JOURNAL OF AEROSPACE COMPUTING, INFORMATION, AND COMMUNICATION
Vol. 2, January 2005

Multi-Objective Spacecraft Trajectory Optimization with Synthetic Agent Oversight

Jamie A. Lennon*
Naval Research Laboratory, Washington, DC, 20375

and
Ella M. Atkins†
University of Maryland, College Park, Maryland, 20742

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 feasible solutions. This work integrates a rule-based planner inspired by human cognition with an optimal controls trajectory planner to automatically construct trajectories that do not require manual inspection or adjustment. The cognitive agent translates mission goals into cost function weights expected to produce motions that appropriately trade fuel and time efficiency as well as proximity to obstacles. The quality of the resulting full-state trajectory is then evaluated based on a set of computed trajectory features and specified constraints. Although each trajectory is mathematically optimal with respect to its dynamics and the weighted cost function, the agent may find it unacceptable locally (e.g., passes through an obstacle) or globally (e.g., requires too much fuel). The violating condition(s) are either translated to a new weight set or the trajectory is locally repaired, iterating until an acceptable trajectory is generated or the domain is deemed unsolvable. An ideal planar robot implementation introduces the models and provides intuitive baseline results. A three-dimensional spacecraft implementation is presented, a domain in which fuel savings and safety are critical for success.

Nomenclature

$\{A\}$	=	action set for planning problem p_0
a	=	generic dynamical equation
bc	=	boundary conditions for $X < t_0, \mathbf{x_0}, \mathbf{x_f} >$
$\mathbf{c}$	=	cubic spline coefficients in $o_i(r_i)$
c_s	=	coefficient of sliding friction
$\mathbf{F_i}$	=	trajectory feature vector for planning state s_i
$\mathbf{f_i}$	=	fuel component of $\mathbf{u_i}$ over $\mathbf{t_i}$ for planning state s_i
G	=	feature-based goal state required by the high-level strategic planner
$G_\sigma(\sigma)$	=	dynamic equation for the modified Rodrigues vector $R(\sigma)$
g	=	generic cost function

Received 18 August 2004; revision received 22 November 2004; accepted for publication 8 December 2004.
*Graduate Fellow, Naval Center for Applied Research in Artificial Intelligence, 4555 Overlook Ave. S.W. Student Member.
†Assistant Professor, Aerospace Engineering Dept./Space Systems Lab. Senior Member AIAA.

H	=	rotational inertial matrix of a robotic spacecraft
$\{HIST\}$	=	history of the optimal trajectory search for planning state s_i
I	=	full feature-based [symbolic] initial state provided by the high-level strategic planner
i, j	=	generic indices
J_i	=	integrated cost over trajectory ($\boldsymbol{t_i}, \boldsymbol{x_i}, \boldsymbol{u_i}$)
$\boldsymbol{J(x,u,t\{O\})}$	=	domain-dependent multi-objective cost function with weights $\boldsymbol{W_i}$
k	=	number of obstacles in $\{O\}$
$\boldsymbol{L_i}$	=	feature vector limits (constraints) for planning state s_i ($\boldsymbol{L_0}$ = initial/default limit set)
m	=	mass of a robotic vehicle
n	=	number of planning states s_i expanded to solve planning problem p_0
$\{O\}$	=	set of k obstacles $\{O_1,O_2,...,O_k\}$
$o_i(r_i)$	=	penalty function for robotic vehicle nearness to obstacle i in $\{O\}$
P_S	=	ACT-R production set for strategic planning operations
P_T	=	ACT-R production set specifically related to trajectory planning decisions
p_0	=	trajectory planning problem $<bc, \boldsymbol{W_0}, \boldsymbol{L_0}>$
R_i	=	radius of circular obstacle i in $\{O\}$
r_i	=	distance from robotic vehicle to center of obstacle i in $\{O\}$
S	=	matrix representation of the cross product
s_i	=	current state of the plan proposed to solve p_0
$\boldsymbol{t_i}$	=	vector of trajectory time points $\{t_1, ..., t_m\}$ for planning state s_i
$\boldsymbol{u_i}$	=	control actuation vector over $\boldsymbol{t_i}$ for planning state s_i
$v_i(r'_i)$	=	penalty function for robotic vehicle speed near obstacle i in $\{O\}$
$\boldsymbol{W_i}$	=	cost function weighting factor vector used in planning state s_i
X	=	solution $<J_n, L_n, \boldsymbol{t_n}, \boldsymbol{x_n}, \boldsymbol{u_n}>$ returned for planning problem p_0
$\boldsymbol{x_i}$	=	position/velocity state vector over $\boldsymbol{t_i}$ for planning state s_i
δJ	=	variation of the cost functional J
$\boldsymbol{\lambda_i}$	=	time-varying vector of Lagrange multipliers for planning state s_i
μ	=	approximation of the universal gravitational constant used when one body is much larger than the other
$\boldsymbol{\sigma_i}$	=	angular position component to state vector $\boldsymbol{x_i}$ for planning state s_i
$\boldsymbol{\tau_i}$	=	electrical component of $\boldsymbol{u_i}$ over $\boldsymbol{t_i}$ for planning state s_i
$\boldsymbol{\omega_i}$	=	rotational velocity component to state vector $\boldsymbol{x_i}$ for planning state s_i

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 robot 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 lifespan ends. Preserving vehicle health is another priority, and all of this must be done while respecting the dynamical constraints of the vehicle, and the dynamical properties of its environment.

Optimal control theory is a well-developed tool that can assist in just such problems. Goals, priorities, and constraints can be encoded in a cost functional. The cost functional is then optimized via the calculus of variations and returns both an optimal full-state trajectory and the control inputs needed to follow it. The construction of cost functionals, however, is often a non-trivial task. Some constraints which can be imposed in theory make the problem computationally intractable. Further, the process of adjusting the relative weights of the cost functional components to reflect the user's intentions is often an iterative process with no clear or principled guidelines. The iteration is typically carried out by a human expert who compares the trajectory that is optimal with respect to the cost functional to what he "really meant" to request. Such a process may work well for a spacecraft whose entire trajectory can be plotted out years in advance, but it does not work well for space, air, and surface vehicles that require some level of autonomy.

We aim to replace the human expert with a synthetic one that can be deployed onboard the robot. We provide it with an understanding of potentially desirable trajectory qualities as well as knowledge about how changing the parameters of the cost functional is likely to affect the trajectory. It also has the ability to decide when a trajectory is close enough to acceptable, except for one small deviation – then correct that deviation itself and recheck the resulting trajectory. For example, if a trajectory satisfies all constraints except that it passes just a short distance into a known obstacle, the cognitive agent – the intelligent autonomous overseer onboard the robot – can reroute the path around the obstacle, interpolate the velocities needed, and then check to ensure that these changes did not create a new problem (e.g., violating a constraint on acceleration). To our knowledge, this is the first time artificial intelligence techniques have been applied to the trajectory optimization problem. Our method of combining a model of human decision-making with a computational technique for obtaining optimal or near-optimal trajectories is novel and of use to the autonomous vehicles community.

Getting to a particular location is typically the province of path planning. Many robust techniques exist and have been implemented on mobile robots operating in complex environments. Voronoi diagrams, tangent graphs, cell decomposition and potential fields[1] all are viable path planning methods. However, they must all be augmented in some way to allow the use of a cost function that involves more than minimizing path length or distance from obstacles. Robots require fuel, power, and time resources to move about their environment. To fully define a "trajectory", a "path" (i.e., sequence of positions) must be augmented with velocities and angular motion parameters (e.g., heading, angular velocity). Resource costs in the form of forces/torques and traversal times can then be computed from the governing equations of motion. To incorporate quantities such as fuel and time into a traditional path planner's cost function, system "state" must be augmented with velocities, etc. An exhaustive search through a space of discretized dynamic parameter values (e.g., velocities) given constraints (e.g., limited accelerations) could theoretically be used to augment each path segment with a good or even optimal trajectory. However, computational efficiency is poor, and optimality is subject to the level of dynamic parameter discretization.

Optimal control algorithms build full trajectories rather than paths. Calculus of variations[2] is used to minimize a functional, which can include both a cost function and constraints on the state (imposed by system dynamics or otherwise). If the dynamics of a certain mobile robot are used as constraints, the procedure will only return a trajectory that the robot is capable of following. When a cost function is included, the trajectory will be otherwise minimized with respect to that. This method is not global and performs the minimization in the area around a given initial solution guess, but techniques have been developed to increase this area and the general robustness of the associated numeric solution.[3] This is an offline planning technique that is both mathematically rigorous and provably optimal, at the expense of computational complexity. It is therefore quite different from the work done in machine learning[4] which is for fast, reactive behaviors that do not have a global perspective. A global planner, in addition to avoiding the dead-ends that may break a reactive navigation system, can take advantage of maneuvers which, although immediately very costly, may result in a lower total cost. This comes, of course, at the price of requiring a model of the world.

The offline nature of the optimal controls approach is of some concern. There are, however, some near-optimal[5] approaches that can operate in real-time. These can offer orders of magnitude more fuel efficiency than a reactive approach and the consideration of a multi-objective cost functional. We have adopted a standard and accepted offline trajectory planner for this work, but since our synthetic agent reasons over trajectories and cost functionals, not trajectory generation methods, it will not be difficult to eventually integrate it with one such method.

Our optimal control cost function contains three terms: fuel use, clearance from obstacles, and time. Prior to this work, cost function weights were set in an ad hoc fashion, often determined experimentally. While we will develop our basic behavior-eliciting weights experimentally, we have developed rules to dictate weight adjustment "within" as well as "between" behaviors. Perhaps the most analogous work is the hybrid dynamical systems approach.[6] In this work on low-level navigation, a dynamic “comfort level” is used to adjust the weighting parameters of repulsor fields surrounding environmental obstacles. This is a purely reactive method, however, that does not attempt to calculate or minimize any costs over the entire trajectory.

Weighting factors are needed for cost functions that have more than one term. “The shortest path that uses the least amount of fuel” is often neither the shortest possible path, nor the path that uses the least fuel, but one which strikes a balance between them. The relative weights of these terms determine what sort of balance results. Substantial oversight is often required to analyze the sensitivity of solution characteristics to cost function weights, and this sensitivity analysis may be specific to particular problems rather than fully generalizable. Typically, researchers test different weight combinations until one that produces the desired behavior is found or use a default weight set (e.g., all weights equal[3]). Since the quantities being weighted can be of different units and even different orders of magnitude, there is often no more principled technique available.

Changing the weights changes the resulting behaviors. If saving fuel and avoiding obstacles are highly prized, the robot may travel slowly, avoiding rapid accelerations and looping far around obstacles. If a short completion time is heavily weighted, the robot will zoom forward, dodging obstacles at high speeds with low clearance. Human observers might give emotionally-inspired names to these behaviors: careful, reckless. But this work is different from emotionally-based behaviors.[7] In that work, different goals and environmental factors contribute to producing emotions, which then inhibit or excite certain behaviors. The value of a new behavior is computed from both a weighted sum of "releasers" or triggers for the behavior, and excitory or inhibitory impulses generated by other, active behaviors. The weights are static, so the system depends on sufficiently high levels of releasers to trigger a new behavior. This is entirely in keeping with the theory behind behavior-based robotics.[8] However, it does not allow for computational optimization of any kind. Our research is aimed at eliciting behaviors that are optimal with respect to specific physical quantities (e.g., fuel, time, and distance from obstacles). Given the agent’s goals and environment, some of these quantities may be perceived as more important than others, and weighted more heavily, resulting in a behavior that is optimal for this agent, with these goals, in this particular environment. Emotional language may be used to characterize these behaviors-aggressive, fearful, or cautious, for example - but we do not seek to model emotions as a means of achieving these behaviors.

We will instead use a computational cognitive modeling system to recognize what kind of behavior is mostly likely to be successful, given the goal and the environment at the moment. Then, we use the trajectory planner to make physical motions best representing that “stereotyped” behavior. Systems like ACT-R,[9] Soar,[10] and EPIC,[11] all try to model not only the end result of human cognition, but also the process by which those results are reached. We chose to use ACT-R, although the other architectures could be adopted as well. In ACT-R, procedural rules fire in the presence of certain "chunks" of symbolic declarative memory. This is a serial system, using the bottleneck as a point of coordination among different cognitive modules. This makes it a very attractive option for implementing on a physical robotic system. Finally, ACT-R also has an extension, ACT-R/S, which supports spatial awareness. The Naval Research Laboratory has leveraged this into a model of perspective-taking[12] which we plan to integrate as part of the cognitive model in our architecture.

As a preliminary, we disambiguate terminology used throughout this paper. The word “robot” is used to denote an embodied autonomous or semi-autonomous vehicle operating in an environment. An “agent” is not necessarily embodied; however, it is an intelligent actor in the system. Our cognitive model is software, but it performs an intelligent function for the robot (or robot simulator) on which it is installed. We often refer to it as “the cognitive agent” or “the agent.” To emphasize that all members of an intelligent group work within the same environment as a team, we refer to them all as “agents” with specifications: “robotic agent,” “software agent,” and “human agent.”

This paper is organized as follows: We present an overview of the system architecture and a review of optimal control theory. We discus in detail the ACT-R model we have developed. Our first set of results is for a 2-DOF point robot moving in the plane, a sort of simplified rover case. We discuss its assumed dynamics, the cost functional used, and the results of adjusting the parameters of the cost functional. We then repeat this treatment for a 6-DOF spacecraft flying in flat space with limited control authority. We conclude with a brief summary and discuss extensions to a multi-agent case, to provide flexible but optimized formation navigation.

II. Architecture

Figure 1 shows an outline of the agent's processes. At the center sits the ACT-R model, overseeing all activities. The human user interacts with this module, monitoring events rather than directly participating in trajectory generation processes. The ACT-R trajectory planning agent accepts a planning problem, 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.

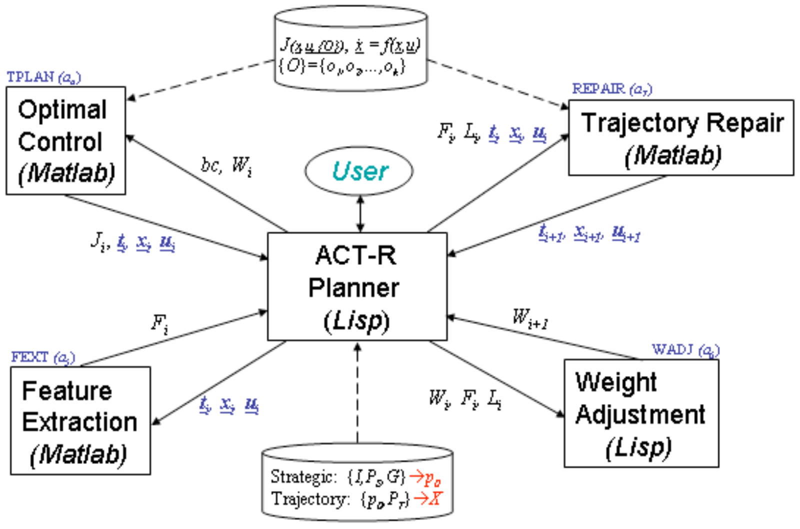

$J(\underline{x},\underline{u},\{O\}),\ \dot{\underline{x}} = f(\underline{x},\underline{u})$
$\{O\}=\{o_1,o_2,\ldots,o_k\}$
TPLAN (a_s)
Optimal Control (Matlab)
REPAIR (a_7)
Trajectory Repair (Matlab)
$bc,\ W_i$
User
$F_i,\ L_i,\ \underline{t}_i,\ \underline{x}_i,\ \underline{u}_i$
$J_i,\ \underline{t}_i,\ \underline{x}_i,\ \underline{u}_i$
ACT-R Planner (Lisp)
$\underline{t}_{i+1},\ \underline{x}_{i+1},\ \underline{u}_{i+1}$
F_i
W_{i+1}
FEXT (a_i)
WADJ (a_j)
Feature Extraction (Matlab)
$\underline{t}_i,\ \underline{x}_i,\ \underline{u}_i$
$W_i,\ F_i,\ L_i$
Weight Adjustment (Lisp)
Strategic: $\{I,P_S,G\}\rightarrow p_0$
Trajectory: $\{p_0,P_T\}\rightarrow X$

Fig. 1 Component architecture and dataflow.

A trajectory planning problem p_0 is defined as $<bc,\ \boldsymbol{W_0},\ \boldsymbol{L_0}>$ with boundary conditions $bc=<t_0,\ \boldsymbol{x_0},\ \boldsymbol{x_f}>$. The goal is to return feasible and optimal solution $X=<J_n,\ \boldsymbol{L_n},\ \boldsymbol{t_n},\ \boldsymbol{x_n},\ \boldsymbol{u_n}>$, where J_n and $\boldsymbol{L_n}$ summarize solution cost and the feature limits/constraints, respectively, and the set $<\boldsymbol{t_n},\ \boldsymbol{x_n},\ \boldsymbol{u_n}>$ specifies the full-state trajectory to be executed. ACT-R incrementally builds a history of activities $\{HIST\}=\{HIST_1,\ HIST_2,\ \ldots\}$ with each $HIST_i$ described by an action A_i and planning state s_i. It can then use *{HIST}* to identify which weight adjustment strategies it has already employed, to avoid infinite loops. For the trajectory planning problem, action set *{A}* is defined as *{INIT, EVAL, RET, TPLAN, FEXT, WADJ, REPAIR},* with blue text denoting actions performed as ACT-R calls to external functions. Each of these modules will be explained in more detail below. Each planning state $s_i=<\boldsymbol{W_i},\ \boldsymbol{F_i},\ \boldsymbol{L_i},\ J_i,\ \boldsymbol{t_i},\ \boldsymbol{x_i},\ \boldsymbol{u_i}>$ includes all information pertaining to a single trajectory generation cycle, with blue text denoting the continuous trajectory ACT-R tracks and passes to external functions but does not process as part of its feature space. Cast in the context of a complete mission planning problem, $\{I,P_S,G\}$ is the ACT-R knowledge base for strategic decision-making, where I is the full feature-based [symbolic] initial state, G is the feature-based goal state, and P_S is the ACT-R production set for strategic planning operations. P_T is the ACT-R production set specifically related to trajectory planning decisions; it includes all productions required to control the Fig. 1 processes as well as perform actions *INIT, EVAL,* and *RET.*

Figure 2 shows the possible paths through the architecture. *INIT* initializes the problem state, p_0. Next, *TPLAN* generates an initial optimal trajectory. *FEXT* extracts the relevant trajectory features, $\boldsymbol{F_i}$, and sends them to *EVAL*. If all $\boldsymbol{F_i}$ are within the limits $\boldsymbol{L_i}$, the trajectory is good and the solution X is returned by *RET* to the user and the higher-level strategic planner. Otherwise, based on history *{HIST}* and any limit violations, *EVAL* decides to either adjust weights and re-plan the trajectory or repair the trajectory locally. If *EVAL* decides to adjust the weights $\boldsymbol{W_i}$, it calls *WADJ*, which uses a local rule set to decide which changes to make based on which limits were violated. These $\boldsymbol{W_{i+1}}$ are returned to *TPLAN* and the process iterates until a good trajectory is found and *RET* is called. If *EVAL* instead decides that local trajectory repair is appropriate, it calls *REPAIR* for this purpose. The repaired trajectory is re-evaluated by *FEXT* and *EVAL* to ensure that the repair process did not introduce any new problems. If it did not, *RET* fires as above. If it did, *EVAL* uses *{HIST}* to recall the pre-*REPAIR* state of the problem and calls *WADJ* to attempt a fix instead.

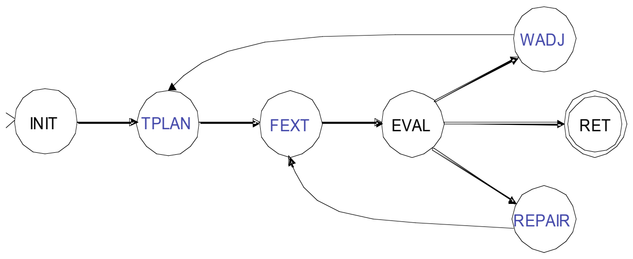


Fig. 2 Trajectory generation under ACT-R supervision.

The Fig. 1 supporting functions are modular and easily altered to fit into an existing problem domain. For example, the optimal controller (*TPLAN*), which generates the optimal trajectory and the control inputs needed to attain it, could be replaced with one of the near-optimal controllers referenced above. The feature extraction unit (*FEXT*) operates on the returned trajectory, identifying properties that may be of concern to the ACT-R planner. Trajectory features include: cost, final time, fuel used, energy used, obstacle penalty incurred, minimum distance from each obstacle (min_sep), average minimum distance from all obstacles, total path length, maximum velocity, maximum speed, maximum rotational velocity, average velocity, average speed, average rotational velocity, number of changes in sign of velocity, number of changes in sign of rotational velocity, percent of trajectory spent in velocity plateaus, maximum acceleration, minimum acceleration, maximum rotational acceleration, minimum rotational acceleration, average acceleration, average rotational acceleration, number of sign changes in acceleration, number of changes in sign of rotational acceleration, and percent of trajectory spent in acceleration plateaus.

The weight adjustment module (*WADJ*) contains rules for changing the scalable parameters (e.g., weights and constants embedded in obstacle penalty functions) if the ACT-R planner determines that the returned trajectory does not meet limits $\boldsymbol{L_0}$ and needs to be recomputed with new parameters. Finally, the trajectory repair unit (*REPAIR*) may be called to make small alterations to trajectories when $\boldsymbol{L_0}$ is nearly satisfied and a total recomputation of the trajectory is deemed too time-consuming.

Theoretically, some of the trajectory constraints or limits $\boldsymbol{L_0}$ could be encoded in the cost functional or incorporated as direct constraints for the optimal controller. In particular, there are techniques for representing limits on the state variables, so "no-go" zones could be defined for both position and velocity. Unfortunately, in practice, this creates singularities that violate the assumptions underlying the numeric algorithms used to solve the dual-boundary value problem. The common solution is to use techniques like potential fields that approximate a step function, but are smooth to the second derivative. That, of course, introduces the possibility that, given sufficient motivation by other weighting factors, the trajectory can be driven through the potential field. So the process requires some oversight to ensure that this does not happen, which the ACT-R planner provides.

Other limits are not easily encoded in a cost functional. For example, in these problems the final time can be left free (a variable to be solved for) or fixed (specified by the user). The free end time case can have penalties to drive the final end time towards the neighborhood of a certain known time, if desired. But there is not an obvious way to specify, for instance, "As quickly as possible, but no more than three hours." It is, however, very simple to check the final time of a computed trajectory using *FEXT*. If $\boldsymbol{L_0}$ contains a value for t_f, ACT-R will decide to either call on *WADJ* to adjust other scalable parameters to induce a shorter, faster trajectory or it will simply fix the end time at the specified maximum.

A primary theme of this work is to bring together representations and algorithms from AI and control communities to autonomously solve the larger problem of unsupervised trajectory generation for complex dynamic systems. The use of MATLAB and Lisp was not accidental: it facilitates transition to new domains as well as initial implementation. Extensive MATLAB "toolboxes" have been developed and are widely used by researchers and practitioners in academia and industry. The symbolic data management inherent in Lisp facilitates supervision of the trajectory generation process, and the use of ACT-R, a symbolic planning system, enables modeling of the knowledge used to make the strategic decisions that actually pose each planning problem p_0. Certainly, if this work leads to a widely-used system, the issue of language should be revisited since the tradeoff will then favor speed of execution over ease of code generation.

A. ACT-R Model

An ACT-R model consists primarily of "chunks" of data in a knowledge base, a series of production rules, P_T, and a set of buffers (e.g., goal, retrieval, vision) where chunks can be "remembered" and both trigger and be altered by P_T. The P_T firings are serial, and only one chunk may be stored in a buffer at a time. Productions can be used to create new knowledge chunks, such as the history *{HIST}* described above. Further "subsymbolic" processes are controlled via set parameters. These can control the probability with which a particular chunk is recalled, or which production will fire (if several are appropriate). All this together is a powerful tool that can recall what has been done and can choose, potentially from several options, what should be done in the present and the future.

In our architecture, *INIT* takes high-level goals and encodes them into chunks useable by the *EVAL* module. Although these goals can come from any appropriate strategic planner, in our present implementation, the goals come directly from the human user and the initial weight set $\boldsymbol{W_0}$ and constraint set $\boldsymbol{L_0}$ are standardized constant vectors. Although beyond the scope of this work, *INIT* should ideally be capable of interpreting and assigning $\boldsymbol{W_0}$ values based on p_0. With sufficient domain knowledge, *INIT* could even be made to transform strategic goals such as "keeping under cover" into new terms for the cost functional.

EVAL is very much the center of the ACT-R procedure. This core process begins simply, by comparing the features $\boldsymbol{F_i}$ returned by *FEXT* to the $\boldsymbol{L_0}$ generated by *INIT*. When all $\boldsymbol{F_i}$ are within the bounds set by $\boldsymbol{L_0}$, nothing more needs to be done except to return the trajectory via *RET*. When some $\boldsymbol{L_0}$ are violated, *EVAL* has choices to make. It must be aware of what it has tried before so that fruitless iterations are avoided. It must decide if the current trajectory is a candidate for *REPAIR*, or if *WADJ* is a more appropriate approach. Trajectories are candidates for *REPAIR* only if certain restrictions are met: that the violation is of a constraint that can be fixed by *REPAIR*, that the violation is not too large, that there are not too many such violations. Otherwise, *WADJ* is to be done, and *EVAL* needs to prepare input for that routine. Which $\boldsymbol{L_0}$ were violated and by how much? More importantly, are there conflicting $\boldsymbol{L_0}$ demands? The strategic planner may unintentionally request competing limits that cannot be mutually satisfied. *EVAL* must recognize these situations and deal with them. If the strategic planner (human or otherwise) has requested a low level of autonomy, *EVAL* should inform the planner that $\boldsymbol{L_0}$ cannot be met. If a higher degree of autonomy has been requested, *EVAL* needs to be able to intelligently decide which $\boldsymbol{L_0}$ can be relaxed and by how much to get a solution that is as close to the planner's request as possible. Once *EVAL* finds that an identified optimal trajectory meets all constraints, *RET* returns the trajectory and control input schedule to the strategic planner. If *EVAL* made changes to $\boldsymbol{L_0}$, these are reported as well.

B. Feature Extraction (FEXT)

The goal of the *FEXT* module is to extract numerical attributes from the continuous trajectory that, in some fashion, quantify overall features of that trajectory. The current list of features can be found in Table 1. Some are maximum or minimum values which are straightforward to express in $\boldsymbol{L_0}$; 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.

C. Weight Adjustment and Trajectory Repair (WADJ and REPAIR)

WADJ and *REPAIR* are both typically highly dependent on domain-specific knowledge. Their overall purpose was discussed above; their specific implementation for different cases will be discussed with those results, below.

D. Optimal Control (TPLAN)

The *TPLAN* module generates the trajectory and control input schedule via the use of optimal control theory. It contains both the dynamics for the robotic system and the cost functional being optimized. In this research, we consider a cost functional with three terms: control effort (energy and/or fuel use), time, and clearance from obstacles. Other terms, based on domain and goals, are certainly possible, but we have captured the standard terms in our existing cost functional.

Generally, a cost functional is of the form:

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

where ***x(t)*** is the state vector and ***x'(t)***is its derivative. A variation in the functional, *δJ*, can be defined for small changes of *g(**x(t),x'**(t),t)*. If a relative minimum for *J* exists, it is necessary that *δJ* be zero at that point. Applying the definition of *δJ* to Eq. (1) yields the Euler Equation:

$$\frac{\partial g}{\mathrm{d}\mathbf{x}}\left(\boldsymbol{x}^*(t),\boldsymbol{x}'^*(t),t\right)-\frac{d}{\mathrm{d}t}\left[\frac{\partial g}{\mathrm{d}\mathbf{x}'}\left(\boldsymbol{x}^*(t),\boldsymbol{x}'^*(t),t\right)\right]=0 \tag{2}$$

where ***x*(t)*** is an extremal state vector and ***x'*(t)*** its derivative.

The problem is to find an admissible input (or control) vector ***u*(t)*** that causes a system described by the differential equations in Eq. (3) to follow an admissible trajectory ***x*(t)*** that minimizes the cost functional Eq. (4).

$$\boldsymbol{x}'(t)=a(\boldsymbol{x}(t),\boldsymbol{u}(t),t) \tag{3}$$

$$J(\boldsymbol{u})=\int_{t_0}^{t_f} g(\boldsymbol{x}(t),\boldsymbol{u}(t),t)\mathrm{d}t \tag{4}$$

At all points along an admissible trajectory, Eq. (3) holds and can be rewritten:

$$a(\boldsymbol{x}(t),\boldsymbol{u}(t),t)-\boldsymbol{x}'(t)=0 \tag{5}$$

and added to *g(**x(t),u**(t),t)* with Lagrange multipliers ***λ*** to form an augmented cost functional:

$$J_a(\boldsymbol{u})=\int_{t_0}^{t_f} g_a(\boldsymbol{x}(t),\boldsymbol{x}'(t),\boldsymbol{u}(t),\boldsymbol{\lambda}(t),t)\mathrm{d}t$$

or, rewriting,

$$\begin{aligned}J_a(\boldsymbol{u})=\int_{t_0}^{t_f} & g(\boldsymbol{x}(t),\boldsymbol{u}(t),t)\\ & +\boldsymbol{\lambda}^T\left[a(\boldsymbol{x}(t),\boldsymbol{u}(t),t)-\boldsymbol{x}'(t)\right]\mathrm{d}t\end{aligned} \tag{6}$$

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, which is defined as:

$$\begin{aligned}&H(\boldsymbol{x}(t),\boldsymbol{u}(t),\boldsymbol{\lambda}(t),t)\\ &\quad=g(\boldsymbol{x}(t),\boldsymbol{u}(t),t)+\boldsymbol{\lambda}^T[a(\boldsymbol{x}(t),\boldsymbol{u}(t),t)]\end{aligned} \tag{7}$$

The necessary conditions are then:

$$\boldsymbol{x}'^*(t)=\frac{\partial H}{\partial\boldsymbol{\lambda}}(\boldsymbol{x}^*(t),\boldsymbol{u}^*(t),\boldsymbol{\lambda}^*(t),t) \tag{8a}$$

$$\boldsymbol{\lambda}'^*(t)=-\frac{\partial H}{\partial x}(\boldsymbol{x}^*(t),\boldsymbol{u}^*(t),\boldsymbol{\lambda}^*(t),t) \tag{8b}$$

$$0=\frac{\partial H}{\partial\mathbf{u}}(\boldsymbol{x}^*(t),\boldsymbol{u}^*(t),\boldsymbol{\lambda}^*(t),t) \tag{8c}$$

for all $t\in[t_0,t_f]$. For a fixed final time and a fixed final state, we have boundary conditions

$$\boldsymbol{x}(\mathrm{t}_0)=\boldsymbol{x}_\mathbf{0} \tag{9a}$$

$$\boldsymbol{x}(t_f) = \boldsymbol{x}_f \tag{9b}$$

which gives the equations needed to determine the constants of integration. Optimization over final time is enabled by identifying the time at which the returned solution has minimum-cost. The split boundary value problem is not in general solvable in closed-form, so numeric methods were employed in this work. We use the MATLAB routine `bvp4c` as our numeric solver. This routine uses the collocation method to optimize the trajectory and is fully described elsewhere.[13] It is a somewhat slow offline planner, suited for planning trajectories through well-characterized and predictable environments (such as space). Unpredictable environments require a more reactive approach that could be used as *TPLAN* so long as a full-state trajectory is returned.

III. Point Mass Planar Robot

A simplified 2-D domain model was developed as an intuitive baseline case for our architecture and as a method of developing initial modules to populate the Fig. 1 architecture. The basic ACT-R "supervisor" model P_T, largely domain-independent, was applied to this problem. The optimal controls model, on the other hand, is somewhat unique to each dynamic system, and is described below for the 2-D planar robot. Weight $\boldsymbol{W_i}$ adjustment criteria are then derived from a study of trajectory features $\boldsymbol{F_i}$ as a function of relative weight values in $\boldsymbol{W_i}$, and an example is presented that illustrates complete operation from problem p_0 initialization (*INIT*) to return (*RET*) of dynamically-feasible solution X that meets feature limits $\boldsymbol{L_i}$, maintained as constant set $\boldsymbol{L_0}$ in the current implementation.

A. Optimal Control Model (TPLAN)

We began our investigations with a 2-DOF 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:

$$\begin{bmatrix} \boldsymbol{x}'(t) \\ \boldsymbol{x}''(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -c_s/m \end{bmatrix} \begin{bmatrix} \boldsymbol{x}(t) \\ \boldsymbol{x}'(t) \end{bmatrix} + \begin{bmatrix} 0 \\ \boldsymbol{u}(t)/m \end{bmatrix} \tag{10}$$

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.

B. Terms of the Cost Functional

In robotic applications, two concerns are usually paramount: conserving fuel or battery power and not running into obstacles. Additionally, there may be time constraints on a mission. Equation (11) gives the cost functional J and weight set $<W_1, W_2, W_3, LIM>$. Each term is described more fully below.

$$J = \int_{t_o}^{t_f} \left(W_1 \sum_{j=1}^{length(\boldsymbol{u})} \left(u_j^2(t) \right) + W_2 + W_3 \max_{i \in \{O\}} \left(o_i(r_i) \right) \right) \mathrm{d}t \tag{11}$$

Energy Use

Since we are considering a hypothetically battery-powered vehicle, we followed Kirk[2] in adding a minimum-energy term. We have simplified his representation somewhat; he includes a potentially different weight for each $u_j^2(t)$ in the control vector. We do, however, account for varying individual control vector elements in the 6-DOF example shown below, where translational actuators require fuel and rotational actuators require electrical power.

Time

Since J is an integral, the cost functional only needs a constant term, W_2, to minimize time. Over the integral, the resulting $W_2 * t_f$ will be minimized.

Clearance to Obstacles

To keep the vehicle away from obstacles, we add the term $W_3 * \max_{i \in \{O\}} \left(o_i(r_i) \right)$, presuming simple circular obstacle geometries. We assume that our agent has an *a priori* map of the region it will traverse, possibly obtained from an orbiter or fly-over. $o_i(r_i)$ is a function which increasingly penalizes the agent as it approaches obstacle i. W_3 is the relative weight in overall cost from Eq. (11), and r_i is the distance from the vehicle to obstacle i's center. $o_i(r_i)$

is maximum over the center of the obstacle, attains fixed value K at the obstacle boundary a distance R_i from the obstacle center, and decreases to zero at a distance *LIM* away from the obstacle's edge (Fig. 3). These constraints are described by Eq. (11) and also include a smoothness condition *o'(LIM)*. A third-order polynomial solution [Eq. (12)]that meets these constraints was selected as $o_i(r_i)$. This solution is positive within the region of influence (r_i <*LIM*) and effectively repels the path given sufficient K, *LIM* values.

$$\begin{cases} o_i(R_i) = K \\ o_i(LIM) = 0 \\ o_i'(LIM) = 0 \end{cases} \tag{12}$$

$$o_i(r_i) = \begin{cases} K(c_1 r_i^3 + c_2 r_i^2 + c_3 r_i + c_4), & r_i \le R_i \\ K(c_5 r_i^3 + c_6 r_i^2 + c_7 r_i + c_8), & R_i < r_i \le LIM \\ 0, & r_i > LIM \end{cases} \tag{13}$$

In our analysis of the effects of changing the parameters on the trajectory, we included changes not only in W_3 but also in *LIM*. They had distinctly different effects, as discussed below.

In addition to these terms, the system dynamic equations are adjoined to J as discussed above. This ensures that only dynamically feasible trajectories will be considered.

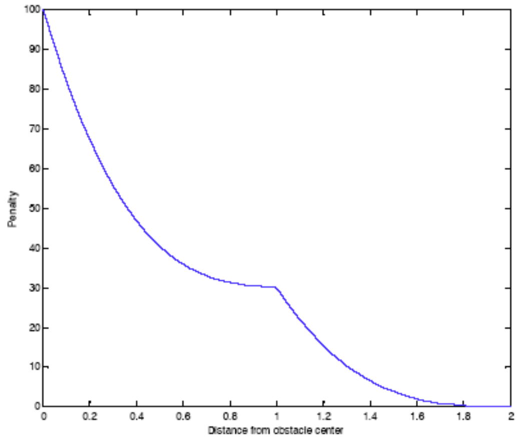


Fig. 3 Piecewise cubic potential field used as a penalty function for clearance from obstacles.

C. Weight Adjustment (*WADJ*)

To acquire and encode expert-level knowledge on trajectory adjustment, we constructed a series of experiments. We identified a series of "trajectory features" – gestalt qualities such as total path length, maximum acceleration, the amount of time spent in velocity or acceleration "plateaus," the number of changes of sign in the acceleration, and so on. (See Table 1 for complete list). Not every quality will be relevant to every goal trajectory, of course; ride quality measures like changes in acceleration won't matter if the vehicle is structurally strong and no passengers are onboard. We then generated sample trajectories for various combinations of the weights W_i for some different obstacle fields. Of note is that, while the absolute magnitude of *LIM* is of importance, the absolute magnitudes of the other weights are not. Their effects are a function of their relative values. The weight set <2, 1, 1,*> gives the same trajectory as <100, 50, 50,*>, for example. Therefore, we found it most useful to analyze the W_i in ratios. We looked for relationships between trajectory features and the ratios W_1/W_2 and W_3/W_2. The choice of W_2 as the normalizing factor was arbitrary.

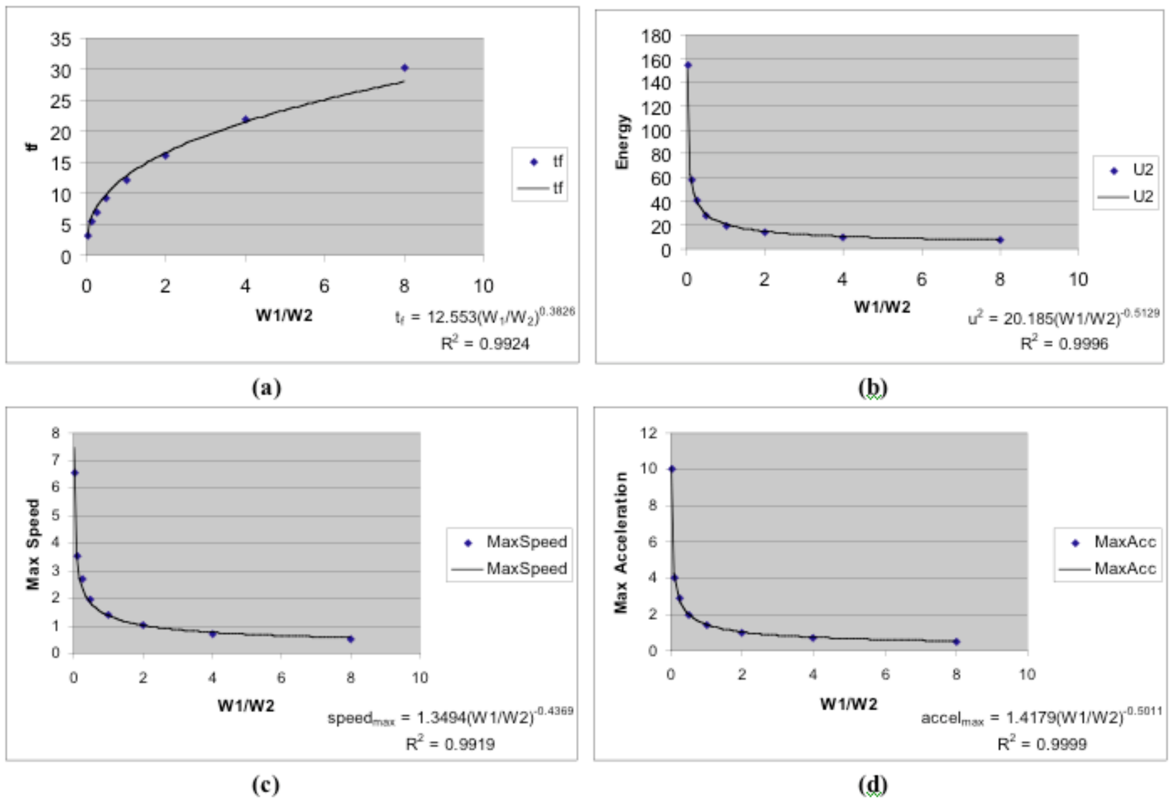


Fig. 4 Power rules for selected trajectory features (a) final time (b) energy (c) maximum speed and (d) maximum acceleration, in a field with no obstacles.

Results are encouraging. A baseline zero-obstacle case yields power rules relating many of the energy and fuel-related features to the ratio W_1/W_2 (Fig. 4). Figure 4a shows the optimal time as a function of W_1/W_2. When the ratio is very small – that is, W_1 (the weight on energy) is small compared to W_2 (the time weight), the optimal solution has a very short final time. This is the same as saying that when time is prized more highly than energy, trajectories should take less time to complete. As the W_1/W_2 ratio grows, W_1 equals then exceeds W_2 and now energy is more important than time. We see a corresponding increase in the final time of the optimal trajectory. Figure 4b shows the relationship between the energy used and the W_1/W_2 ratio. Again, when W_1/W_2 is small, W_1 (energy) is less important than W_2 (time). So these optima have very high energy uses. As W_1 increases with respect to W_2 and their ratio increases, conserving fuel becomes a priority. The optimal solutions become those that use smaller amounts of energy. Figures 4c and 4d show relationships between the maximum velocity and acceleration that occur during the trajectory versus W_1/W_2. Attaining high velocities and accelerations takes a larger amount of energy than attaining lower ones. Also, it seems likely that faster trajectories (with lower final times) are more likely to have high maximum velocities and accelerations. So we see again that when W_1/W_2 is small and energy is "unimportant" compared to time, the maximum velocities and acceleration are much higher than for larger W_1/W_2 ratios where conserving energy takes precedence.

These rules are somewhat less robust when one and then three obstacles were added to the field (Figs. 5 and 6). With these environment changes, the value of equation constants varied and the correlation coefficient that indicates quality of the data fit was reduced. This is hardly unexpected. The presence of one or more obstacles will of course result in different data compared to a traverse of a clean, straight-line path from target to goal. But results from the obstacle cases are sufficiently similar to the zero-obstacle case to justify using the zero-obstacle results as a framework or heuristic for adjusting W_1 and W_2. These rules are not absolutely predictive; they are guidelines that, depending on the number and location of obstacles in the environment, may be more or less accurate. This is why we need the intelligent oversight of the *EVAL* module. When the *WADJ* rules are less than perfectly predictive, the intelligent oversight guides future iterations in the direction of an acceptable trajectory, if one exists.

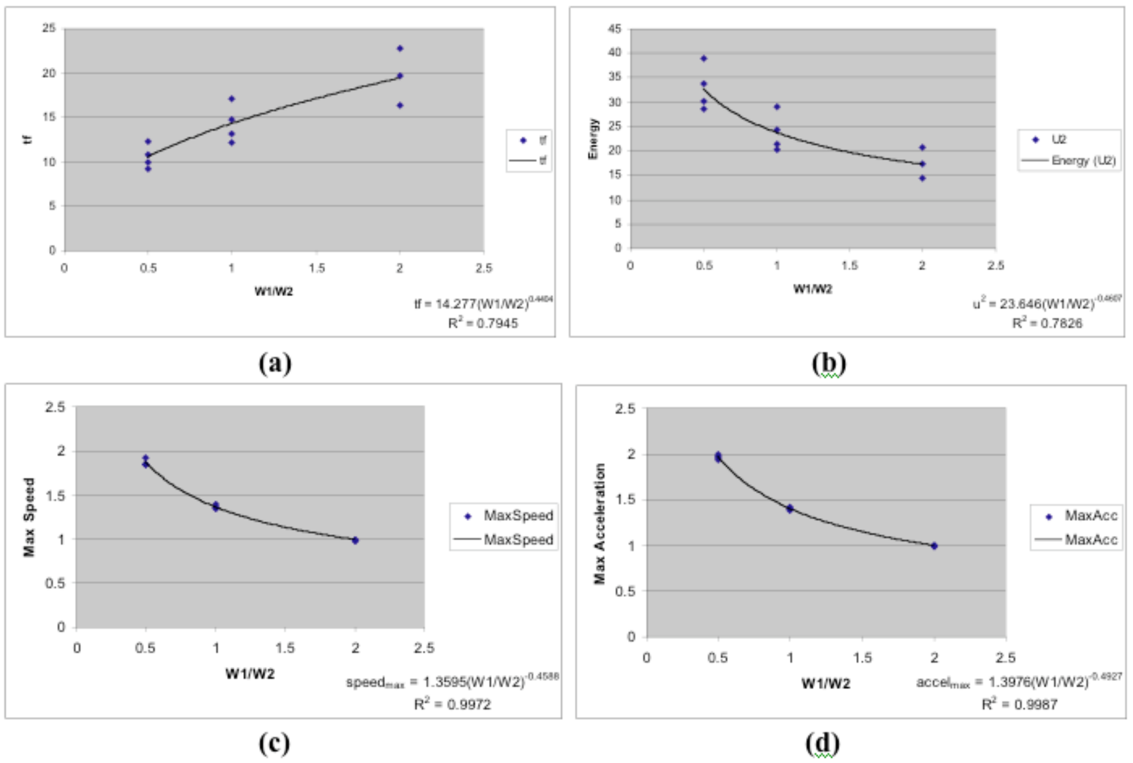


Fig. 5 Power rules for selected trajectory features (a) final time (b) energy (c) maximum speed and (d) maximum acceleration, in a field with one obstacle.

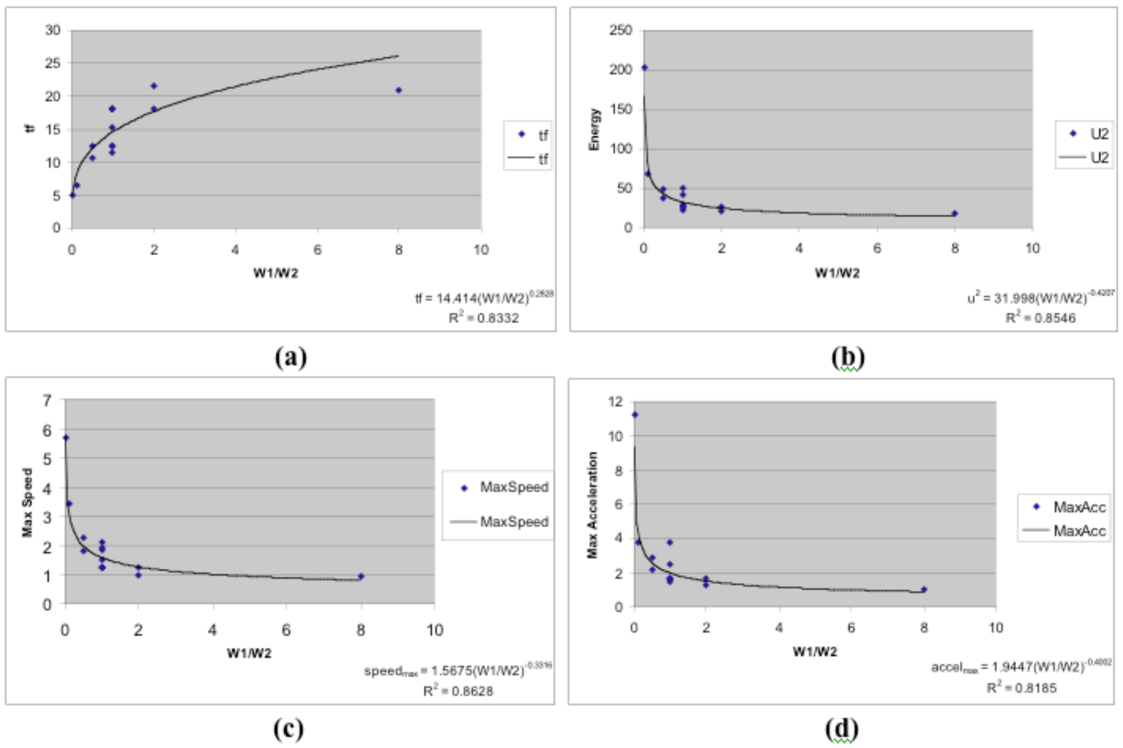


Fig. 6 Power rules for selected trajectory features (a) final time (b) energy (c) maximum speed and (d) maximum acceleration, in a field with three obstacles.

Path-related features, such as minimum separation from obstacles, did not vary with the W_1/W_2 ratio (Fig. 7a). There was a weak relationship with W_3 (Fig. 7b), but *LIM* actually had the most impact (Fig. 7c). This is graphically illustrated in Fig. 8. For all of the blue paths, *LIM*=3, $W_1/W_2 = 1$ and W_3 varies from 1 to 5. For the green paths, $W_1=W_2=W_3$ and *LIM* is varied from 1 to 7. Clearly, changing *LIM* is the way to effect major changes in path qualities for this simple planar robot.

The resulting *WADJ* rules are listed in Table 1. More rules could be written – one or more for each trajectory feature in $\boldsymbol{F_i}$ – but these rules capture the salient features we felt were most important for planar robot and spacecraft domains. When the trajectory features found using *FEXT* do not meet limits $\boldsymbol{L_0}$, *WADJ* rules are used to determine how to change the weights $\boldsymbol{W_i}$ to bring the trajectory features into compliance with the limits.

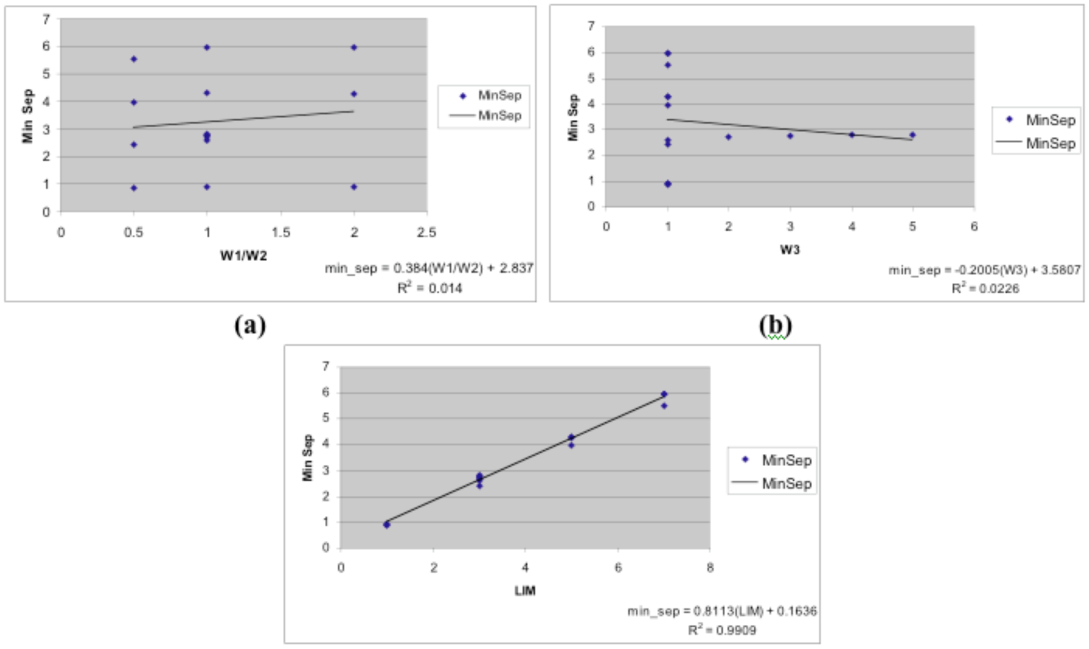


Fig. 7 Minimum separation from a single obstacle along a path as a function of (a) W1/W2 (b) W3 (c) LIM.

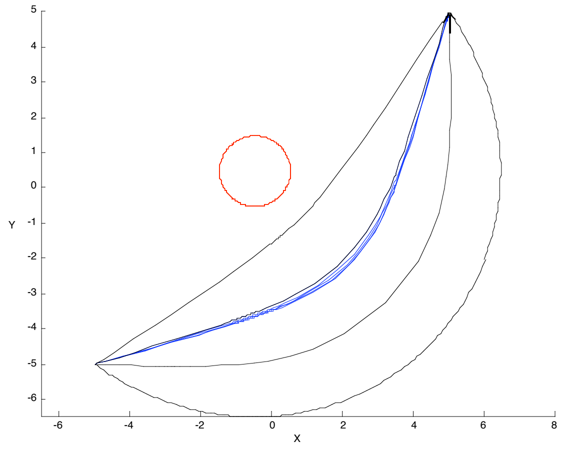


Fig. 8 Effects on path of changing W3 (solid blue lines) and LIM (dashed black lines).

D. Trajectory Repair (REPAIR)

The trajectory repair module, *REPAIR*, is included to deal with a small set of special conditions. It may be that the trajectory returned meets almost all constraints in ***L₀***, but still fails one or two limits by a small amount. One common example is a path that passes just barely inside an obstacle. With our current, computationally expensive version of *TPLAN*, it is undesirable to recompute the entire trajectory to repair one or two small errors. It is worthwhile, at least, to determine if a small, smooth adjustment of the trajectory to within the set limits is feasible. *REPAIR* makes the adjustment; the result is passed back to *FEXT* to determine if the changes were good or if they only introduced more errors.

Table 1 Weight Adjustment Rules for the Planar Robot

IF	THEN
	Use power rule $u^2 = c_1(W_1/W_2)^{-0.5}$
u^2 not met	Compute c_1 from current W_1, W_2 and u^2 values
	Use desired u^2 and c_1 to compute new W_1/W_2
	Use power rule $t_f = c_2(W_1/W_2)^{0.4}$
new W_1/W_2 computed	Compute c_2 from current t_f and old W_1/W_2
	Use c_2 and new W_1/W_2 to compute new expected t_f
	Test trajectories in range +/- 10% of new t_f value
	Use power rule $t_f = c_2(W_1/W_2)^{0.4}$
t_f not met	Compute c_2 from current t_f and W_1/W_2
	Use desired t_f and c_2 to compute new W_1/W_2
	Test trajectories in range +/- 10% of new t_f value
Min separation from	Use linear rule *min_sep* = c_3 *LIM*
obstacle not met	Compute c_3 from least of all current *min_sep* values and current *LIM*
	Use desired *min_sep* and c_3 to compute new *LIM*

We currently are only investigating addressing one trajectory problem, the obstacle minimum separation constraint discussed above. *REPAIR* uses a cubic spline to match positions and velocities at two “good” trajectory points on either side of the region in violation, and then to interpolate smooth position and velocity curves between them (for all DOF) that circumvents the obstacle at or above the required minimum separation distance. Although *REPAIR* was motivated by preliminary examples where obstacle boundaries were in fact penetrated, in the case study presented below, we did not encounter paths that passed through the obstacles, so *REPAIR* was not called.

E. Planar Robot Case Study

We used a 10 x 10 field with two obstacles for this example. Since this is a simplified example, there are no particular units; distance is measured in “distance units,” time in “time units” and so on. We imposed the following ***L₀*** on the trajectory:

$$\boldsymbol{L_0} = \begin{cases} u_{tot}^2 \le 20 \\ \forall\{O\},(r_i - R_i) \ge 1.0 \end{cases} \tag{14a}$$

where u^2_{tot} is the total energy used, $\{O\}$ are the obstacles and $(r_i - R_i)$ is the distance of the point robot from the edge of each. As a starting guess, we took W_1=W_2=W_3 and *LIM*=3. The resulting trajectory (Fig. 9) used 25.83 energy units, violating $\boldsymbol{L_{0,1}}$. *WADJ* used this feature value, $\boldsymbol{F_{1,1}}$, and the W_1/W_2 ratio was used to compute the coefficient for the power rule governing energy use. Using a desired $\boldsymbol{F_{1,1}}$ of 20, a new W_1/W_2 value was computed. This value, together with a coefficient derived from the time power rule, was then used to estimate the new optimal time. This is an artifact of our particular instantiation of *TPLAN*, which converges much more quickly when a good guess for the general neighborhood of the true optimal time is supplied. The new $\boldsymbol{W_i}$ were passed to *TPLAN* and the process iterated. The results are shown as the first two iterations in Table 2, after which an acceptable trajectory was found (Fig. 10).

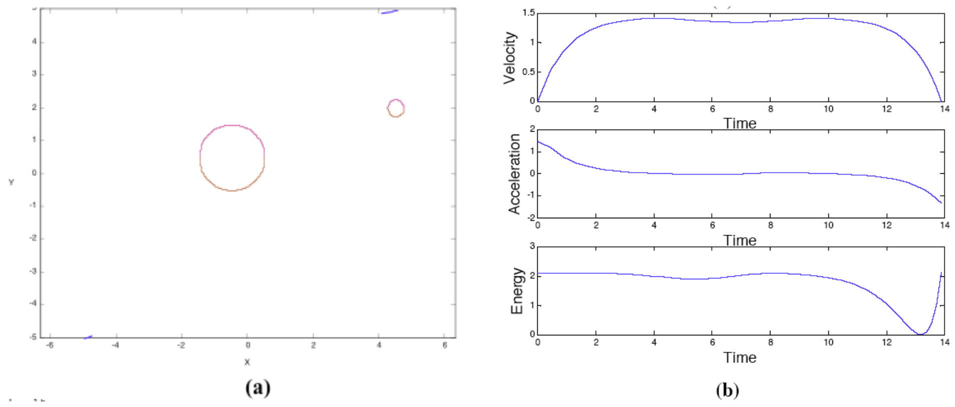


Fig. 9 Initial (a) path and (b) velocity, acceleration and energy use for two obstacle example.

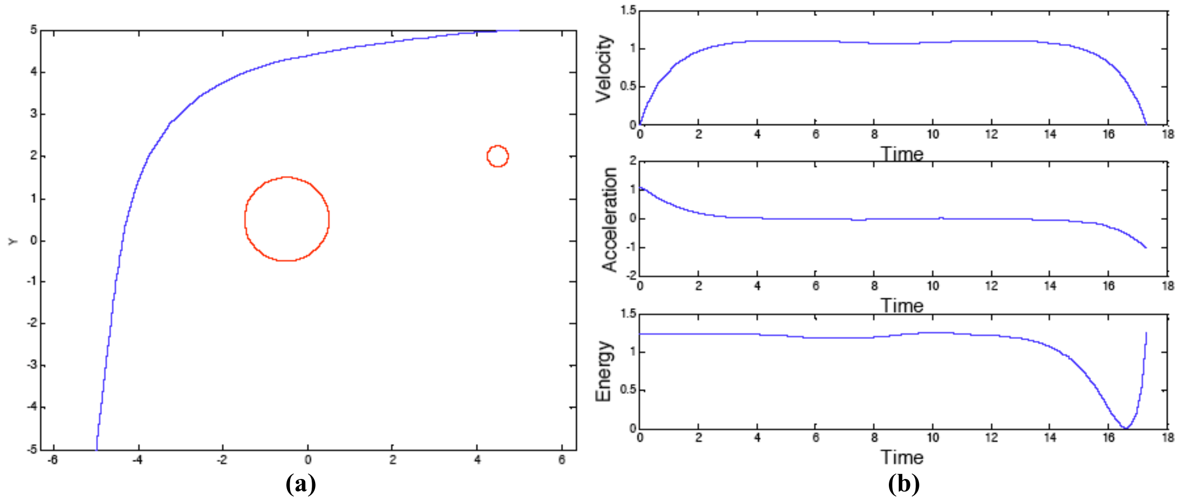


Fig. 10 (a) Path and (b) velocity, acceleration and energy use for two obstacle example with weight set <1.708, 1, 1, 3>.

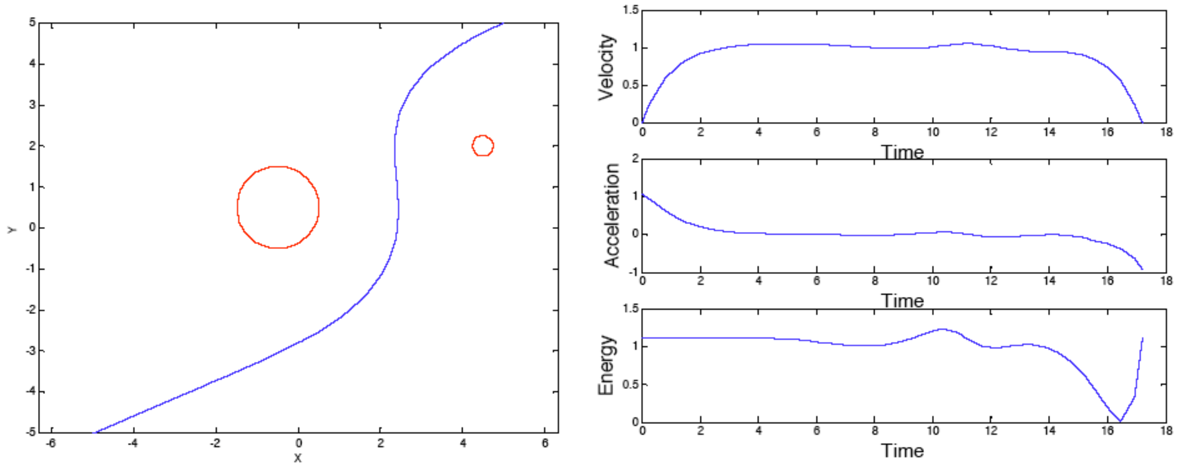
To extend the example, we added a third component to $\boldsymbol{L_0}$ at this point acting as "maximum separation" (e.g., appropriate should the vehicle wish to survey each obstacle it passes):

$$\mathbf{L_0} = \begin{cases} u_{tot}^2 \le 20 \\ \forall\{O\},(r_i - R_i) \ge 1.0 \\ \forall\{O\}\ ,(r_i - R_i) \le 2.0 \end{cases} \tag{14b}$$

Note that an unobstructed path will not be diverted to survey obstacles without an additional cost function term that attracts the path toward survey sites. Using the linear relationship between *LIM* and the minimum separation, *WADJ* calculated the coefficient based on current data and evaluated the *LIM* needed for a minimum separation distance of 2.0 distance units. This came out to 2.33. Sending the results back to *TPLAN* (with the same time schedule as Iteration 2, since the final time does not depend strongly on *LIM*) returned the results on the last line of Table 2. As an added and expected bonus, there is a fuel savings as well when the vehicle can more closely approach obstacles. Since $\boldsymbol{L_{0,1}}$ does not require that the energy be near 20 units, only less than 20 units, there is no reason to recompute the trajectory. Figure 11 shows the resulting path and trajectory information.

Table 2 Planar Robot Case Study Results

Iteration #	W_i	Energy	Min separation from Obstacle 1	Min separation from Obstacle 2	W_{i+1}
0	<1, 1, 1, 3>	25.83	2.00	1.94	<1.668, 1, 1, 3>
1	<1.668, 1, 1, 3>	20.24	2.00	1.94	<1.708, 1, 1, 3>
2	<1.708, 1, 1, 3>	18.82	2.57	2.73	<1.708, 1, 1, 2.33>
3	*<1.708, 1, 1, 2.33>*	*16.72*	*1.94*	*1.90*	



(a) **(b)**

Fig. 11 (a) Path and (b) velocity, acceleration and energy use for two obstacle example with weight set <1.708, 1, 1, 2.33>.

IV. □ Spacecraft Trajectory Optimization

A spacecraft flying through an obstacle field was modeled with dynamic model and optimal controls (*TPLAN*) procedure adopted from previous work by Henshaw and Sanner.[3] No external gravity fields are present; it is a "flat space" problem. The original algorithm was applicable to cases with no obstacles, with stationary obstacles, and with moving obstacles. Here, we show examples of the same stationary obstacle set solved for a range of weights.

A. Optimal Control Model

The spacecraft modeled has three fuel-powered saturating thrusters for translation (inputs $\boldsymbol{f_i}$) and three pairs of electrical rotation actuators with saturation (inputs $\boldsymbol{\tau_i}$) for rotation. Its 6-DOF dynamic equations are:

$$\begin{bmatrix} \mathbf{x}' \\ \mathbf{x}'' \\ \boldsymbol{\sigma}' \\ \boldsymbol{\omega}' \end{bmatrix} = \begin{bmatrix} \mathbf{x}' \\ -\mu\mathbf{x}/\|\mathbf{x}\|^3 \\ G_\sigma(\boldsymbol{\sigma})\boldsymbol{\omega} \\ H^{-1}S(H\boldsymbol{\omega})\boldsymbol{\omega} \end{bmatrix} + \begin{bmatrix} 0 \\ R(\boldsymbol{\sigma})\boldsymbol{f}(t)/m \\ 0 \\ H^{-1}\boldsymbol{\tau}(t) \end{bmatrix} \tag{15}$$

where $\boldsymbol{\sigma}$ is a modified Rodrigues vector[14] (a representation of angular position, chosen instead of the quaternion for computational reasons) and $\boldsymbol{\omega}$ is the rotational rate vector expressed in the body frame. Since $\boldsymbol{u}_i$, composed of $\boldsymbol{f}_i$ and $\boldsymbol{\tau}_i$, is limited, control switching curves were computed using Pontryagin's Minimum Principle. These discontinuous curves were then approximated with continuous versions for computational reasons.

B. Terms of the Cost Functional

The cost functional for the 6-DOF example shares some terms with the simple 2-DOF case, but has a few additional terms due to the increased complexity of the model. The cost functional is given in Eq. (15) and the individual terms explained below.

$$J = \int_{t_o}^{t_f} \left(W_1 \|\boldsymbol{f}(t)\|_1 + W_2 \sum_{j=1}^{length(\boldsymbol{\tau})} \left(\boldsymbol{\tau}_j^2(t)\right) + W_3 + W_4 \max_{i\in\{O\}} \left(o_i(r_i)\cdot v_i(r'_i)\right) \right) \mathrm{d}t \tag{16}$$

Fuel Use

For mass ejection thrusters, the form of the cost functional term is typically taken as $W_1 * \|\boldsymbol{f}(t)\|_1$.

Energy Use

The electrical rotation actuators use the same minimum-energy term as the point-robot model, above. In this example, the energy weight W_2 can be set separately from the fuel weight.

Time

The same constant weight, W_3, penalizes time here as in the point-robot model.

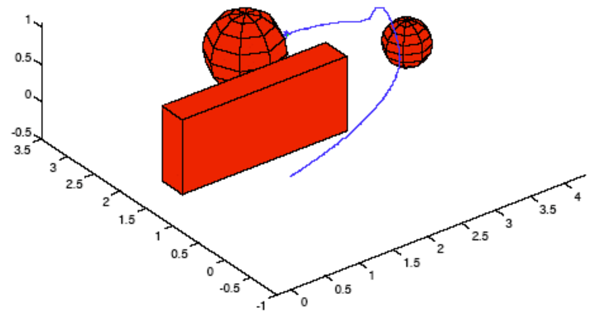
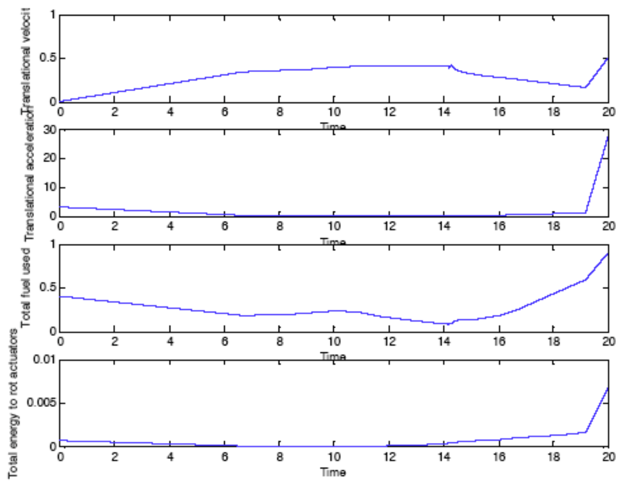
Clearance to Obstacles

In this example, perfect tracking of the trajectory is not assumed. Therefore, it is not only prudent to penalize nearness to obstacles, but also high velocities near obstacles. Otherwise, since the cost functional is optimized over a time integral, not a path integral, the planner can avoid high obstacle penalties by getting very close to obstacles, but passing them very quickly. This is not desirable behavior. So, in addition to penalizing nearness with $o_i(r_i)$, we add a penalty term $v_i(r'_i)$ which is based on the velocity of the spacecraft when it is within a certain distance of the obstacle. The total penalty is then the product of these two components, weighted by W_4. There is also an obstacle "influence" parameter which serves the same function as *LIM* in the 2-DOF case. Our weight set $\boldsymbol{W}_i$ for this example is then $<W_1, W_2, W_3, W_4, LIM>$. As in the 2-DOF case, the system dynamical equations are adjoined to Eq. (15) before the cost functional is minimized. This ensures that all the considered trajectories are dynamically feasible.

C. Sample Spacecraft Trajectories

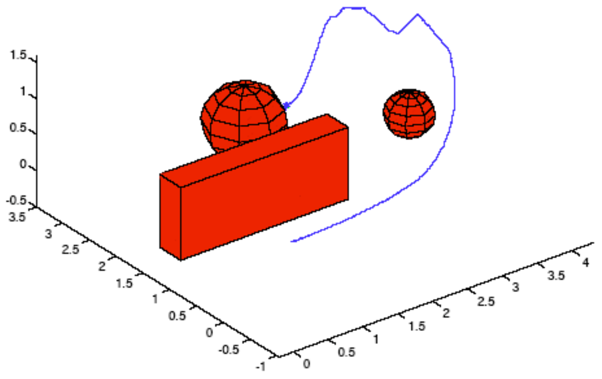
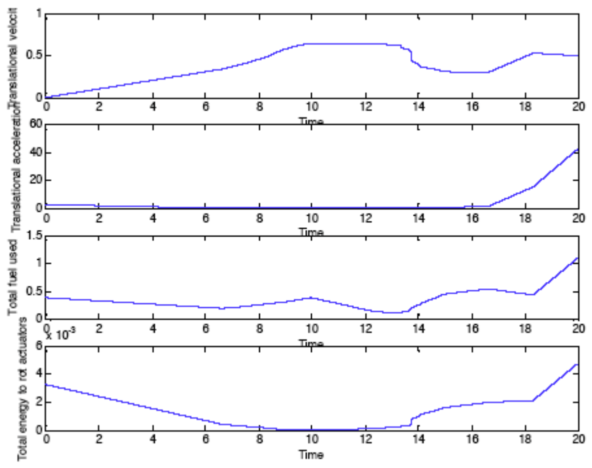
As an illustration of the effects of adjustable weights on the resulting trajectory, we provide figures for three different cases. Each of these was computed for a fixed end time of 20 time units, so changes to the time weight W_3 had no effect. This allows us to compare just the tradeoffs between fuel and safety (obstacle avoidance), which are more critical than maneuver time in many spacecraft operations.

Figure 12 shows a baseline case for weight set <1, 1, 1, 1, 1>. Changing *LIM* from 1 to 2 results in the path shown in Fig. 13. The pronounced corners appear to be an artifact of the shape of the potential field around the rectangular obstacle. Weight set <2, 2, 1, 1, 1>, giving a fuel and energy efficient path, is shown in Fig. 14. The potential to elicit different trajectory characteristics in this domain, as was done for the simple 2-DOF example, is apparent. We expect to find both power and linear weight adjustment rules in the 6-DOF domain as we did for the completed 2-DOF domain based on these similarities. The WADJ module for the spacecraft case is under construction and includes data from simulations with free end times.

(a) (b)

Fig. 12 (a) Path through space and (b) trajectory information for weight set <1, 1, 1, 1, 1>.

(a) (b)

Fig. 13 (a) Path through space and (b) trajectory information for weight set <1, 1, 1, 1, 2>.

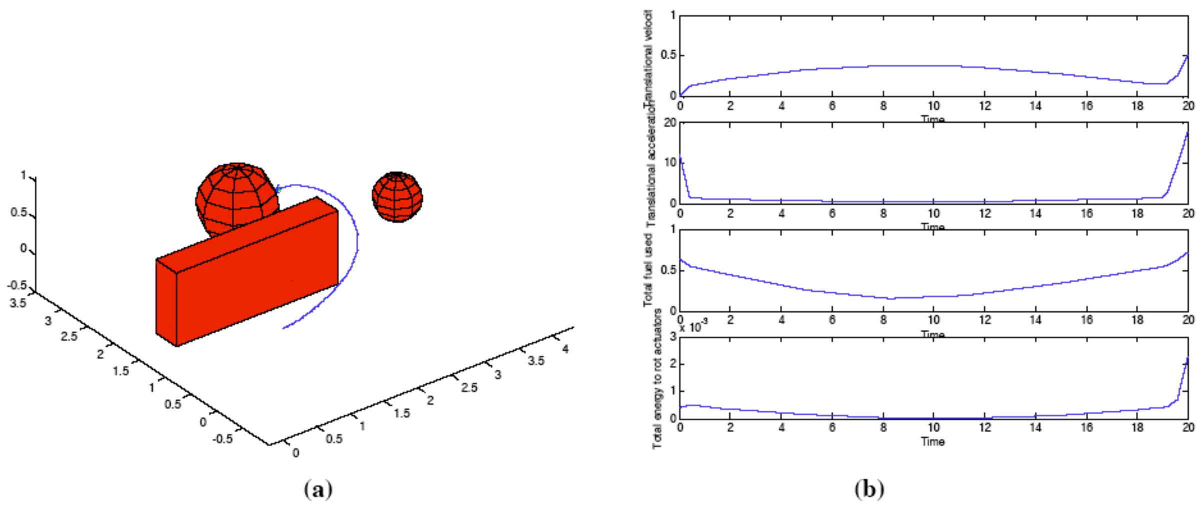


Fig. 14. (a) Path through space and (b) trajectory information for weight set <2, 2, 1, 1, 1>.

D. Architecture Scalability

The 2-DOF point robot model had a state vector with eight entries (x_1 and x_2, derivatives for both, plus four corresponding Lagrange multipliers). The rigid-body 6-DOF spacecraft model – a realistic and complex system model that accurately portrays flat space dynamics and thruster limitations – had a state vector with 24 entries. This increase in complexity was apparent at runtime. Ongoing data collection to develop *WADJ* rules has runtimes of about thirty minutes for most of the zero-obstacle cases, and runtimes of up to eight hours for the single-obstacle case. In contrast, the 2-DOF zero-obstacle runs converged in seconds, and the single-obstacle cases typically finished within 15-30 minutes.

This performance is acceptable only when extremely long planning cycles are possible, as is typical for spacecraft mission design and even detailed trajectory planning after launch given the significant drift time between maneuvers. Multi-hour performance per iteration is, however, not acceptable for online (real-time) execution in other air or ground/space robotic domains, given that one frequently needs to replan in the field because of some unexpected contingency. Vehicles like aircraft may not have a static "safe mode" they can return to even for a full minute, let alone a full hour, while the system replans the trajectory.

In our architecture, the vast majority of the computation time is taken up by the *TPLAN* module. Our current *TPLAN* is the MATLAB `bvp4c` algorithm, which is in no way intended to be a real-time computational tool. Although less commonly used, we have identified two near-optimal trajectory planners (Ref. 5 and an unpublished method under development at the Naval Research Laboratory) as possible alternatives to the current *TPLAN*. These offer trajectory costs that approach the optimal with runtimes that approach that of a reactive planner. The adaptation of one of these algorithms, or another similar one, and a comparison of the results to those obtained with our current *TPLAN* is a definite future goal.

V. Conclusions and Future Work

An architecture has been presented that integrates a symbolic planner with an optimal controls trajectory planner to automatically construct trajectories without the manual oversight traditionally required. Weight sets are selected based on mission constraints, and trajectories are validated against this constraint set since the mathematical optimization process is performed over a single total cost value rather than individual constraints such as those listed in Table 1. Unacceptable trajectories are repaired locally or re-optimized over different weight sets expected to better meet violated constraints. Specific implementations for a 2-D planar robot and 6-DOF spacecraft are presented along with results illustrating the significant effects of altering weights.

Identification of a universal set of rules for weight adjustment over all problem instances and all domains has proved to be a difficult task, leading us to conclude that the best ACT-R (or a human agent) can expect is to base initial weight sets on historical trends that work in most but not all cases. The use of an AI planner as the oversight agent is, in fact, strongly motivated by this challenge, since planners are specifically designed to search through the host of alternatives whenever its default rules (or heuristics) do not initially produce an acceptable solution. Because

ACT-R maintains a history of trajectory optimization attempts, it will try weight combinations until it exhausts the set of appreciably-different weight sets that might meet the constraints. However, in some cases, ACT-R may not be able to identify a weight set that produces a valid trajectory, either because no such trajectory exists or because the combination of (*TPLAN*, *EVAL*, *WADJ*) algorithms cannot generate such a solution. We have seen no such case in our limited set of examples, but such problems do, in fact, exist, and human users are often faced with the same issues. With a fully-autonomous system, a failure status would be returned to the meta-level rule set that posed planning problem p_0 in the first place, at which time either constraint set $\boldsymbol{L_i}$ would be relaxed or else a different plan that didn't require the solution for p_0 would be identified. This framework is a new method by which the trajectory generation and optimization process can be addressed in an autonomous system, and in such a way that it supports the higher-level goals of the system.

Work is in progress to extend the rule sets and the spacecraft implementation to better adjust weights and to function in the presence of central gravity fields. Although this paper enables comparison between a "baseline" (e.g., $\boldsymbol{W_i} = <1, 1, 1, 1>$) trajectory versus refined solutions that better meet constraints, we also plan to explore the tradeoff between optimized trajectories that are slow to compute (minutes or even hours for a single *TPLAN* execution) with suboptimal but real-time reactive trajectories resulting from a more traditional path planning approach. The challenge with this comparison will be to select a "representative" path planner generally accepted by the community and then to appropriately augment the result with velocities/forces required to compute cost and trajectory feature (Table 1) values needed for quantitative comparison of results. We also plan to study the performance of our ACT-R agent in comparison to expert and novice human overseers of the trajectory planning process, enabling a statistical performance characterization as well as potential improvement to the ACT-R rules based on expert strategies not yet incorporated as production rules.

We also plan to apply this research to multi-agent trajectory planning tasks. Teams of autonomous air, space, and ground agents, whether docking, in formation, or cooperatively achieving overall goals, are featured in most future NASA and Department of Defense missions. The methods outlined above provide a way to implement navigation for formations or single vehicles in close proximity that is optimizable and flexible provided environmental characteristics are known.

We will explore two approaches to the problem. The first is a "true optimal," where all vehicles are included in one large state vector. The formation is defined by cost functional constraints penalizing deviation from certain relative positions. The trajectories for all formation members are then computed simultaneously, resulting in an overall optimized system. This is computationally extremely expensive, and inspires a second approach. Here, we will first have the formation leader plan its optimal trajectory. It then becomes a known moving obstacle in the next vehicle's trajectory generation run. We can express as a penalty its deviation from a certain position relative to this "moving obstacle." It and the first vehicle are obstacles for the third vehicle's trajectory generation, and so on.

Acknowledgments

The authors would like to gratefully acknowledge Catherine McGhan from the University of Maryland, who collected most of the data upon which the parameter adjustment rules were based. This work was performed in part at the Naval Research Laboratory with funding from the Office of Naval Research under work order N0001404WX30001.

References

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

[2]Kirk, D. E., *Optimal Control Theory: An Introduction*, Prentice-Hall Inc., Englewood Cliffs, NJ, 1970.

[3]Henshaw, C. G., and Sanner, R. M., "A Variational Technique for Spacecraft Trajectory Planning," *2003 GSFC Flight Mechanics Symposium*, 2003.

[4]Santamaria, J. C., and Ram, A., "Learning of Parameter-Adaptive Reaction Controllers for Robotic Navigation," *Proceedings of the World Multiconference on Systemics*, *Cybernetics, and Informatics*, 1997.

[5]Miles, D. W., "Real-Time Dynamic Trajectory Optimization with Application to Free-Flying Space Robots," Ph.D. Dissertation, Dept. of Aeronautics and Astronautics, Stanford Univ., 1997.

[6]Aaron, E., Sun, H., Ivancic, F., and Metaxas, D., "A Hybrid Dynamical Systems Approach to Intelligent Low-Level Navigation." *Proceedings of Computer Animation 2002*, 2002, pp.154-163.

[7]Velasquez, J. D., "An Emotion-Based Approach to Robotics," *Proceedings of the 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1999, pp. 235-240.

[8]Brooks, R. A., "Intelligence Without Representation," *Artificial Intelligence Journal*, Vol. 47, 1991, pp. 139-159.

[9]Anderson, J. R., and Lebiere, C., *The Atomic Components of Thought*, Lawrence Erlbaum, Mahwah, NJ, 1998.

[10]Newell, A., *Unified Theories of Cognition*, Harvard Univ. Press, Cambridge, MA, 1990.

[11]Kieras, D., and Meyer, D. E., “An Overview of the Epic Architecture for Cognition and Performance with Application to Human-Computer Interaction,” *Human Computer Interaction*, Vol. 12, 1997, pp. 391-438.

[12]Trafton, J. G., Schultz, A. C., Perzanowski, D., Bugajska, M. D., Adams, W., Cassimatis, N. L., and Brock, D. P., “Children and Robots Learning to Play Hide and Seek,” *Cognitive Systems Research*, under review.

[13]Shampine, L. F., Kierzenka, J., and Reichelt, M. W., “Solving Boundary Value Problems for Ordinary Differential Equations in MATLAB With bvp4c,” The MathWorks, Inc., 2000, [available online at http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?ref=bvp_tutorial&objectId=3819&objectType=file (cited 18 Sept. 2003)].

[14]Tsiotras, P., “Stabilization and Optimality Results for Attitude Control Problem” *Journal of Guidance, Navigation, Control and Dynamics*, Vol. 19, No. 4, 1996, pp. 772-779.