Leveraging a Cognitive Architecture to Help a Robot Walk and Chew Gum at the Same Time

Bradley J. Best, Christian Lebiere, Marc Gacy
(bbest@maad.comm, clebiere@maad.com, mgacy@maad.com)
Micro Analysis & Design, 4949 Pearl E. Circle, Suite 300
Boulder, CO 80302 USA

Abstract

Most robotic systems rely on dense, homogenous representations of the environment and use these representations to navigate and interact with the world. This requires a computationally expensive approach to solving problems in navigation. In addition, these approaches are often difficult to integrate with systems traditionally used for modeling problem-solving, planning, and higher level cognition in general. We describe the use of sparse, hierarchical representations of the environment, extracted through processing locally interesting features, to guide general navigation, coupled with dense local sampling of the environment to enable fine-grained collision avoidance. This choice results in a system capable of engaging in problem-solving activities that are only loosely coupled to interactions with objects and obstacles in the immediate environment.

Dual-Tasking for Navigation Planning and Execution of Planned Navigation

Our work integrates a planning system based on cognitive approaches to navigation with a locomotion system based on robotic approaches to locomotion. This system depends critically on the ability to sensibly interleave longer-range planning activities with more urgent locomotion activities (e.g., collision avoidance). Human capacities and limitations for interleaving motor activities with cognitive activities have been studied in great detail and have been encoded in the form of a computational theory of executive function within the ACT-R cognitive architecture for some time now (e.g., Byrne and Anderson, 1997). Although the intent of this framework is to model human cognition, the problems an effective robotic controller must face are very similar to those routinely solved by the human cognitive architecture. Therefore, we investigated the possibility that the ACT-R framework for executive control would be equally effective in providing a robust controller for a robot, and would support the interleaving of long-range planning tasks and other cognitive activities with short-range locomotion tasks.

Conventional Robotic Representations Compared to Cognitive Representations

The emphasis of this paper is on bridging the gaps between cognitive approaches to navigation and path planning with robotics approaches to the same problems. The following section illustrates some of the key assumptions and methods from each of these fields and discusses ways to merge them.

Conventional Dense Representations and Algorithms for Robotic Navigation

The conventional approach to addressing many robotics navigation problems has been to form a volumetric grid (two or three dimensional) that covers the space in which the robot will perform with a homogenous grid. This representation is supported by many robotic mapping approaches, where individual grid locations (voxels) can be represented as full (solid), empty, or unknown. Robotic navigational planning then reduces to a search problem across this homogenous grid, and is commonly achieved using an A* algorithm, D* algorithm, or a derivative of these. A unifying aspect of many of these implementations is that every voxel is equally represented, meaning that the grid is homogenous. Alternative approaches for path planning to the A* and D* algorithms, such as that presented by Burgess and Darken (2004), have kept the underlying assumption of representational homogeneity, and attempted to refine the extraction of good routes based on costs of crossing voxel boundaries. Rather than follow this approach, we have developed algorithms for navigation based on modeling human navigation, which are largely incompatible with a homogenous view of space and inherently require a different fundamental representation of space on which to operate.

Cognitive Approaches to Navigation

The representation used by conventional robotic algorithms is very unlike the representations used by human navigators as described in the spatial navigation and cognitive psychology literature. The alternative we are exploring here is employment of a representation consistent with human navigation, and the consequences of that representation on a robotic platform, so first we will attempt to describe some of the general findings from these fields, and then we will explain how that is operationalized in our system.

Human navigation is characterized by the use of a hierarchical representation of space, with little or no representation of 'uninteresting' space, and a reliance on main trunks or arteries, decision points, and a sense of the costs of various paths and the connectedness of those paths. This was demonstrated in a study of the spatial representations used by taxi drivers by Chase (1982).

Similarly, the model of the task used by Anderson, Kushmerick, and Lebiere (1993) was also specified at this level. Intuitively, this kind of representation is extremely consistent with the way we give each other driving directions – we do not specify directions at evenly spaced intervals, but rather give instructions for how to handle decision points, what to do at specific points of interest, and in general, compress the rest of the uninteresting details right out of our instructions.

By using this representation to plan a path, the human solver typically solves a much *easier* problem than the corresponding robotic system. That is, the representation used for solving the path planning problem is significantly more powerful, and as a result requires many fewer operations and much less depth of search. This is simply a result of a sparse representation which keeps branching at a minimum and tree depth of the search space extremely small. It is worth noting that this is consistent with findings of human representations in other domains. For example, Gobet and Simon (1996) demonstrated that human chess masters typically search only a few plies deep for potential moves, and only consider a few *good* moves at each ply (level of the tree). Chess playing programs such as Deep Blue, on the other hand, often fall back on the processing power of modern computers to search many orders of magnitude deeper and broader in the state space to overcome the use of a weaker representation of the space.

Although this may make it sound as if the cognitive approach is simpler, this is not necessarily the case. The main difficulty in applying a cognitive solution to the problem is in extracting or deriving the representation to be used. Again using the example of chess playing, understanding how human players only consider the *good* moves, rather than all moves, is a significant research issue, and addressing that issue in the context of robotics is a key in making progress in applying a cognitive approach to robotic navigation.

Deriving and Using a Sparse Representation of Space to Enable Cognitively Plausible Robotic Navigation

A primary goal of this work is to leverage the huge body of existing knowledge on human performance in search tasks, apply it to navigation tasks, and implement it on a robotic platform. The following section will describe how to develop a workable representation for established search techniques[1] such as means-ends analysis, hill-climbing, forward-chaining search, progressive deepening, etc., and then will demonstrate the application of one of these methods, hill climbing, to the problem domain. Figure 1 shows a line-drawing map of one of the environments used in the robotic studies as viewed in the SRI SIM robotic simulator.

[1] Workable representations in this context are assumed to be graph-based representations that can be operated on straightforwardly by graph-traversal and search algorithms.

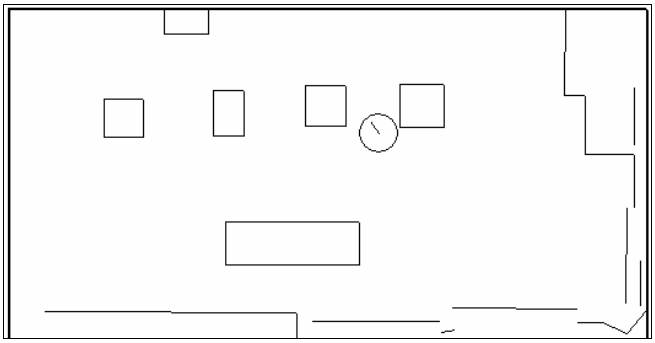

Figure 1: A Map of a *Real* Blocks World Interior Space with the Robot Navigating between Obstacles.

Deriving a Sparse Representation of Space

A real environment provides a set of challenges in representing the space, especially if the goal is to derive a sparse representation of that space using a waypoint and link representation. This requires determining the waypoints, and determining which waypoints are linked.

One way to determine waypoints is to choose points that could be expected to be of interest. This is the general approach we have chosen here. The process of waypoint identification is a feature extraction process applied to the map of the interior space, selecting points relative to corners, where corners are defined as line intersections of more than 15 degrees and less than 165 degrees. This process is analogous to feature extraction in computer vision, and as such, is a complete research topic unto itself. However, for the purposes of determining waypoints that would cover an interior space, it turns out that corners are a primary determinant of intersections, and allow for a straightforward extraction of relevant waypoints that allow for navigation throughout a space, and yet compress cognitively uninteresting portions of the space (those where there is no decision to make regarding path because there are few options).

Figure 2 displays the potential features that this algorithm picks out from a line drawing with two corners. These features represent a group of potential waypoints near a corner in the displayed map. Although we have chosen to use four potential waypoints spaced at 90 degree intervals, this choice is somewhat arbitrary, and other numbers of potential waypoints could be used.

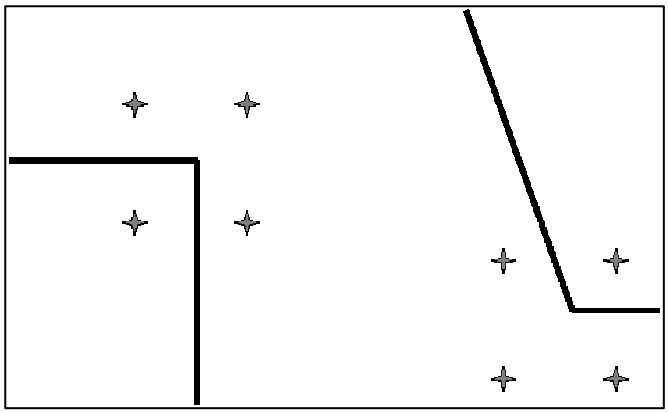

Figure 2: Potential Waypoints Extracted from Wall Corners.

This leaves us with the problem of link computation, the development of edges in the navigation graph. Again, although many complex solutions exist to problems such as this, a simple heuristic provides a workable solution. In this case, a link is initially created for any two points that have a line of sight to each other. A second stage of processing removes any link that is not more than 5% shorter than the sum of any two other connected links. This eliminates many unnecessary links and produces a very sparse representation of the space. Using the algorithm for determining potential waypoints as discussed above, and applying this method of link computation produces a relatively sparse graph of waypoints and links that covers an interior space (the results of this process are displayed in Figure 3 for a real internal space).

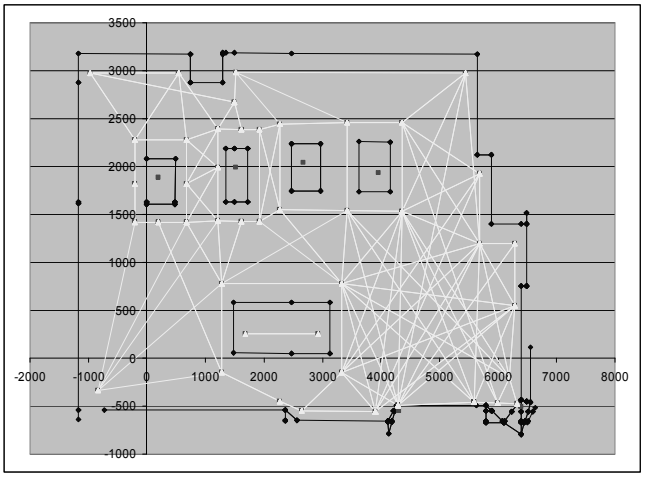

Figure 3: A Sparse Graph of Waypoints and Links Covering an Interior Space.

This representation is an undirected graph. As such, solving navigation problems is equivalent to other state space search problems including broad categories such as planning, and individual problems such as the Traveling Salesperson Problem. There is a long tradition in cognitive science of producing algorithms that solve these problems by traversing the graph (e.g., Michie and Fleming, 1968). Similarly, many problems described by Newell and Simon (1973) use an equivalent representation.

Applying a Cognitive Problem-Space Method to a Sparse Spatial Representation

A fundamental approach for studying human problem-solving behavior, proposed by Newell and Simon (1973), is to conceptualize human behavior as a search through a problem space accomplished through the application of various operators. This search can either be a planning search, in which potential operations are proposed with a complete solution developed prior to taking any moves in the actual environment, or an online search, in which the problem-solver applies the operators in the external environment and reasons from their actual outcomes.

Algorithmically, these alternatives are largely indistinguishable. Practically, however, they differ, since the cost of backtracking from a dead-end may be significantly higher in an actual environment than if the operations are just tested out on an internal (mental) representation of the problem. Human problem solvers often interleave both methods of problem solution: at times they simply act on the external environment, while at other times they pause to consider the outcome of several possible combinations of steps. This distinction is often imposed by the task, where a game such as chess strongly encourages consideration of several moves in the future, while real-time video games strongly discourage focusing on any steps other than the immediate one. Clearly, modeling human planning behavior requires the ability to achieve both of these types of search within the same framework, and in a way where they can be interleaved as needed. The approach described here will demonstrate that ability across a variety of navigational behavior. In particular, we will detail how we have used the representation described above in two distinct cognitive models of navigation both developed within the ACT-R cognitive architecture. The first model is a reinforcement-learning model of high-level navigational behavior that learns to associate paths to a goal with the likelihood of exposure to a threat, and subsequently chooses to avoid the high-threat paths. The second model is a planning model of high-level navigational behavior.

Learning High-Level Navigation

To approach (or even exceed) the performance of a human expert in a domain like spatial navigation, three basic approaches are possible. One is the algorithmic approach that attempts to solve the problem in a systematic fashion. However, that approach sometimes requires very high computational costs to achieve its objective of optimal or near-optimal performance. It is also unable to adjust to new or unforeseen situations. A second approach is knowledge engineering. One can attempt to extract from human experts knowledge about how they perform the task and codify it into a computational system (e.g. expert or rule-

based system). However, knowledge engineering is a notoriously difficult and time-consuming task, even more so when the task involves implicit skills and knowledge such as those used in spatial navigation. Moreover, that system will not be able to adapt itself to situations that have not already been coded in its knowledge base. A third approach is to have the system learn, itself, the knowledge of how to perform the task. While this approach does not carry the guarantees of an algorithmic approach and might be more computation-intensive than a knowledge engineering approach, it has a number of advantages. While it requires significant computation to accumulate its expertise, afterwards it is not nearly as computationally intensive as algorithmic methods because its expertise is essentially stored computation that algorithmic methods have to re-create for every problem. While it requires some engineering up front to specify the problem representation, it does not require the intensive knowledge extraction effort of traditional knowledge engineering approaches because it does not specify the content, merely the form of the knowledge. And its principal advantage might be that the system is constantly learning and optimizing itself to its environment, by which it can adapt and improve its performance when faced with new, unforeseen situations. It should be emphasized that learning can take many forms. A model can learn from being given a demonstration or explicit instructions from a human expert. It can learn by performing the task with a human expert, in this case in an adversarial role. Alternatively, it can learn from its own performance in exploring the environment. We will focus on the latter in this application but the other sources of learning would use the same mechanisms in similar ways.

The choice of a proper high-level representation is critical to the performance characteristics of a model. The ACT-R architecture provides some relief from the demands of this task by constraining the choice of representation in a way that is both computationally tractable and cognitively plausible. That is, the constraints of the architecture guide encourage some representations while discouraging others. The representations that are encouraged are cognitive representations, and as such intuition and introspection can provide significant leverage in choosing the representation.

When considering possible intermediate points on the way to the goal, a number of attributes will have an impact of the decision. One attribute is the distance from the current position. Larger distances lead to more exposure and should be discouraged. Another attribute is the size of the associated cluster of obstacles. A larger cluster is desirable because it provides better protection. A third attribute is the direction from the current position. That direction should be in the general direction of the goal, otherwise excessive detours might result. All these attributes are not symbolic but instead vary over continuous ranges of distances, sizes and directions. In ACT-R, information is represented in the form of a chunk, which is a typed, named collection of a small number of fields. Thus, a sample cluster to consider as an intermediate waypoint would be:

Cluster57
    isa cluster
    position Position34
    size 2.5
    distance 1.3
    direction 35.3
    tested Not

The "isa" field indicates the type of the chunk. The second field gives the position of the cluster, which is itself a chunk constituted of its Cartesian coordinates. Note that the name of a chunk, e.g. Cluster57 or Position34, is irrelevant and that what matters is the content of its slots. The final slot indicates whether that cluster has already been considered in this round as an intermediate destination.

Based on this representation, a baseline performance is needed for learning to be effective. One basic property of ACT-R is its ability to retrieve chunks that provide the closest match to a desired pattern. Therefore, the default navigation method is to attempt to retrieve the intermediate cluster that provides maximum size, minimum distance and the same direction as the final destination. The best compromise among possible intermediate destinations will be chosen according to the partial matching equation:

$$M_i = A_i - MP \cdot \sum_{matches} Sim_{vd}$$

It specifies that the match score of chunk i is its activation A_i minus the sum, scaled by a mismatch penalty factor MP, of all matches of the similarity between the desired value d requested by the match pattern and the actual value v present in the chunk. While all the fields matched in this case (i.e. the size, distance and direction) have real values, similarities can be defined between symbolic chunks as well, and will reflect the similarities between the contents of the chunks. For instance, similarities are defined between positions to reflect the value of their fields (i.e. the Cartesian coordinates of the position).

Given this baseline competency, the model will select the next intermediate point that best satisfies the constraints of size, distance and direction, then move to that intermediate destination, scan the landscape for the next possible intermediate destination, then repeat the process until it reaches destination or encounters failure. While this hill-climbing procedure is fairly robust and effective, it tends to fail when the early choices lead to dead-ends from which only poor choices can be made going forward. That is when learning comes in. The ACT-R architecture automatically adds any completed goal as a chunk in long-term memory. The top-level goal for this model is of type 'navigate'. This type holds, for each decision step, the cluster chunk that was chosen as the next intermediate destination as well as the outcome of that decision. Therefore, learning will build

upon the basic competency described above as follows. Once a cluster has been selected as the best compromise of those not yet considered, memory is prompted for any experience with a similar choice. If no experience exists or if it has been positive, then that intermediate destination is selected as it would have been, and the move is made. Otherwise (if a negative outcome resulted from a similar past choice) that move is marked as considered and another intermediate destination is chosen among those not yet considered. In that way, learning helps the model improve its performance by avoiding what have been poor choices in the past. This process is illustrated by the difference in the first and second iterations through the map in Figure 4.

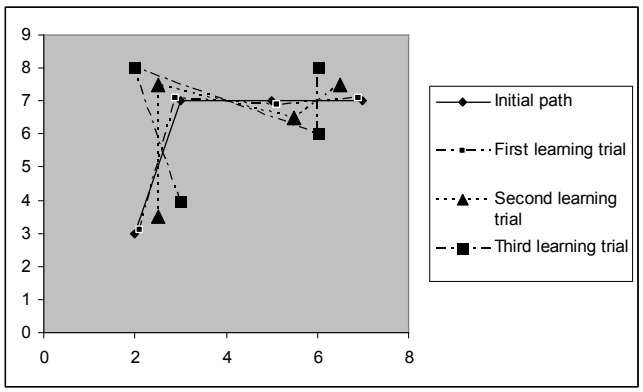

Figure 4: Path Adjustments Due to Learning.

On its first traversal of the map, the model chooses the lower path because its first move is most promising. But then it faces a long move to the final destination without any protection, producing a negative outcome. On its second traversal of the map, it still considers first the move leading to the lower path, then remembers the negative outcome and considers other intermediate destinations. Of those, it selects the first move of the higher path, which affords better subsequent protection and ultimately leads to success. On subsequent traversals of the map, the model will recall those decisions with positive outcomes and follow the higher path consistently.

If the model depended upon the map being always the same, this would be an interesting but ultimately very limited process. Even if we operate in the same environment that we have encountered before, a number of specific details tend to change unpredictably, such as small positional changes of movable items (e.g. furniture) or disappearance of some landmarks. Moreover, we want learning to generalize to new situations, which will never be identical to those that we have encountered previously. Fortunately, the same partial matching mechanism used for the base performance can also be used to generalize the learning of past outcomes. When an intermediate destination choice is compared to past outcomes, it is compared not just identical past choices but all similar ones as well. That process is based upon the similarities among clusters and positions. As we mentioned before, similarities can be defined not only between continuous value ranges but between structured chunks as well on the basis of the similarities between their constituent fields. Thus one can generate past learning to situations involved similar clusters or positions. We demonstrate that capability with the last three paths through the map. Each configuration for those paths is similar to the original one, but the position of each cluster has been increasingly altered for each following path. One can see in Figure 4 that the model will successfully generalize and find the right path for the third and fourth iteration, until in the fifth iteration the landscape has been so altered that the position of two competing choices have been reversed. At this point, the model will make the other choice, realizes that it is now in a new situation, and fall back to the default competency to complete its path successfully.

A number of issues remain in completing a fully satisfactory model of learning in this domain. One is the issue of feedback delay and probability. This model assumes that the outcome of a move is always immediately available. In practice, knowledge of the outcome is sometimes delayed, perhaps until the conclusion of the process, or it may not be given deterministically but instead may be the result of a stochastic process that obscures the quality of choices. The mechanisms in ACT-R responsible for the learning process described here have been shown to perform successfully in such environments, albeit less efficiently (e.g. Sanner, Anderson, Lebiere & Lovett, 2000; Gonzalez, Lerch, & Lebiere, 2003). Another issue is further learning beyond an explicit storage of the quality of choices. The model as described here still has to consider all choices and their outcome. One would expect a human expert to immediately recognize the situation and generate the right decision without having to consider and rule out other alternatives. Such learning is possible using the proceduralization process in ACT-R. That mechanism would create new production rules that would propose moves that have been successful in the past and preempt the more general production rules for the consideration and evaluation of candidate moves as described moves. Those production rules would have associated with them a utility that would be learned to reflect the final outcome. Production rules that propose consistently poor moves would see their utility downgraded below that of the production rules for the general process and would stop being selected.

Developing a Plan to Reach a Target

An alternative to the approach defined above in the learning model is to form a plan in advance and use that as the basis for traveling a particular path. This is commonly achieved in robotic environments with a search algorithm such as the A* algorithm that heuristically searches a very large space of possible paths. Achieving the equivalent formulation of a plan within a cognitive framework, however, is more difficult due to the constraints and limitations of cognition imposed by a cognitive architecture. Since a plan that can be assembled and maintained in memory is, by necessity of

memory limitations, simple in comparison to a typical plan generated by an A* search algorithm, it can only operate on a fairly simple, or sparse representation. Exactly such a representation was described above, and we have used that representation to demonstrate the capability to generate and execute a navigational plan while respecting the limitations on cognition imposed by the ACT-R architecture.

The search method we developed is a variant of progressive deepening (Chase and Simon, 1973; Gobet, 1997; Gobet and Simon, 1996), and proceeds by developing a search path along the undirected graph describing the navigational space. The algorithm starts with a plan consisting of the origin and a goal to reach a particular destination. The search is then developed heuristically at each node in the search path by adding the edge that leads most directly towards the destination. In most cases in the spaces we have worked with, this does, in fact, provide a navigable path to the destination. In cases where the search results in a dead-end, the dead-end choice is noted, and the search is resumed again from the *root* of the tree, retrieving each link in the previous plan, and potentially exploiting a different branch in the tree at each step. This exploitation is probabilistic (and weighted according to expected path utility based on the directness of the alternative in relation to the destination) and may not happen at any node until the final node. If the search does reach the final (successful) node in the plan without choosing an alternative, then an alternative path is chosen from that final node (the node where the dead-end path was chosen from).

This is a simple search algorithm, and as a result it is compatible with real-time path discovery, where the robot actually traverses the path, as well as with the pre-planning version described here. The major shortcoming in this method is that it does not take advantage of the hierarchical structure of the search graph. For example, the search graph may have points at which a node attaches to a sub-graph that has no connection to the rest of the search graph. If the destination is not within the sub-graph, there is no need to search any of it. Although this can only be known if the search algorithm has an understanding of the search space, and the search algorithm cannot in general expect to have that knowledge, human navigation takes advantage of vision to uncover exactly that information prior to moving into an unknown area in search of a path to a goal. For example, a human driver would likely not drive into a box canyon looking for a path through it, and would instead perceive the potential dead-ends waiting there and choose a different route. Thus, the next challenge for extending this algorithm is to integrate the visibility and connectedness of the whole visible portion of the navigation graph.

Obstacle Detection and Collision Avoidance

The previous sections have dealt with the high-level cognitive issues in planning. The approach described here follows that of many planning systems (e.g., the GraphPlan system of Weld, Anderson, and Smith, 1998) which use a symbolic representation of the operators and transitions available as the basis for successful planning. In a robotic environment, however, these symbolic decision choices must be resolved in an environment characterized by real-time, continuous values. In particular, perception in these environments fundamentally consists of continuous (real-valued) sensor readings, while action consists of real-valued effector outputs. Obstacle avoidance is not the process of deciding whether or not an obstacle exists, but instead is the process of deciding how to control the robot's motion in a way that avoids hitting the inevitable obstacles. This section will deal with these issues separately, while a later section will discuss the integration of obstacle detection and collision avoidance with real-time planning and cognition.

The basis for robot perception and action is through the implementation of a set of sensors and effectors. Through the sensors, the robot perceives the shape of its surroundings, obstacles, entities, and its location. In the case of the ActivMedia Pioneer 3DX robot configuration used for this project, the sensors include sonar, ladar, contact bumpers, a video camera, a compass and gyros (as well as various actuator sensors). The effectors, or action outputs, on the system include the wheel motors, a speaker, and pan-tilt-zoom camera controls.

The ActivMedia software provides APIs that allow for access to low-level raw sensor data and high-level processed sensor data. Similarly, actions can be specified through the APIs as either high-level actions, such as move to an (x,y) position, or low-level actions such as changes in wheel motor velocity. The high-level action APIs permit a straightforward mapping of the synthetic agent API developed for ACT-R Unreal Tournament bots (Best and Lebiere, 2003) directly onto the ActivMedia APIs, providing substantial code reuse.

In addition to the real robotic environment, the ActivMedia software package includes a simulation environment for the robot to allow faster testing and prototyping of control code without the need to run the real robot. The simulated ActivMedia platform provides reasonably high fidelity and includes aspects of the real platform such as sensor noise and wheel slippage (Figure 1 is a screenshot of this simulator during a model run).

This work follows the methodology used by Best and Lebiere (2003) for embedding ACT-R agents in the Unreal Tournament environment. Location updates are provided to the agent at 100ms intervals, in this case through a localization algorithm provided with the platform in conjunction with its sensor data. The location information is then processed by a line-of-sight algorithm that determines the surfaces that are visible from the robot's location, and a representation of these visible surfaces is provided to the agent. In other words, the agent is given a real-time representation of the visible obstacles in its environment from its vantage point.

Using this representation, the robot can be directed to move to any visible point (it is also possible to direct the robot to an invisible location, but this is handled by this platform as a navigation task). The robotic agent uses its

knowledge of the currently visible obstacles in the environment to steer away from those obstacles while trying to reach the visible destination. This is accomplished through a simple ruleset that implements sensors which are sensitive to collisions. For example, if an object is close by on the left-front of the robot, this matches the conditions of a production that steers the robot slightly to the right. The complete set of rules for locomotion and obstacle avoidance consists of only seventeen rules. These rules include rules for accepting a goal to move to a location, managing the speed and direction of the movement, and halting the motion when the target is reached.

Although this system generates collision data from intersections with a line map of the environment, this is functionally equivalent to online range sensor data. This is because the line intersection is calculated relative to the robot location, providing a sensor distance along that line to the nearest obstacle. Real-time laser range data provides the exact same information for a range of angles (360 samples across the range of the SICK laser). The calculation of line intersection differs primarily by being insensitive to noise. This is due to the incorporation of all sensor readings in the localization process. It appears that the same sensor information for obstacle avoidance could be computed in real-time from the sensors (and, in fact, this is exactly what is done in the standard ActivMedia software package).

It is worth noting that actions in this system occur across time. A movement is not instantaneous, an adjustment in course takes time, and sensing is, on average 50ms behind ground truth (half of the cycle time of perception). Despite this, the small adjustments made by the system tend to be averaged across by the physical system. That is, if a contradictory adjustment is made – one that is inconsistent with the majority of the adjustments – it is almost inevitably countered by a subsequent adjustment before it can actually take full effect. This combination of discrete decisions and continuous effects may provide extra robustness for the system, though this is only conjecture at this point and is in need of further exploration.

Stratified Behavior Architectures

The subsumption architecture, described by Brooks (1991), attempts to integrate behavior in a robotic environment in a sensor-driven, stratified way where more complex cognition arises through the interaction of sensors and very simple goals which trickle down through the architecture and may be modified by knowledge encoded in the sensors and effectors at the bottom of the pyramid.

Although the original subsumption architecture was specifically non-symbolic, the idea of stratified control of robotic behavior – placing reactive sensor-driven components at one pole of the architecture and goal-directed behavior at the other– has been subsequently incorporated into several symbolic models of robotic control.

This type of stratified control architecture, as exemplified by the 4DRCS system (Albus and Meystel, 2001), has many interesting properties and corresponding issues, among which the integration of reactive, or bottom-up, processing with goal-driven, or top-down, processing is perhaps the most fundamental issue that any complex robotic system must address. In systems employing this kind of architecture, it is possible for a low level sensor to impede a high-level cognitive goal, with the intent of preventing damage to the robot, and the imposition of a kind of physical world common sense onto the cognitive system. This has met with some success, but an issue arises when the sensor impedes the desired behavior without notifying the cognitive level about exactly what the problem is. That is, the sensors can take over the overall system behavior, and managing this propensity can be tricky at best.

This is not to say that the goal of integrating sensor driven behavior and cognitive behavior is either undesirable or impossible. However, it appears to call for considerable involvement from the cognitive level when problems do occur. If the results of DARPA's Grand Robotic Challenge are any indication of the state of the art (none of the robots completed more than a small portion of the course, and the majority encountered some obstacle that they could not reason their way around), then we can assume that this is one of the primary areas of difficult in robotic navigation. The integration of goal-directed behavior with reactive behavior can thus be seen as one of the main research challenges in developing autonomous robotic systems.

Implementing a Stratified Behavior Architecture in a Production System

Our approach to addressing the sensible integration of sensor-driven reactive behavior and goal-driven behavior is to attempt to abstract the fundamental operations of a stratified robotic control architecture, but within the ACT-R cognitive system, and according to cognitive constraints. We have done this by implementing procedural rules related to sensor outputs that are goal-free. In this implementation, production rules have replaced sensor triggered functions and the weighting of different behaviors is achieved through the ACT-R conflict-resolution system, providing the cognitive system with a means of control over reactive behavior, and the capability to learn patterns for effectively managing behavioral selection.

During each decision cycle the ACT-R navigation models described here attempt to match elements of procedural memory to the current goal and context. They system may match a goal-driven production that involves the next step in planning, or in performing some cognitive activity. However, it may also match a reactive goal-free production that specifies how to deal with an immediate situation. The conflict resolution process of the production matcher in ACT-R then arbitrates among these potentially conflicting options for behavior, thus centralizing that decision. In addition, an extension of partial matching to spatial conditions has enabled the coverage of a huge range of possibilities with a very limited set of productions. The production below will be used to detail these capacities:

```
(p face-move-direction
   =move>
     isa move-to-location
     location =location
     - initiated t
     start-time nil
     - location-ego-angle 0
==>
   !bind! =time (game-time (get-bot-
object))
   =move>
     start-time =time
   !eval! (orient-to-point *bot-
platform* =location)
   !eval! (format t "~%Turning toward
point ~s~%" =location))
```

This production, paraphrased in English, specifies that if the robot is tasked with a move to a location that has not been initiated, and it is not facing the location, it should turn toward it. That is, the robot should face where it is going before it tries to go there.

The line that specifies that the relative angle to the location is not zero (- location-ego-angle 0) would be problematic in a system that did not use partial matching. A potential knowledge-engineering approach to this problem in a rule-based system might be instead to specify a range of possible relative angles, for example, -15 degrees to 15 degrees, over which the rule applies. However, authoring knowledge in this way requires significant bookkeeping on the part of the knowledge engineer. The partial matching mechanism described above overcomes this by allowing the programmer to specify a central tendency of the production, or a prototypical case, without having to specify or even know the boundary conditions, which can be determined by the conflict resolution mechanism.

Integrating Path Planning and Collision Avoidance with a Cognitive Architecture

The implementation of a modular system for locomotion and obstacle avoidance is consistent with the buffer paradigm introduced with ACT-R 5.0, in which modules external to the central executive may exhibit both asynchronous and synchronous behavior through a principled, low bandwidth connection. This allows the interleaving of cognitive actions and reactive actions within the same system. Theoretically, this places constraints on how much reasoning the robotic agent can do while it is also navigating (i.e., this is a dual-task implementation where it is quite possible to have interference effects). The practical benefit to this organization is that the high-level and low-level knowledge is accessible within the same code-base, and managed through the same system for selecting important actions as selecting important goals.

The integration of path planning and collision avoidance is largely accomplished by the architecture. Though this may seem intellectually unsatisfying, if a system is defined in this way within the ACT-R architecture, the integration has, for most practical purposes, already been accomplished. The interleaving of cognitive (navigation) and reactive (locomotion) behaviors is accomplished through the ACT-R conflict resolution mechanism. When a collision avoidance production matches the current situation exactly, it tends to predominate over other productions (this is a tendency rather than an absolute distinction because of the stochastic nature of the ACT-R system). Less exact matches result in a lower expected utility for the production, and therefore produce behavior with less urgency.

Since an urgent situation will have a chance to capture the attention of the system at every cycle, it is unlikely to go unnoticed for long. The following segment is an annotated trace of a robot run excerpt detailing the interleaving of productions that occur during the operation of the path-learning model described above. The action is picked up at the end of traversing one of the path segments shown in figure 3 with the action of the robot explained for each individual cycle of the excerpt.

```
Time 19.101: Intermediate-Move Selected
MOVING TO 2270.0 1550.0 OUTCOME:
NEGATIVE
Time 19.301: Intermediate-Move Fired
```

At this point, the robot has noted that the waypoint it has just reached, which is an intermediate point along the way to the destination, is exposed to the threat, and the robot notes that the outcome is negative for the move. As part of the learning, however, it will continue exploring the path (a path with some exposure might still be the best path), and so selects a new waypoint to travel toward.

```
Time 19.301: Face-Move-Direction
Selected
Turning toward point (2270.0 1550.0 0)
Time 19.501: Face-Move-Direction Fired
```

The robot is now in the process of turning to face the direction of the next waypoint. The production "Face-Move-Direction" has no goal, but only matches if a move is in progress and the destination is *not* in front of the robot.

```
Time 19.501: Move-To-Location Selected
Moving to point (2270.0 1550.0 0)
  Time 19.701: Move-To-Location Fired
```

The production "Move-To-Location" is only applicable if the target is in front of the robot, so this means the robot is (approximately) facing the move destination.

```
Time 19.701: Cognitive-Operation
Selected
Time 19.901: Cognitive-Operation Fired
```

We gave the path-learning robot a generic goal to complete a cognitive task to demonstrate the capacity to

interleave cognitive and reactive behavior. This particular production, "Cognitive-Operation", is a production that is masked by urgent tasks. In this case, the robot has turned to its next waypoint, and has started moving in that direction. At this point, it is free to "walk and chew gum", and does exactly that. The locomotion is continuing, but does not require attention to make it continue, so the cognitive system is free to do what it will.

```
Time 19.901: Move-To-Location-Indirect
Selected
Avoiding obstacle between current
location (731,445,0) and target
(2270.0, 1550.0, 0) by moving straight
ahead to (2342, 1442, 0)
Time 20.101: Move-To-Location-Indirect
Fired
```

At this point, the robot has noticed an obstacle along its path. This type of event can be expected to happen with regularity in a robotic environment: though the path was clear, the robot drifts slightly, and that drift requires attention to get back on course.

```
Time 20.101: Move-To-Location-
Impossible-Right-Obstacle-Detected
Selected
Obstacle detected on right-front, clear
to left, so turning left toward (2342,
1442, 0), setting speed to creep
Time 20.301: Move-To-Location-
Impossible-Right-Obstacle-Detected
Fired
```

Here the robot has engaged in a simple reactive behavior – there is an obstacle to one side, and no impediment to the other, so the robot slows and curves away from the obstacle.

```
Time 20.301: Move-To-Location-
Incomplete-Path-Clear-Facing-
Target-Full-Speed Selected
Direct path clear now and facing
target, so moving directly to
(2270.0, 1550.0, 0)
Time 20.501: Move-To-Location-
Incomplete-Path-Clear-Facing-
Target-Full-Speed Fired
```

The simple remedy of swerving slightly was sufficient to solve the problem, and the robot now continues at full speed towards the next waypoint,

This example demonstrates the interleaving of reactive and cognitive behavior achieved by leveraging a cognitive architecture. Although these reactive and cognitive behaviors occur at conceptually different levels, the implementation of these different abilities is through procedural knowledge at the same level, and the production conflict resolution mechanism of ACT-R specifies a-priori how they interact within a unified framework.

Despite the apparent simplicity of the rules described to constrain the behavior of the locomotion system, extremely complex behavior arises from these rules. This reflects the seminal work of Simon (1979), and also shares a theme with the robotic vehicles proposed by Braitenberg (1984) that exhibit behaviors whose complexity is a property of the environment, not the rules that govern them.

Conclusions and Future Research

Given the current research landscape in robotics and cognitive modeling, a mismatch exists between the two. Our work takes steps towards bridging that gap. Table 1 presents fundamental differences between the approaches.

Table 1: Comparison of Robotic and Cognitive Navigation.

Issue	Robotic	Cognitive
Space Representation	Homogenous and Complete	Hierarchical and Sparse
Feature extraction	Cheap	Expensive
Path Planning	Deep, broad search	Shallow, narrow search
Search Algorithms	A*, D*	Means-ends analysis, hill-climbing, progressive deepening
Computational cost of Path Planning	Expensive	Cheap
Cost of integration with problem-solving models	High	Low

Note that although the table suggests that the cognitive approach to path planning is computationally cheap, it is the algorithm combined with the representation that makes it so. The algorithm itself is expensive – a method like progressive deepening is more expensive than an equivalent search that avoids revisiting nodes. The reduction in overall computational cost comes through a dramatic reduction of the problem space. The majority of the computational work is avoided, and the cognitive solution is cheap relative to the robotic solution, though the algorithm is not so.

In fact, a general lesson from cognitive modeling is that computationally demanding processes are often unnecessary for leveraging cognitive representations, and much of the computational work is done in generating the representation rather than using it (e.g., Best, 2004). Human solvers often apply an inefficient algorithm to a more powerful representation. In addition to leveraging a more powerful, more compact representation, similarity-based matching allows for selecting the *right* choices for comparison, further narrowing the range of comparisons needed.

This approach depends critically on leveraging perceptual processes to accomplish feature extraction of only important features. The analogy in common navigational terms is driving, where we rarely pay attention to navigational issues on long empty stretches of open highway, and rarely represent them when giving directions. In this case, we use perception to filter the environment, and we rely on a sparse representation based on intersections and decision points to both navigate and communicate about navigation. Extracting a cognitive representation of space from a raw representation still poses a serious challenge. Our approach is a first step that leaves considerable work to be done. For example, though feature extraction and link determination was straightforward in an interior environment, it is unlikely to be so in an exterior environment.

This work can be seen as a demonstration of the leverage provided by a cognitive architecture in integrating mechanisms. Though a mechanism implemented in a cognitive architecture is not likely to be optimal, the principled combination of mechanisms within the architecture in this case produces robust behavior with minimal calculation. The combination of locomotion and navigation was accomplished painlessly.

Finally, the use of a common platform for robotic and virtual ACT-R agents allows mixing humans and virtual agents in real and robotic environments, enabling comparison of robotic and human performance on identical tasks, examination of teamwork between robotic entities and humans, and the extension of cognitive modeling to a real-time, physical, three-dimensional world.

Acknowledgments

The bulk of this work was funded through the DARPA Architectures for Cognitive Information Processing (ACIP) program. In addition, a portion of this work was funded through an Army Research Lab Director's Research Initiative grant. In particular, we would like to thank Troy Kelley for supporting this research.

References

Albus, J. and Meystel, A. (2001). *Engineering of Mind*, John Wiley & Sons, Inc.

Anderson, J. R., Kushmerick, N., and Lebiere, C. (1993). Navigation and conflict resolution. In J. R. Anderson (Ed.), *Rules of the Mind*. Hillsdale, NJ: Erlbaum.

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

Best, B. J. (2004). *Modeling Human Performance on the Traveling Salesperson Problem: Empirical Studies and Computational Simulations*. Doctoral dissertation, Department of Psychology, Carnegie Mellon University, Pittsburgh, PA.

Best, B. J., & Lebiere, C. (in press). Cognitive Agents Interacting in Real and Virtual Worlds. In *Cognition and Multi-agent Interaction: From Cognitive Modeling to Social Simulation*, R. Sun (Ed.), Cambridge, UK: Cambridge University Press.

Best, B. J. & Lebiere, C. (2003). Teamwork, Communication, and Planning in ACT-R Agents Engaging in Urban Combat in Virtual Environments, In *Proceedings of the 2003 IJCAI Workshop on Cognitive Modeling of Agents and Multi-Agent Interactions*

Best, B. J., and Simon, H. A. (2000). Simulating Performance on the Traveling Salesman Problem. *Proceedings of the 2000 International Conference on Cognitive Modeling*.

Braitenberg, V. (1984). Vehicles: Experiments in Synthetic Psychology. Cambridge, Massachusetts: MIT Press.

Brooks, R. A. (1991). Intelligence without reason, In *Proceedings of the 1991 International Joint Conference on Artificial Intelligence*.

Burgess, R. and Darken, C. (2004). "Realistic Human Path Planning using Fluid Simulation", Proceedings of *2004 Conference on Behavior Representation in Modeling and Simulation*.

Byrne, M. D. & Anderson J. A. (1997) Enhancing ACT-R's perceptual-motor abilites. In Proceedings of the Nineteenth Annual Conference of the Cognitive Science Society. Stanford University, p. 880. Mahwah, NJ: Erlbaum.

Chase, W. G. (1982). Spatial representations of taxi drivers. In D. R. Rogers and J. A. Sloboda, eds., *Acquisition of Symbolic Skills*. Plenum Press, New York.

Chase, W. G., and Simon, H. A. (1973). The mind's eye in chess. In W. G. Chase (Ed.), *Visual information processing*. New York: Academic Press.

Compton, B. J., and Logan, G. D. (1993). Evaluating a computational model of pereptual grouping by proximity. *Perception & Psychophysics*, 53(4).

Gobet, F. (1997). A pattern-recognition theory of search in expert problem solving. *Thinking and Reasoning*, 3.

Gobet, F., and Simon, H. A. (1996). The roles of recognition processes and look-ahead search in time-constrained expert problem solving: Evidence from grandmaster level chess. *Psychological Science*, 7.

Gonzalez, C., Lerch, F. J., Lebiere, C. (2003). Instance-Based Learning in Dynamic Decision Making. Cognitive Science, 27.

Michie, D., Fleming, J. G., and Oldfield, J. V. (1968). A comparison of heuristic, interactive and unaided methods of solving a shortest-route problem. *Machine Intelligence*, 3.

Newell, A., & Simon, H. A. (1972*). Human problem solving*. Englewood Cliffs, NJ: Prentice-Hall.

Sanner, S., Anderson, J. R., Lebiere, C., & Lovett, M. (2000). Achieving efficient and cognitively plausible learning in backgammon. In Proceedings of the Seventeenth International Conference on Machine Learning. San Francisco: Morgan Kaufmann.

Simon, H. A. (1979). The sciences of the artificial. Cambridge, MA: MIT Press.

Weld, D. S., Anderson, C. R., and Smith, D. E. (1998). Extending graphplan to handle uncertainty and sensing actions. In *Proceedings of the Fifteenth National Conference on Articial Intelligence*, Menlo Park, CA: AAAI Press.