) were once again linear with the influence limit of the obstacle penalty function. Unlike the 2DOF case, the trajectories were much more likely to be plotted through obstacles. To handle this, we added an implicit hard constraint to every trajectory, *min-sep* >0 m. If the path went inside an obstacle, the trajectory failed and *LIM* was adjusted to move the path out of the obstacle. This solved that problem.

Torque presented us with a challenge. Our *WADJ* test for torque included a translation and a rotation, so that we would see the effects of dynamic coupling. Following the intuition we had from the translational features, we tried plotting the ratio of the torque weight W_2 versus the time weight W_3. However, since torque and the resulting rotational motions are the source of nonlinearity in the system, initial results indicated no power law for *WADJ*, thus at first we were concerned this heuristic may not be applicable. Upon further examination, however, we identified a more promising heuristic. Figure 13 shows the torque data grouped by time and force. First, the data were grouped by their torque/force weight ratio (W_2/W_1), but plotted versus the time/force ratio (W_3/W_1) as shown in Fig. 13. Each line in Fig. 13 represents a fixed W_2/W_1 ratio. Even though that ratio is fixed, the amount of torque applied can

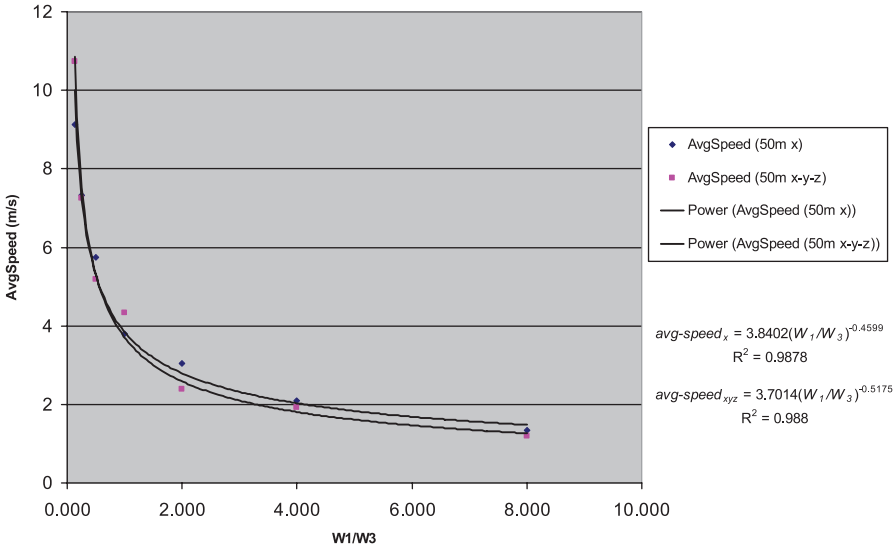

Fig. 12 *WADJ* curve for *avg-speed* in 6DOF domain.

be increased or decreased by adjusting the W_3/W_1 ratio. Conversely, if the W_3/W_1 were known and fixed, changing the W_2/W_1 ratio could jump the torque up or down that family of linear curves. Was there a predictable relationship between the slopes of the lines in Fig. 13 and W_2/W_1? Figure 14 shows that there was. Our torque heuristic was implemented as follows: first, all nontorque features were checked for limit failures and, if there were failures, the weights were adjusted. Then the torque feature was checked. If it failed, the desired torque value was divided by the current W_3/W_1 value to get the slope of the line we would like to be traversed in Fig. 13. Then the power relationship shown in Fig. 14 was used to calculate the necessary W_2/W_1 ratio from the desired slope. The fuzzy rules were generated as they had been for the 2DOF case. The *WADJ* data were reverse engineered so that "very high," "high," etc., weight values were correlated back to the trajectory features they elicited. We again found that the range from $2^{-3}, 2^{-2}, \ldots, 2^2, 2^3$ was sufficient to describe the *WADJ* relationship.

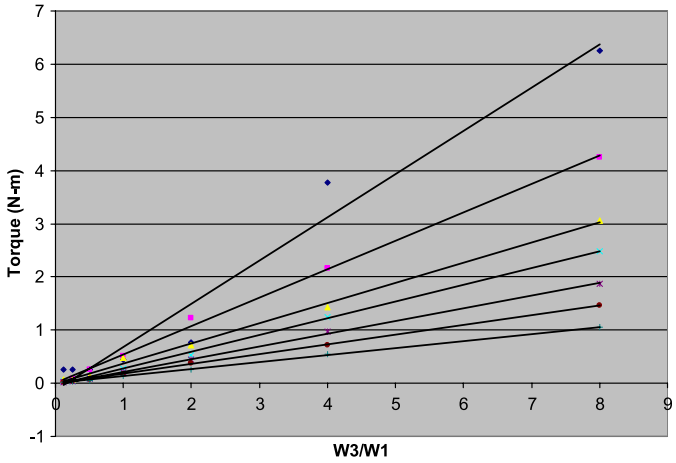

Fig. 13 First-stage *WADJ* heuristic to determine torque in 6DOF.

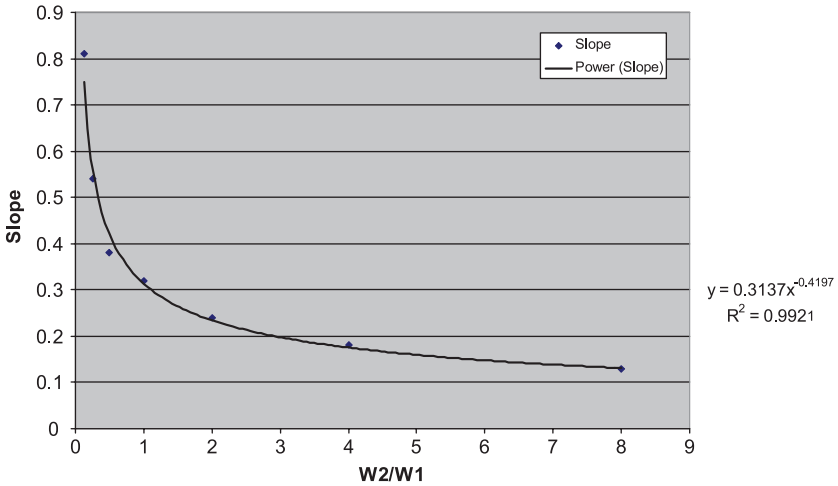

Fig. 14 Second stage in torque heuristic in 6DOF.

B. Results

Six different sets of constraints $\mathbf{L}^0$ were enforced on four different obstacle fields $\{\mathbf{O}\}$ for a total of 24 trials. Spherical obstacles are denoted as a set containing their center in space (x, y, z) and their radius r, all in meters:

$$\begin{aligned}\{\mathbf{O}\} = &\{\{\{(6, 6, 0), 1\}\},\\ &\{\{(3, 2, 3), 0.5\}, \{(8, 5, 8), 0.5\}, \{(14, 12, 12), 0.5\}\}\\ &\{\{(6, 6, 12), 0.5\}, \{(11, 11, 22\}, 0.5\}, \{(16, 16, 32), 0.5\}\},\\ &\{\{(20.38, 25, 25), 1\}, \{(27.31, 21, 25), 1\}, \{(27.31, 29, 25), 1\}\}\}\end{aligned}$$

However, one constraint/obstacle pairing proved to be intractable and BVP4C2 could not converge on a solution. This trial (Constraint Set 4, Obstacle Set 4) is omitted from the following results. There were, overall, 20 hard limits ($\mathbf{H}^0$) and 60 soft limits ($\mathbf{S}^0$). The simplest constraint set enforced one soft constraint; the most extensive (Constraint Set 5) had two hard constraints, two explicit numeric soft constraints, and four soft constraints arising from the fuzzy constraint to move "moderately safely":

$$\begin{aligned}\mathbf{L}^0 = &\{\{\mathbf{H}^0 = \{\textit{max-speed} <= 5.5\,\text{m/s}\}, \mathbf{S}^0 = \{\text{somewhat quickly}\}\},\\ &\{\mathbf{S}^0 = \{\text{exceedingly efficiently}\}\},\\ &\{\mathbf{S}^0 = \{\text{a little quickly}\}\},\\ &\{\mathbf{H}^0 = \{\textit{max-acc} <= 1.3, \textit{max-speed} <= 4\}, \mathbf{S}^0 = \{2.7 <= \textit{force} <= 5.4, 1.8 <= \textit{avg-speed} <= 4\}\},\\ &\{\mathbf{H}^0 = \{\textit{max-acc} <= 1.3, \textit{max-speed} <= 4\}, \mathbf{S}^0 = \{2.7 <= \textit{force} <= 5.4, 1.8 <= \textit{avg-speed} <= 4,\\ &\quad \text{moderately safely}\}\}\}\end{aligned}$$

Each of the 23 successful test cases was run from a default weight vector $\boldsymbol{\Omega}^1_{\textit{default}} = [1,\ 1,\ 1,\ 1]$ and from a $\boldsymbol{\Omega}^1$ provided by *INIT*. Again, only the collocation *TPLAN* BVP4C2 was used for trajectory generation. The results after one iteration (labeled $\textit{Default}^1$ and $\textit{INIT}^1$) and after program completion ($\textit{Default}^n$ and $\textit{INIT}^n$) were examined for both starting weight vectors. Figure 15 shows the total number of failures for each test case. As in the 2DOF case, these are $\textit{INIT}^1$, the solution generated using the weights suggested by *INIT*, $\textit{INIT}^n$, the solution generated by running the $\textit{INIT}^1$ solution to conclusion through the architecture, $\textit{Default}^1$, the solution generated using a default weight vector with all weights equal and $\textit{LIM} = 1$, and $\textit{Default}^n$, the $\textit{Default}^1$ solution run to completion.

The results for soft-limit failures $\mathbf{S}^0$ are similar to the 2DOF case. The *INIT* procedure results in noticeably fewer soft-constraint failures after only one iteration. There are more hard-limit $\mathbf{H}^0$ failures with *INIT* due to our examples with competing constraints, but again no hard-limit failures were present in the final solutions. Margins of success

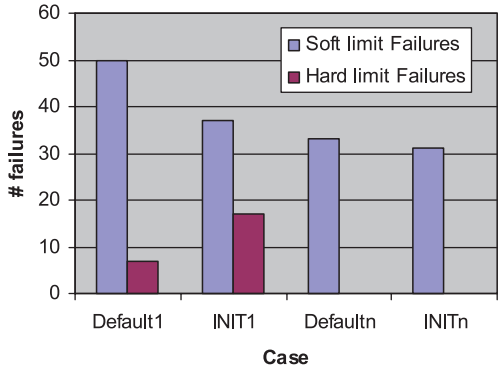

Fig. 15 Failures for each case out of 20 $\mathbf{H}^0$ and 60 $\mathbf{S}^0$.

and of failure were largely comparable between the methods, and similar average numbers of iterations through the architecture were needed to complete the trajectory. However, using INIT did have one advantage, as shown in Fig. 16: a majority of the runs starting with *INIT* finish in one or two runs, while those starting from default weights need a minimum of three runs. Analyzing the $\mathbf{S}^0$ failures by obstacle set shows trends similar to the 2DOF case, although the median failure rate is higher. In the 2DOF case, the median $\mathbf{S}^0$ failure rate was $50\% \pm 6\%$ and there was no clear relationship to the obstacle set. For this 6DOF case, the median $\mathbf{S}^0$ failure rate was 65% with a spread of 12%—greater variability, but no clear trends by obstacle set.

Fig. 17 shows the percentage of $\mathbf{S}^0$ failures by constraint set. Here we see definite trends, with some constraint sets being apparently simple to entirely satisfy, while others had 100% $\mathbf{S}^0$ failure rates. Constraint Sets 2 and 3 had small numbers of noncompeting soft constraints and no hard constraints. Constraint Set 2 had a soft numeric range limit on thrust; Constraint Set 3 was "a little quickly," which defuzzified into soft constraints on *max-speed* and *avg-speed*. With no other requirements, these constraints were solved much more successfully. $INIT^n$ solved them entirely for all obstacle cases; $Default^n$ had small errors with Obstacle Set 1 and Obstacle Set 4. Constraint Set 1 was very similar to Constraint Set 1 in the 2DOF case; we set a hard limit on *max-speed* and also the soft preference for "somewhat quickly." The hard limit was toward the low end of the fuzzy ranges that define "quickly," forcing the system to hit a small window of feature values that would satisfy both. While this constraint set gave the 2DOF case some problems, here it was entirely successfully solved in all cases that went to completion.

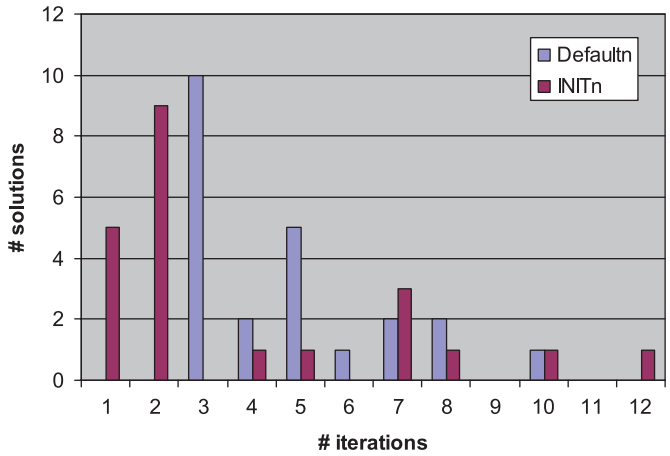

Fig. 16 Number of iterations through the architecture.

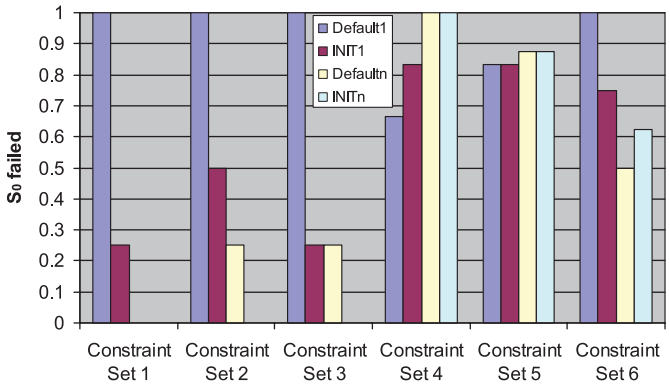

Fig. 17 6DOF $\mathbf{S}^0$ failures by constraint set.

We also note that for these first three constraint sets, $INIT^1$ returns remarkably better initial solutions than $Default^1$. By happenstance, the default weights produce results that do not meet any of the $\mathbf{S}^0$, while $INIT^1$ achieves at least partial success in that. So we see here one practical application for *INIT*: in those cases where there are few or noncompeting constraints, it provides an excellent initial guess compared to using default weights. Constraint Sets 4 and 5 were clearly less successful. Constraint Set 4 included two hard upper limits on *max-acc* and *max-speed* and two soft-range limits on *force* and *avg-speed*. It appears that when the suggested weights for the *force* and *avg-speed* ranges were combined, the force terms were much more sensitive to the change away from their own desired values. Further, the hard limit on *max-acc* was greatly exceeded in all initial cases. By the end of the iterations, the hard limits were all met, but *force* was failed in all cases: failed under the lower limit of the range. By requiring such a low *max-acc*, we were required to use less thrust than specified by the soft range. Similarly, all final *avg-speeds* failed low, as the trajectory had to go slowly enough to meet the upper limit on *max-speed* (a hard limit). Essentially, the stated soft constraints in Constraint Set 4 *had* to fail for the hard constraints to be met. Since this was the case, there was almost no way for the system to succeed, whether or not the weights had been initialized. So we see no difference in the failure rates for the $INIT_n$ and $Default_n$ cases. Constraint Set 6 contained the most constraints, several of which were in competition. Here, trying to compute weights that satisfy all constraints actually decreases result quality relative to the solution generated from the default weight vector. This appears to be a case in which highly constrained optimization is extremely sensitive to initial conditions. It suggests that, when there are more than 2–4 constraints on a problem, we may need to "seed" more than one initial solution to arrive at the best answer.

Constraint Set 5 added the soft fuzzy preference "moderately safely" to Constraint Set 4. This defuzzified into four more soft-range constraints. We were practically guaranteed a certain failure rate, since the upper limit of "safely's" *avg-speed* constraint equaled the lower limit of the soft-range constraint on *avg-speed* from Constraint Set 4. Of course, if we failed low on *avg-speed*, as we did for all of the Constraint Set 4 cases, we would be making the "safely" constraint, decreasing our overall failure rate. Soft constraints on *max-acc* and *max-speed* arising from "safely" were sometimes met when the hard constraints were met, again decreasing the failure rate. (And when they were failed, they failed low as in Constraint Set 4.) We saw very large failure margins which were greatly reduced by the end of the iterations.

Constraint Set 6 was "very energy-saving," which decomposed into soft-range constraints on *force* and *torque*. But since the only torque needed in the trajectories was that required to avoid obstacles, the trajectories all failed low; they could not use enough torque to satisfy the "low *torque*" constraint. "Low *force*" was more typically made, or failed high with very small margins (0.05, 0.02 N) for the completed cases ($Default_n$, $INIT_n$). Since the nonlinearity of the system is in its rotational dynamics, and since we wish to show that our architecture will work with nonlinear systems, we decided to rewrite the $\boldsymbol{x}_f$ to include explicit rotational changes, and rerun it over the set of obstacles under Constraint Set 6. The results are presented separately. For many of the runs, a solution was returned after between one and three iterations (Fig. 16). For one or two iterations, the weights are converged upon with no overshoot. For those default weight cases that took three iterations, some converged monotonically to the correct weights, while others overshot on the second iteration and corrected with the third.

C. Torque

In our original data set, only one of eight completed trajectories (4 $Default^n$, 4 $INIT^n$) met the "low *torque*" requirement. The rest were too low to be considered "low" by the standards of our fuzzy rule set. Since none of our goal states required a rotation, the only rotations required were those needed to avoid obstacles. These did not use sufficient torque to be considered low. So, to test the *torque WADJ* rules, we included a rotation change in each axis at the goal state and reran the tests. We also had some concerns about the possible interaction of *force* and *torque*. In *WADJ*, all features except *torque* are first checked for adjustment. Then the selected W_3/W_1 ratio is used together with the desired *torque* value to calculate a slope from Fig. 13; that slope is then used to pick a W_2/W_1 ratio via the equation in Fig. 14. The "low *force*" requirement was keeping W_3/W_1 small, and the heuristic is less well-conditioned for W_3/W_1 less than 1. Although meeting mixed constraints is an important goal, we also wanted to isolate the torque response to the *WADJ* process, since it is so different from the other *WADJ* heuristics. So we created additional test cases: Constraint Set 7, "low *torque*," and Constraint Set 8, "medium *torque*."

Figure 18 shows our overall failure rates for these three constraint sets (Constraint Sets 7 and 8, and the revised Constraint Set 6); each case had 16 runs (four soft constraints run over four obstacle sets). The $Default^1$ and $INIT^1$

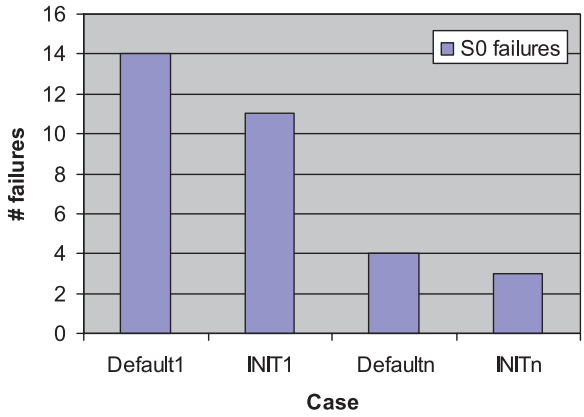

Fig. 18 6DOF S^0 failures for Constraint Sets 6, 7, and 8.

cases are high again, not unexpectedly, and the failure rates for the completed runs are much lower. All seven failures at run completion were torque failing low; of those seven, four were from the "medium torque" Constraint Set 8. W_2/W_1 was continually adjusted down to discount it, to allow for greater torque in these cases, but what was required was that W_3/W_1 be increased. Since there was a tacit assumption that some other state feature would be relying on W_3/W_1 and that it may have been adjusted to affect that other feature, the torque *WADJ* never altered W_3/W_1, and W_2/W_1 could not be adjusted sufficiently before *time_limit* was reached. Our current *TPLAN* cannot handle direct maximization of trajectory qualities; it can only minimize. We have found that we can minimize features which are inversely related to our feature of interest for a maximizing effect; thus by penalizing time, we can usually force an increase in speed. Another *TPLAN* might allow for direct maximization of features.

Figure 19 shows the number of iterations required for these runs. All of those runs which took four or fewer iterations to return a solution returned a complete success. The utility of *INIT* is again shown in the large number of runs that returned successful trajectories after only one or two iterations; eight (two-thirds of the total) of the trajectories created using *INIT* were solved in two or fewer iterations, while only three trajectories created using default weights met this standard.

We have demonstrated that *WADJ* heuristics can be developed for a deep-space 6DOF domain with nonlinear dynamics. Our results were, if anything, better in the 6DOF domain than in our 2DOF domain, with smaller $\mathbf{S}^0$ failure margins and larger success margins for $\mathbf{H}^0$. The average number of iterations required to find a solution was commensurate with the 2DOF case, also arguing that the technique implemented in the architecture will scale well

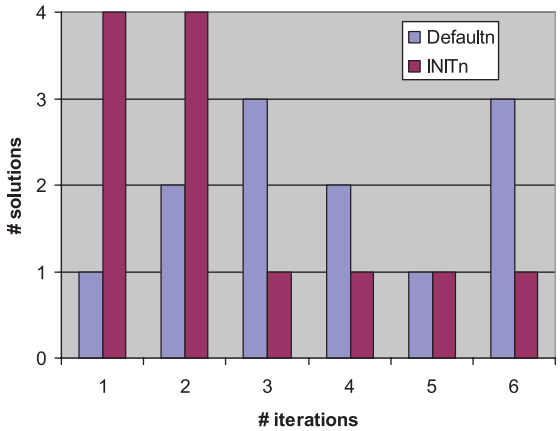

Fig. 19 Iterations required for Constraint Sets 6, 7.

with the dynamic complexity of the domain. The surprising similarity of the 2DOF and 6DOF *WADJ* curves, even to the values of the coefficients, is noteworthy. That also argues for the potential for a general application to the optimization of dynamic systems.

The performance of our *TPLAN*, BVP4C2, was less robust and slower than we had hoped. Prior work on this algorithm required extensive hand-tuning of several sets of gains just to solve a single trajectory problem. We were running it, on average, 5.2 times per problem for 24 fairly different problems. So these difficulties are not entirely unexpected. In the future, however, a different *TPLAN* should be selected for work with nonlinear systems.

VI. Conclusions

We have presented an architecture for optimizing trajectories over user preferences as well as hard constraints. Through insertion of a "feedback loop" to iterate over weighting factors, our approach extends traditional optimization methods requiring a statically weighted sum of all objectives. Through convergence to a single solution that meets constraints and is consistent with preferences, our approach is more computationally efficient than multi-objective methods that must compute the multidimensional Pareto frontier, and then extract an "acceptable" solution from the nondominated solution set. The presented 2DOF and 6DOF examples demonstrate our architecture can efficiently identify trajectories that satisfy hard constraints and natural language preferences. The *WADJ* heuristics consistently direct the weights toward values that meet hard and soft constraints and are robust to differences in initial weight sets. The fuzzy logic enables a more natural human interface, opening a route to easy tasking of autonomous agents by nonexpert users (e.g., hospital staff commanding a robotic assistant, war fighters with a Future Combat System robot, the elderly using a companion robot). However, the ability to meet hard numeric constraints is not lost in adding the fuzziness. This allows the system to be used as an "automated graduate student," overseeing trajectory generation, rejecting those which do not meet required hard constraints, and making intelligent adjustments to the weights to move the solution in the required direction. Substantial knowledge engineering and preprocessing was required to develop the fuzzy rules, the *WADJ* rules, and the *TPLAN* implementation. But once this offline process had been completed, the system was applicable to a wide range of obstacle and constraint conditions with no further adjustments. This makes the architecture useful for robots operating long-term in a consistent environment, but not so useful for "one-off" operations such as technology demonstrations.

VII. Discussion and Future Work

In the future, a more sophisticated version of *INIT* could look at the constraint set as a whole and recognize potential conflicts. Currently, the system will execute many iterations to satisfy conflicting constraints. An early detection of this kind of possible conflict, or else a software monitor that detects a pattern of cycling back and forth, would both be useful to have. *INIT* should also be invariant to the order in which constraints are processed. In this implementation, the order in which the hard constraints are considered impacts the returned weight vector. After the soft constraints have been aggregated via a centroid computation, *INIT* cycles through the hard constraints and checks to see if the currently suggested weights are liable to meet them. If they are not, *INIT* adjusts the weights up or down as needed. If competing constraints are being considered, the last one addressed by *INIT* will be favored, rather than a median weight which might satisfy both.

The *WADJ* process could also be rendered more sophisticated. After the *INIT* cycle, the "adverbial modifiers" like "very" or "somewhat" are lost in the weight-adjustment process. The endpoints of the fuzzy regions for the soft constraints are fixed, without regard to the strength of the user's preference. Nor are they currently considered when deciding which of several competing soft constraints must fail. The assumption has been that the *INIT* process would put the solution in approximately the correct region in weight space, and further iterations would reflect that. That assumption does not necessarily hold, as the *WADJ* rules can cause oscillations of initially very large, then decreasing, magnitude in weight space. Something that preserves the knowledge of soft preference strength past the *INIT* phase would help this adhere more closely to true user preference, and perhaps reduce total iterations needed.

A more sophisticated notion of error margins in *FEXT* might also be of use here. A *WADJ* algorithm that seeks to minimize the entire vector of errors, rather than each error individually, would be computationally more expensive (an optimization within an optimization), but could yield superior results with fewer iterations. Other forms of *WADJ* specific to other cost functionals could be explored. A cost functional based on a linear quadratic regulator (LQR), in which components of the state vector like the velocities are directly penalized, could replace the time component

of the cost functionals used here. Of course, these new terms would still have weighting terms and the relationships between them would have to be investigated, following the procedures outlined here.

We would like to augment *EVAL* with an understanding of the adverbial modifiers, as mentioned above, so that preferences the user described as weaker would be violated in favor of meeting more strongly held preferences. Additionally, some mechanism whereby the original set of limits $\mathbf{L}^0$ can be revisited and perhaps altered by the architecture is an avenue of further research. There could be cases where the slight easement of a limit could lead to an overall acceptable solution; we would like to be able to identify these cases and flag them for the user. In this vein, the addition of "firm" versus "hard" or "soft" constraints might be considered: those constraints which the user very greatly prefers to be met, but which do not indicate total failure if failed.

Some optimization routines use negative weights in the cost functional, to allow certain terms (e.g., a quality measure) to be maximized. Users must be very careful when doing this, because it becomes possible for the term to grow without bound as time goes to infinity. The cost goes to negative infinity, dominated by this term times its negative constant. If the user has determined that, due to the properties of his particular problem, this will not happen, then such a term can be used. This work does not investigate the possibility of adding such terms, and we could look to that in the future as well.

Finally, this work considered only cases with static obstacle sets. Research with dynamic obstacles is required to determine whether our architecture remains tractable in such cases if the obstacles follow known paths. Due to the computational complexity of performing numeric solutions to the calculus of variations problem, this technique cannot be used, as is, as a reactive or real-time planner. However, other work [40] has investigated the use of optimal controls as part of a receding-horizon planner, for example, that along with MILP could provide a more real-time *TPLAN* algorithm for our architecture.

Acknowledgment

This work was performed in part at the Naval Research Laboratory under funding from the Office of Naval Research under work order N0001404WX30001.

References

[1] Andersson, J., "A Survey of Multiobjective Optimization in Engineering Design," Technical report LiTH-IKP-R-1097, Department of Mechanical Engineering, Linköping University, Linköping, Sweden, 2000.

[2] Kosko, B., and Isaka, S., "Fuzzy Logic," *Scientific American*, Vol. 269, July 1993, pp. 76–81.

[3] Novák, V., "Fuzzy Logic: Applications to Natural Language," *Encyclopedia of Artificial Intelligence,* 2nd ed., edited by S. C. Shapiro, Wiley, New York, 1992, pp. 515–521.

[4] Perzanowski, D., Schultz, A. C., and Adams, W., "Integrating Natural Language and Gestures in a Robotics Domain," *Proceedings of the IEEE International Symposium on Intelligent Control*, National Institute of Standards and Technology, Gaithersburg, MD, 1998, pp. 247–252.

[5] Tversky, B., and Lee, P. U., "How Space Structures Language," *Spatial Cognition. An Interdisciplinary Approach to Representing and Processing Spatial Knowledge*, edited by C. Freksa, C. Habel, and K. F. Wender, Springer-Verlag, Berlin, 1998, pp. 157–175.

[6] Abella, A., and Kender, J. R., "Qualitatively Describing Objects using Spatial Prepositions," *Proceedings of IEEE Workshop on Qualitative Vision*, IEEE Publications, Piscataway, NJ, 1993, pp. 33–38.
doi: 10.1109/WQV.1993.262952

[7] Oliver, P., Maeda, T., and Tsujii, J., "Automatic Depiction of Spatial Descriptions," *Spatial Reasoning*, Vol. 2, 1994, pp. 1405–1410.

[8] Mukerjee, A., "Neat vs. Scruffy: A Survey of Computational Models for Spatial Expressions," *Computational Representation and Processing of Spatial Expressions*, edited by P. Olivier, and K.-P. Gapp, Kluwer Academic, Norwell, MA, 1998.

[9] Khatib, O., "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots," *The International Journal of Robotics Research*, Vol. 5, No. 1, 1986, pp. 90–98.
doi: 10.1177/027836498600500106

[10] Latombe, J.-C., *Robot Motion Planning*, Kluwer Academic, Norwell, MA, 1991.

[11] Arkin, R. C., *Behavior-Based Robotics*, The MIT Press, Cambridge, MA, 1998.

[12] Velásquez, J. C., "An Emotion-Based Approach to Robotics," *Proceedings of the 1999 IEEE/RSJ International Conference on Intelligent Robotics and Systems*, IEEE Publications, Piscataway, NJ, 1999, pp. 235–240.

[13] Likhachev, M., Kaess, M., and Arkin, R. C., "Learning Behavioral Parameterization Using Spatio-Temporal Case-Based Reasoning," *Proceedings of the 2002 IEEE International Conference on Robotics and Automation (ICRA 2002)*, IEEE Publications, Piscataway, NJ, 2002, pp. 1282–1289.
[14] Santamaría, J. C., and Ram, A., "Learning of Parameter-Adaptive Reactive Controllers for Robotic Navigation," *Proceedings of the World Multiconference on Systems, Cybernetics, and Informatics* (CSI '97), 1997.
[15] Aaron, E., Sun, H., Ivančić, F., and Metaxas, D., "A Hybrid Dynamical Systems Approach to Intelligent Low-Level Navigation," *Proceedings of Computer Animation 2002*, 2002, pp. 154–163.
[16] Goldstein, S., Karavelas, M., Metaxas, D., Guibas, L., Aaron, E., and Goswami, A., "Scalable Nonlinear Dynamical Systems for Agent Steering and Crowd Simulation," *Computers and Graphics*, Vol. 25, No. 6, 2001, pp. 983–998.
[17] Shapiro, D., and Langley, P., "Separating Skills from Preference: Using Learning to Program by Reward," *Proceedings of the Nineteenth International Conference on Machine Learning*, 2002, pp. 570–577.
[18] Fiorini, P., and Shiller, Z., "Motion Planning in Dynamic Environments Using Velocity Obstacles," *International Journal of Robotics Research*, Vol. 17, No. 7, 1998, pp. 760–772.
[19] Shiller, Z., Large, F., and Sekhavat, S., "Motion Planning in Dynamic Environments: Obstacles Moving along Arbitrary Trajectories." *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2001)*, 2001.
[20] Zang, J., Raczkowsky, J., and Herp, A., "Emulation of Spline Curves and Its Applications in Robot Motion Control," *Proceedings of the IEEE Conference on Fuzzy Systems*, 1994, pp. 831–836.
[21] Hwang, J.-H., Arkin, R. C., and Kwon, D. S., "Mobile Robots at Your Fingertips: Bezier Curve On-Line Trajectory Generation for Supervisory Control," *Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, IEEE Publications, Piscataway, NJ, 2003, pp. 1444–1449.
[22] LaValle, S. M., and Kuffner, J. J. Jr., "Randomized Kinodynamic Planning," *International Journal of Robotics Research*, Vol. 20, No. 5, 2001, pp. 378–400.
[23] Fiorini, P., and Shiller, Z., "Time Optimal Trajectory Planning in Dynamic Environments," *Journal of Applied Mathematics and Computer Science*, Vol. 7, No. 2, 1997, pp. 101–126.
[24] Shiller, Z., and Gwo, Y.-U., "Dynamic Motion Planning of Autonomous Vehicles," *IEEE Transactions on Robotics and Automation*, Vol. 7, No. 2, 1991, pp. 241–249.
doi: 10.1109/70.75906
[25] Holland, J. H., *Adaptation in Natural and Artificial Systems*, Univ. of Michigan Press, Ann Arbor, 1975.
[26] Edgewood, F. Y., *Mathematical Physics*, P. Keagan, London, 1881.
[27] Pareto, V., *Cours D'Economie Politique*, Vols. I and II, F. Rouge, Lausanne, 1896.
[28] Farina, M., and Amato, P., "On the Optimal Solution Definition for Many-Criteria Optimization Problems." *Proceedings of the NAFIPS–FLINT International Conference 2002*, 2002, pp. 233–238.
[29] Aguirre, A. H., Rionda, S. B., Coello Coello, C. A., Lizárraga, G. L., and Montes, E. M., "Handling Constraints Using Multiobjective Optimization Concepts," *International Journal for Numerical Methods in Engineering*, Vol. 59, No. 15, 2004, pp. 1989–2017.
doi: 10.1002/nme.947
[30] de Weck, O., and Jones, M. B., "Isoperformance: Analysis and Design of Complex Systems with Known or Desired Outcomes," *Proceedings of the 14th Annual International Council on Systems Engineering*, 2004.
[31] de Weck, O., Miller, D. W., and Moiser, G. E., "Multivariable Isoperformance Methodology for Precision Opto-Mechanical Systems," *43rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, 2002.
[32] Kim, I. Y., and de Weck, O., "Adaptive Weighted-Sum Method for Bi-Objective Optimization: Pareto Front Generation," *Structural and Multidisciplinary Optimization*, Vol. 29, No. 2, 2005, pp. 149–158.
doi: 10.1007/s00158-004-0465-1
[33] Schouwennaars, T., De Moor, B., Feron, E., and How, J., "Mixed Integer Programming for Multi-Vehicle Path Planning," *Proceedings of the European Control Conference*, 2001, pp. 2603–2608.
[34] Richards, A., and How, J., "Performance Evaluation of Rendezvous Using Model Predictive Control," *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, 2003.
[35] Garcia, I., and How, J. P., "Trajectory Optimization for Satellite Reconfiguration Maneuvers with Position and Attitude Constraints," *Proceedings of the 2005 American Control Conference*, 8–10 June 2005, vol. 2, pp. 889–894.
[36] Bryson, A. E. Jr., and Ho, Y. C., *Applied Optimal Control*, Blaisdell, Waltham, MA, 1969.
[37] Kirk, D. E., *Optimal Control Theory: An Introduction*. Dover, New York, 2004 (reprint of 1970 edition by Prentice-Hall, Upper Saddle River, NJ).
[38] Roberts, S., and Shipman, J., *Two-Point Boundary Value Problems: Shooting Methods*, Elsevier, New York/Amsterdam, 1972.

[39] von Stryk, O., "Numerical Solution of Optimal Control Problems by Direct Collocation," *Optimal Control – Calculus of Variations, Optimal Control Theory and Numerical Methods, Number 111 in International Series of Numerical Mathematics*, edited by R. Bulirsch, A. Miele, J. Stoer, and K.-H. Well, Birkhauser, Basel, 1993.

[40] Henshaw, C. G., "A Unification of Artificial Potential Function Guidance and Optimal Trajectory Planning," *Proceedings of the 28th AAS Annual Rocky Mountain Guidance and Control Conference*, 2005, pp. 219–234.

[41] Kierzenka, J., and Shampine, L., "A BVP Solver Based on Residual Control and the Matlab PSE," *ACM Transactions on Mathematical Software*, Vol. 27, No. 3, 2001, pp. 299–316.
doi: 10.1145/502800.502801

[42] Lennon, J., "An Architecture for the Autonomous Generation of Preference-Optimized Trajectories," Ph.D. Dissertation, University of Maryland Department of Aerospace Engineering, 2006.

[43] Lennon, J., and Atkins E., "Intelligent Weight Selection for Trajectory Optimization," *Proceedings of InfoTech@Aerospace*, 2005.

[44] Henshaw, C. G., "A Variational Technique for Spacecraft Trajectory Planning," Ph.D. Dissertation, University of Maryland Department of Aerospace Engineering, 2003.

[45] Lennon, J., and Atkins, E., "Multi-Objective Spacecraft Trajectory Optimization with Synthetic Agent Oversight," *Journal of Aerospace Computing, Information and Communication*, Vol. 1, No. 1, 2004 1–20.

[46] Tsiotras, P., "Stabilization and Optimality Results for the Attitude Control Problem," *AIAA Journal of Guidance, Control, and Dynamics*, Vol. 19, No. 4, 1996, pp. 772–779.
doi: 10.2514/3.21698

Kelly Cohen
Associate Editor