Making Decisions about Motion

Jamie Lennon[1] **Ella Atkins[2]**

[1]Naval Research Laboratory, [2]University of Maryland
[1]4555 Overlook Ave. S.W., Washington, DC, 20375
[2]Space Systems Laboratory, Building 382, College Park, MD 20742
lennon@aic.nrl.navy.mil, atkins@glue.umd.edu

Abstract

There is often seen in the literature a disconnect between higher-level artificial intelligence and lower-level controls research. Too often, path planners do not account for the physical capabilities of the agents that will be following their paths. Similarly, the best controllers available need to be given trajectories from somewhere. This work presents a combined architecture that links a high-level cognitive model, which can handle symbolic information and make rational decisions using it, with a lower-level optimal controller for planning detailed robotic vehicle trajectories. The priorities of the cognitive model are passed to the controller in the form of a set of weights that vary the importance of terms in a cost functional. Using the calculus of variations, this cost functional is combined with the physical system dynamics and globally optimized, resulting in a full-state trajectory that is optimal with respect to the agent's strategic goals and physical capabilities.

Introduction

We desire an intelligent robot that can reason, plan ahead, and make decisions based on its goals, its environment, and the desires of any teammates it might have. A cognitive model capable of symbolic inference can, with a large enough rules set and world information, do this for us. It can make judgment calls not only about what actions it should undertake, but *how* it should perform them - quickly, efficiently, cautiously, aggressively.

These behaviors, illustrated with possibly emotionally loaded words, are simply stereotypes of classes of trajectories. An "aggressive driver" is one who drives quickly, leaves little clearance between his vehicle and others, and engages in rapid accelerations between and within lanes. We interpret these dynamic characteristics of the vehicle's motion as aggression, regardless of the actual emotional state of the driver. A "cautious driver" would have different characteristics. He might still drive very quickly on a highway, but would leave larger clearances between his car and others, and would maintain a more constant speed with fewer and more gradual accelerations. How can we elicit these characteristics in position, velocity, and acceleration in a variety of domains?

This research is aimed at providing autonomous mobile agents with such a capability to plan their dynamic behaviors intelligently, based on goals, environmental concerns, and teammate input. Task execution involves tradeoffs between quantities such as speed and fuel efficiency, so the agent must understand which are more highly prized, and the conditions under which that evaluation might change.

We express the various dynamic features of the system as terms in a cost function. Each term is a quantity we wish to minimize to some degree or another: power use, time, or risk from approaching obstacles. Expressed mathematically, we can use weighting terms to give different priorities to the elements of the cost function before using an optimal controls approach to generate trajectories and inputs for the system. 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 (Velasquez 1999). 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 (Brooks 1991). 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). 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, 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 find the physical motions best representing that "stereotyped"

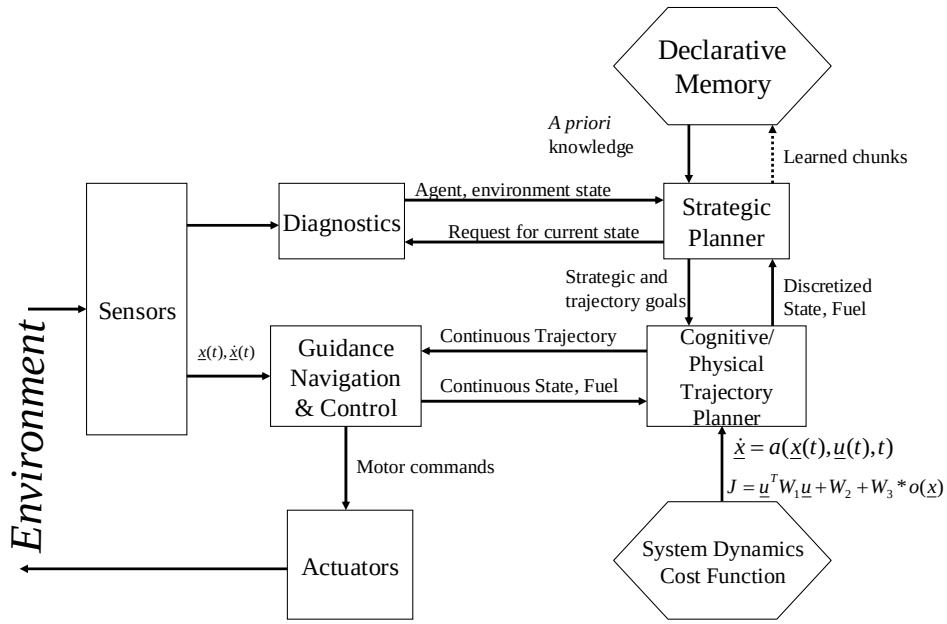


Figure 1: Integration of Cognitive and Physics-based Planners

behavior. A cognitive model has the responsibility of selecting the correct set of weights, given the agent's goals and environment. These weights can then be further tuned by the model for even better performance. We assume the existence of a higher-level strategic planner which produces these goals.

Figure 1 shows the integration of cognitive and physical planning technologies. Inside the Cognitive/Physical Trajectory Planner, a symbolic inference engine (ACT-R) (Anderson and Lebiere 1998) is linked with a continuous trajectory planner to construct smooth motions that reflect behavioral goals as well as physical realities (e.g., obstacle avoidance, limited acceleration). This link is achieved without requiring architectural extension to either the cognitive or trajectory planning components, using ACT-R to specify relative weights in a multi-objective cost function over which trajectories are optimized. As we will illustrate, simple weight adjustments can dramatically vary trajectory

We begin with a review of previous research in path planning, cognition, and behavioral control. Then we discuss our goals for the cognitive model and overview the optimal controls method used to generate trajectories in this work. A brief description of our dynamic model and cost functional is provided. Results include demonstrations of the path and trajectory properties and an example in weight adjustment. This paper introduces a method for a high-level, symbolic agent and physics-based trajectory planner to work together to plan robot motions. Numerous applications ranging from single-robot surveillance to multi-agent missions will ultimately benefit from such integration.

Related Work

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 (Latombe 1991) all offer 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 path length minimization. Agents 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 (Kirk 1970) build full trajectories rather than paths, using the calculus of variations. Additionally, the technique allows constraints to be placed on the trajectories. These can include, for example, system dynamics (including real-world concerns such as motor saturation), resulting in a path that the agent is guaranteed to be able to follow. They can also include a multi-term cost function that will allow us to elicit our

various behaviors. 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 (Santamaria and Ram 1997), 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 maneuver 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.

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 are working toward rules to dictate weight adjustment "within" as well as "between" behaviors. Perhaps the most analogous work is the hybrid dynamical systems approach taken by (Aaron et al 2002). 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.

Weighting factors are needed for any 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. Neither the path planning community nor the optimal controls community addresses the selection of these weights. Typically, researchers test different weight combinations until one that produces the desired behavior is found. Since the quantities being weighted can be of different units and even different orders of magnitude, there is often no more principled technique available.

Once weights are assigned to classes of behaviors, we will need a method for intelligently selecting among them. Cognitive models offer a way of encoding complex decision-making processes, such as both weight selection and trajectory analysis. Systems like Soar (Newell 1990), EPIC (Kieras and Meyer 1997) and ACT-R (Anderson and Lebiere 1998) all try to model not only the end result of human cognition, but also the process by which those results are reached. The desired result is a planning agent that works in a “human-like” way, and can interact with humans in a familiar fashion. 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 hardware 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 (Trafton et al. 2004) which we would like to further extend into our domain in future work.

Cognitive/Physical Planner

The optimal control routine can only work with the constraints and costs it is given. Unfortunately, human goal-setters are not always proficient in translating their intentions into mathematical terms. This results in generated trajectories that are not what the user actually wanted.

The cognitive model can address this in two ways. When working with a human, it may receive feedback from him. We would like for the system to be able to accept and use the sort of critiques that humans offer naturally and easily. Once a natural language processing routine (e.g., Perzanowski, Schultz and Adams 1998) has parsed such utterances, it still remains to determine the size of the change that the user desires. Here, the cognitive model can use information gleaned from the conversation as well as features in the trajectory to estimate what increment will satisfy the user.

But this level of autonomy may be too low for all cases. If the agent is to be more autonomous, we would like for it to be able to perform a self-analysis on the generated trajectory. The optimal control routine cannot evaluate this trajectory past its cost functional; the cognitive model, however, can. It can be given a knowledge base of trajectory features and how desirable they are for different kinds of goals. Such trajectory features might include additional desired waypoints, smoothness criteria, or violations of assumed constraints that were not explicitly stated in the cost function (e.g., a time limit). The same capabilities that allow for high-level feedback from a human user can also be used for interacting with the strategic planner. The planner may be issuing very high-level descriptions of goals, and the agent needs to interpret them. When the model is enabled to make its own human-like decisions, it can decide by how much the trajectory should be altered, and inform the trajectory generator of whatever updated weights or waypoints it must consider to create a better trajectory.

Figure 2 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. The goal is to return feasible and optimal solution $X=<J_n, L_n, \underline{t}_n, \underline{x}_n, \underline{u}_n>$, where J_n and L_n summarize solution cost and the feature limits/constraints, respectively, and the set $<\underline{t}_n, \underline{x}_n, \underline{u}_n>$ specifies the full-state trajectory to be executed. ACT-R incrementally builds a history of activities $\{H\}=\{H_1, H_2, \ldots\}$ with each H_i described by an action a_i and planning state s_i. It can then use $\{H\}$ 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}*. The supporting modules *{TPLAN, FEXT, WADJ, REPAIR}* are modular and easily altered to fit into an existing problem domain. Their function is described briefly below.

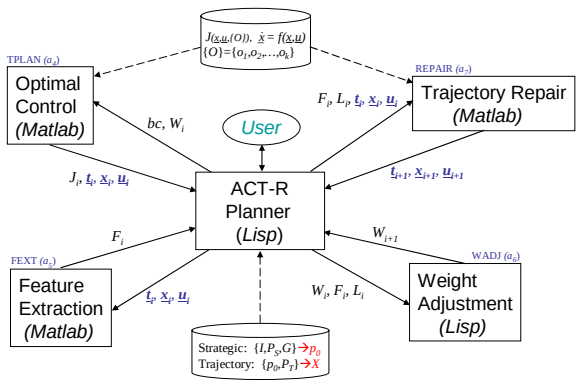


Figure 2: Details of the Cognitive/Physical Planner

Figure 3 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, F_i, and sends them to *EVAL*. If all F_i are within the limits 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 *{H}* 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 W_i, it calls *WADJ*, which uses a local rule set to decide which changes to make based on which limits were violated. These 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 *{H}* to recall the pre-*REPAIR* state of the problem and calls *WADJ* to attempt a fix instead.

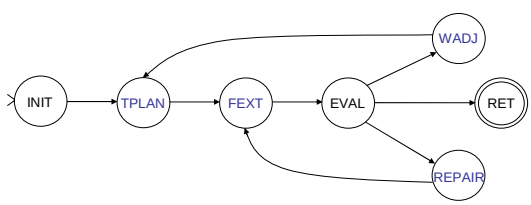


Figure 3: Trajectory Generation Process under ACT-R Supervision.

EVAL is very much the center of the ACT-R procedure. This core process begins simply, by comparing the features *F* returned by *FEXT* to the L_0 generated by *INIT*. When all F are within the bounds set by L_0, nothing more needs to be done except to return the trajectory via *RET*. When some 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 L_0 were violated and by how much? More importantly, are there conflicting 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 L_0 cannot be met. If a higher degree of autonomy has been requested, *EVAL* needs to be able to intelligently decide which 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 L_0, these are reported as well.

Overview of Optimal Control

We can use optimal control theory to solve the problem of finding an admissible input (or control) vector $\underline{u}^*(t)$ that causes a system with dynamic constraints (described by differential equations) to follow an admissible trajectory $\underline{x}^*(t)$ that minimizes a cost functional J. This is done via the calculus of variations in a process which is very analogous to minimizing a function (Kirk 1970).

A function accepts as its argument numbers – scalars, vectors, or matrices. To minimize a function, the general procedure is to take its first derivative, set that to zero, and solve the resulting equation. This finds the extrema, which can then be tested to see which are maxima and minima, and which are globally the most extreme.

A functional accepts as its argument a function. (The integral is a common functional). One can take the *variation* of a functional, which is analogous to the derivative of a function, and set it to zero to find the extrema.

We create a cost functional J, whose value depends on our agent's state, $\underline{x}(t)$, its control inputs, $\underline{u}(t)$, and time t, over the interval $[t_0, t_f]$. By taking the variation of J and setting it to zero, we can solve for the $\underline{x}(t)$, $\underline{u}(t)$, and t that will minimize J.

We can also adjoin to this cost functional a term expressing the system dynamics. When the system's dynamic equations hold, the term is zero; otherwise, the term is positive. J will not be minimal unless the term is zeroed and the dynamic constraints are not violated.

Boundary conditions depend on the particular problem, for instance if the final time is fixed or free, or if the final

state is fixed, free or constrained. In all cases, one can find *2n* equations to determine *2n* constants of integration. Typically, some of these equations deal with the starting state, and some with the ending state: this is called the split boundary value problem. The split boundary value problem is not in general solvable in closed-form, so numeric methods were employed (Shampine, Kierzenka, and Reichelt 2000).

These optimal controls methods are computationally expensive (compared to reactive controllers) and thus are typically employed offline. For a well-characterized, uncluttered environment (such as space), this is not a large drawback. For many other applications, it is not acceptable. There do exist real-time near-optimal controllers (Henshaw 2004, Miles 1997) that offer significant savings over reactive methods. As changes to the environment are detected, these methods can replan new near-optimal trajectories (path and velocity schedule sequences) from the agent's present state to its goal in a timely fashion.

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 (1) gives the cost functional *J*, and each term is described more fully below.

$$J = \int_{t_0}^{t_f} (\underline{u}^T W_1 \underline{u} + W_2 + W_3 \cdot o(r))dt \qquad (1)$$

Energy. To minimize energy use, we add the term $\underline{u}^T \cdot W_1 \cdot \underline{u}$ to the cost functional, *J*, where $\underline{u}$ is the control vector (the energy used to control the vehicle) and is W_1 a weighting term. This term is generally used in optimal controls when electric power is being used.

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

Clearance to Obstacles. To keep the vehicle away from obstacles, we add the term $W_3 \cdot o(r)$. $o(r)$ is a function which increasingly penalizes the agent as it approaches an obstacle; W_3 is the weighting term, and *r* is the distance from the vehicle to the obstacle's center. $o(r)$ is a cubic spline that is at maximum over the center of the obstacle, attains a fixed value *K* at the obstacle's edge, distance *R* from the center, and decreases to zero at some distance *LIM* away from the obstacle's edge. Coefficients a_i are computed to meet these requirements.

$$o(r) = \begin{cases} a_1(R-r)^3 + a_2(R-r)^2 + a_3(R-r) + a_4, & 0 \le r \le R \\ a_5(r-R)^3 + a_6(r-R)^2 + a_7(r-R) + a_8, & R < r \le LIM \end{cases} \qquad (2)$$

To this cost functional, we appended a simple set of dynamics:

$$\begin{bmatrix} \dot{\underline{x}}(t) \\ \ddot{\underline{x}}(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -c/m \end{bmatrix} \begin{bmatrix} \underline{x}(t) \\ \dot{\underline{x}}(t) \end{bmatrix} + \begin{bmatrix} 0 \\ \underline{u}(t) \end{bmatrix} \qquad (3)$$

which is simply "F=ma" with some losses due to friction.

We have found that the relative ratios of W_1, W_2 and W_3 to each other are significant, but their numeric values are not. (That is, <100, 10, 10> and <10, 1, 1> give the same results). The numeric value of *LIM* in $o(r)$, however, is significant. The path is actually more sensitive to changes in *LIM* than in changes in the ratio of W_3 to W_1 or W_2. In Figure 4, the band of solid lines shows the difference in path for W_3/W_1 ranging from 1 to 5. (The circular shape is an obstacle). The dashed lines show the effects of varying *LIM* from 1 (closest to the obstacle) to 7 (farthest away). Therefore, we now augment our weight set $<W_1, W_2, W_3>$ with *LIM*.

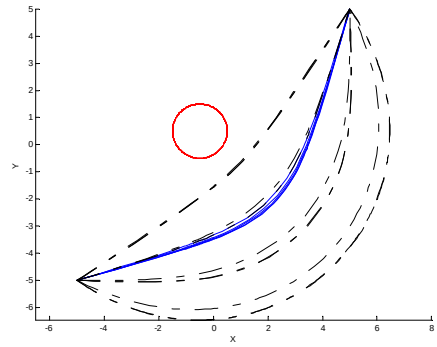


Figure 4: Effects on path of changing W_3 (solid lines) and *LIM* (dashed lines)

Results

Figure 5 shows three paths for three different weight sets $<W_1, W_2, W_3>$ (*LIM* is fixed). Each path is required to start at coordinates (-5, -5) with zero velocity and end at coordinates (5, 5), also with zero velocity. There is one obstacle that blocks a straight-line solution. Despite the very different priorities given to the cost functional terms, each path looks nearly identical.

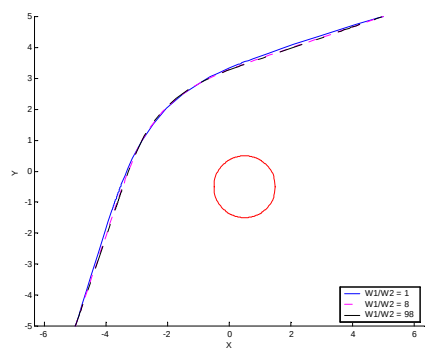


Figure 5: Paths for W_1/W_2 = 1, 8 and 98, W_2/W_3 = 1 for all.

The strength of the optimal trajectory planner is apparent when we compare the velocities and accelerations of these three cases, as shown in Figure 6. The weight set <98, 1, 1> (dashed and dotted lines) weighs saving power 98 times more heavily than saving either time or risk due to the obstacle. It takes 111.3 time units to reach its destination, using very minimal accelerations (which require only very minimal power) to maintain a very small forward velocity, as Figure 6 shows.

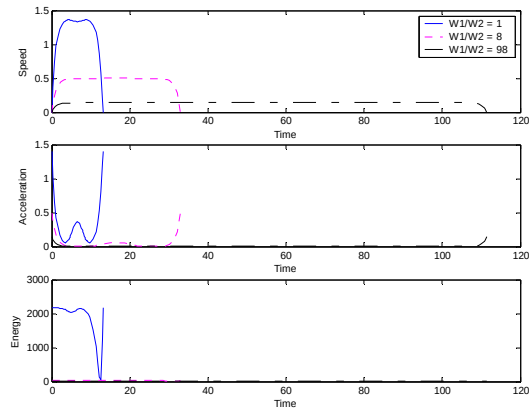


Figure 6: Speed, acceleration, and energy profiles for W_1/W_2 = 1, 8, and 98, W_2/W_3 = 1 for all.

For the weight set <8, 1, 1> (dotted lines), the agent still has a marked preference for saving power. But it is far less extreme than in the previous case, completing the mission in just under a third of the time (34.9 time units). While accelerations over most of the mission are still barely above zero, initial and final accelerations are larger, and the velocity over the mission is more than twice that of the previous case.

The trends continue as W_1 continues to decrease and W_2 and W_3 increase. The solid lines showing weight set <1, 1, 1> show markedly higher speeds, accelerations, and energy usage, with a correspondingly shorter time to finish (13.2 time units). These three optimal trajectories are obviously quite different from one another, despite the fact that the paths over which they move are nearly identical.

The path is not invariant for all combinations of weighting factors, of course. Figure 7 shows the optimal path generated for a weight set in which minimizing time is of the utmost importance. The path, which comes within 0.34 distance units of the obstacle's edge, takes only 2.6 time units to complete, but at a cost of 235.0 energy units.

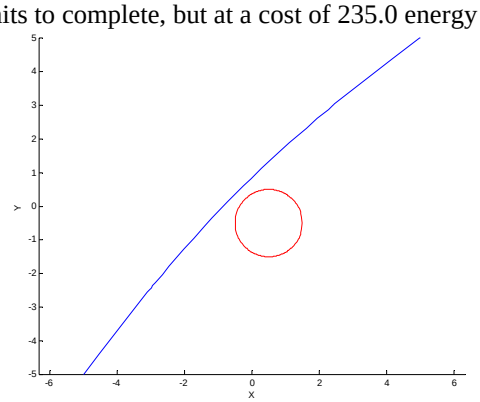


Figure 7: Optimal path for weight set <1, 98, 1>

We want to provide the cognitive agent with mathematical tools it can call upon to adjust the trajectories. Looking at Figure 6, we certainly get the sense that there are tradeoffs between energy use, time, and time-dependent quantities. Figure 4 indicates some relationship between path features and *LIM*. Can we codify these?

We can. We ran simulations for this simplified 2-DOF point robot model in worlds with no obstacles, one obstacle, and three obstacles. We extracted features from the resulting trajectories and plotted them against different weight ratios and *LIM* to look for relationships. For those features which have a time component – velocities, accelerations, forces – there were strong power relationships with the W_1/W_2 ratio that began to degrade as obstacles were introduced. Each relationship was of the form:

$$Energy = C_1 (W_1 / W_2)^{C_2} \tag{4}$$

Table 1 summarizes the properties of the trendlines of each plot. R^2 is the correlation coefficient, an indicator of how well the equation fits the actual data (1.0 is a perfect predictor).

# obstacles	0	1	3
C_1	20.19	23.14	32.00
C_2	-0.51	-0.47	-0.42
R^2	0.9996	0.7703	0.8546

Table 1: Constant values for energy curves

Adding obstacles decreases the accuracy of the power rule. However, as a rule of thumb to guide the agent in making intelligent adjustments, it may prove very useful. We used this data to provide the *WADJ* module with the following rule for adjusting W_1/W_2 when an upper limit on the energy used (u^2) is exceeded:

IF	**THEN**
u^2 limit not met	Use power rule $u^2 = C_1(W_1/W_2)^{-0.5}$
	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

Table 2: Rule for adjusting W_1/W_2 when u^2 limit not met

Similar power rules were derived for other trajectory features (see Lennon and Atkins 2004 for complete list). Path features, such as obstacle clearance, had linear relationships with *LIM*. Their rules were derived in a similar fashion.

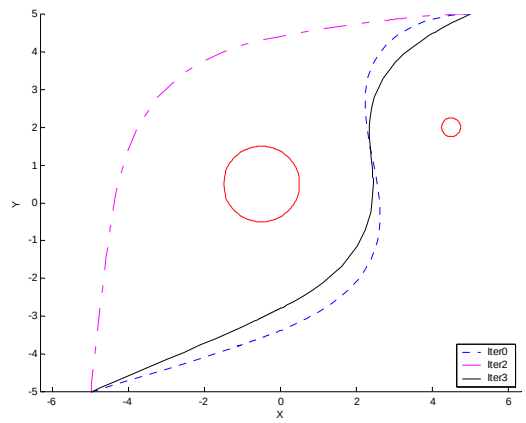
Point Robot Example

We used a 10 x 10 field with two obstacles for this example. We imposed the following limits L_0 on the trajectory:

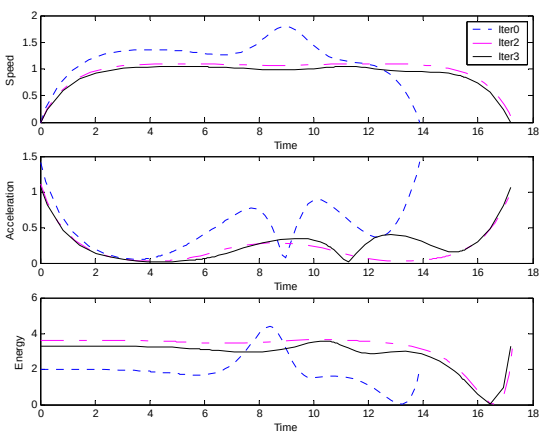
$$L_0 = \begin{cases} u_{tot}^2 \le 20 \\ \forall \{O\}, r_i \ge 1.0 \end{cases} \tag{5}$$

where u^2_{tot} is the total energy used, $\{O\}$ are the obstacles and 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 path (Figure 8, dotted line) used 26.24 energy units, violating the first constraint in L_0. *WADJ* used this feature value, the W_1/W_2 ratio and the rule shown in Table 2 to compute a new W_1/W_2 value. The new W_i were passed to *TPLAN* and the process iterated. The results are shown as the first two iterations in Table 3, after which an acceptable trajectory was found (Figure 8, dashed line).



(a)



(b)

Figure 8: (a) Path and (b) speed, acceleration and energy use for three generated trajectories

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

$$L_0 = \begin{cases} u_{tot}^2 \le 20 \\ \forall\{O\}, r_i \ge 1.0 \\ \forall\{O\} \ , min\_sep_i \le 2.0 \end{cases} \quad (6)$$

min_sep_i is the minimum separation of the agent from the obstacle along the path. We allow the agent to be farther than 2.0 units from the obstacle over the course of the path (it would be difficult to reach the endpoint, otherwise), but require that this nearest approach be less than our limit. 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. Sending the results back to *TPLAN* returned the results on the last line of Table 3. As an added and expected bonus, there is a fuel savings as well when the vehicle can more closely approach obstacles. Since L_0 does not require that the energy be near 20 units, only less than 20 units, there is no reason to recompute the trajectory. The solid line in Figure 8 shows the resulting path and trajectory information.

Iter #	W_i	Energy	Dist 1	Dist 2	W_{i+1}
0	<1, 1, 1, 3>	26.24	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*	

Table 3: Planar Robot Case Study Results. (Dist1 and Dist2 are the minimum clearance to obstacles 1 and 2);

Conclusions and Future Work

We have outlined a flexible architecture that ably combines higher-level cognitive reasoning with a computational trajectory planner. Just as human scientists and engineers use calculators and computers to augment their own cognitive powers when generating complex, optimized full-state trajectories, so too can our cognitive model call upon the computational modules *TPLAN* and *WADJ* to perform mathematical calculations. But these computational tools, while powerful, are generally insufficient. We would otherwise not need scientists and engineers to review their output. Our *EVAL* module provides the same type of high-level, intelligent evaluation of the entire trajectory that a human operator would give. As mission complexity and the number of fielded vehicles increases, it is necessary to automate this evaluation so that the humans in the loop can devote their cognitive resources to other, higher-level tasks.

Our results show that, even when paths are nearly identical, consideration of dynamics can result in very different ways of traversing those paths. An optimal trajectory planner, provided with numerical representations of symbolic goals by *EVAL*, can provide these trajectories to an agent. Cognition supports the physical motion planner by interpreting the high-level goals into these numeric values; the physical motion planner supports the cognitive model by providing a trajectory which is, ideally, the best way to achieve some goal that it has.

But the physical motion planner does not have any real understanding of the agent's goals and needs. It still falls to the agent to review the returned trajectory and check it over, ensuring that any limits that could not be communicated effectively to the trajectory planner are met. If they are not, the cognitive model has tools at its disposal

to alter the inputs to the physical motion planner to drive the solution toward one that meets these unstated requirements.

The next major step will be the improvement of the ACT-R cognitive model. In addition to the weight adjustment scheme shown here, we want it to be able to look at the returned trajectory data and make high-level alterations. The *REPAIR* module has yet to be implemented, and *EVAL* needs the ability to identify conflicting limits in L_0 and determine whether to report failure or to change the limits, based on its knowledge of the higher-level strategic goals.

Acknowledgements

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

References

Aaron, E., Sun, H., Ivancic, F., and Metaxas, D. 2002. A hybrid dynamical systems approach to intelligent low-level navigation. In Proceedings of Computer Animation 2002: 154-163.

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

Brooks, R.A. 1991 Intelligence without representation. *Artificial Intelligence Journal* 47: 139-159.

Henshaw, C.G. 2004. Personal communication.

Kieras, D., and Meyer, D.E. 1997. An overview of the epic architecture for cognition and performance with application to human-computer interaction. *Human Computer Interaction* 12: 391-438.

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

Latombe, J.C. 1991 _*Robot Motion Planning*._ Norwell, MA: Kluwer Academic Publishers.

Lennon, J.A. and Atkins, E.M. 2004. Optimal Path Planning with Behavior-Based Cost Definition. To appear in *Proceedings of the AIAA 1st Intelligent Systems Conference*, Chicago, IL, September 20-22, 2004.

Miles, D.W. 1997. *Real-Time Dynamic Trajectory Optimization with Application to Free-Flying Space Robots*. PhD thesis: Standford University.

Newell, A. 1990. _*Unified theories of cognition*._ Cambridge, MA: Harvard University Press.

Perzanowski, D., Schultz, A.C., and Adams, W. 1998. Integrating Natural Language and Gesture in a Robotics Domain. In *Proceedings of the IEEE International Symposium on Intelligent Control: ISIC/CIRA/ISAS Joint Conference*, 247-252.

Ram, A., Arkin, R., Boone, G., and Pearce, M. 1994. Using genetic algorithms to learn reactive control parameters for autonomous robotic navigation. *Adaptive Behavior* 2(3): 277-304.

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

Shampine, L.F.; Kierzenka, J.; and Reichelt, M.W. 2000. Solving boundary value problems for ordinary differential equations in Matlab with bvp4c. The MathWorks, Inc. tutorial.

Trafton, J.G.; Schultz, A.C.; Perzanowski, D.; Bugajska, M.D.; Adams, W.; Cassimatis, N.L.; and Brock, D.P. 2004. Children and robots learning to play hide and seek. *Cognitive Systems Research*, under review.

Velasquez, J.D. 1999. An emotion-based approach to robotics. In *Proceedings of the 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 235-240.