Integrating Action and Reasoning through Simulation

Samuel Wintermute

University of Michigan
2260 Hayward St.
Ann Arbor, MI 48109-2121
swinterm@umich.edu

Abstract

This paper presents an approach for integrating action in the world with general symbolic reasoning. Instead of working with task-specific symbolic abstractions of continuous space, our system mediates action through a simple spatial representation. Low-level action controllers work in the context of this representation, and a high-level symbolic system has access to it. By allowing actions to be spatially simulated, general reasoning about action is possible. Only very simple task-independent symbolic abstractions of space are necessary, and controllers can be used without the need for symbolic characterization of their behavior. We draw parallels between this system and a modern robotic motion planning algorithm, RRT. This algorithm is instantiated in our system, and serves as a case study showing how the architecture can effectively address real robotics problems.

Introduction

It has been argued by many that low-level motor control and perception are critically important for intelligent behavior in embodied agents (e.g., Brooks, 1991). These views are often put in contrast with more traditional views of intelligence that emphasize symbolic processing (e.g., Newell, 1992). To date, adherents on either side haven't built robots that have exhibited anything close to general intelligence. Solely symbolic robots are rigid, and unable to deal with the subtlety of the real world, where every relevant perception needed to control action doesn't directly map onto a symbol. Robots without a symbolic level, however, aren't able to comprehend tasks anywhere near the level of complexity of those that humans perform, as those tasks can require reasoning at a much higher level than raw perception.

Attempts have been made to bridge deliberative symbolic reasoning and continuous reactive behavior. Many of these systems fit the general diagram in Figure 1a. In this sort of system, perception abstracts out information from the environment, and provides it to the symbolic system. The symbolic system reasons over this information, and invokes a controller with an abstract command. Following this command, the controller maps low-level input to output, causing action in the world. The perception box provides symbols that completely and concisely describe the environment, and invoking a particular action results in a predictable transition in these symbols. This is the classic blocks-world approach: the perception system provides symbols like (on a b), actions are represented by symbols like (move b table), and simple rules describe how the perception symbols will change in response to the action, enabling symbolic action planning.

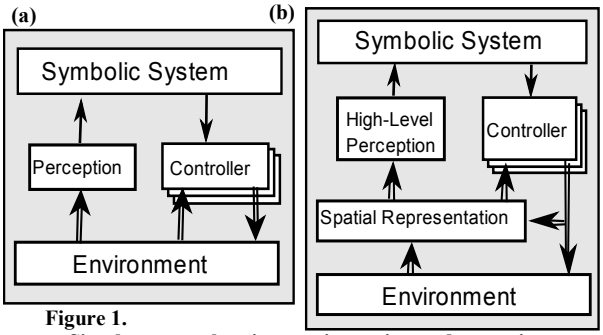

Figure 1.
a: Simple approach to integrating action and reasoning.
b: Our approach, where a spatial imagery layer mediates between the world and the symbolic system.

Embodied AI systems have moved beyond this simple sort of system, but retain much of the flavor. In typical approaches, the core of the system deals with the world in terms of an abstract representation, and peripheral processes, considered as largely independent of the problem-solving part of the system, convert between the abstract representation and the world. This approach leads to systems where the perception and action pieces are designed together—each action corresponds to a straightforward transition in perception, so reasoning solely in terms of the abstract representation is possible.

We are investigating an alternative architecture that integrates action, perception, and reasoning in such a way that the generality of the overall system is increased. The key aspect of this system (Figure 1b) is that perception and action are mediated through a simple spatial representation. The interface between this representation and the symbolic system is generic and fixed. In addition to causing action in the world, controllers can simulate action in the spatial representation.

With this system, generality is enhanced in two ways compared to systems like those in Figure 1a. First, mediating action and perception through a spatial representation makes the system highly modular, allowing different low-level controllers or high-level strategies to be used without changing the rest of the system. Secondly, the need for complicated action-specific perception processes, as well as strategy-specific controllers, is greatly reduced.

Since reasoning does not take place entirely at the abstract symbolic level, perception does not have to provide enough information to completely describe the world, nor must controllers be designed such that commands result in predetermined transitions between abstract states.

This work is done in the context of the Soar cognitive architecture (Laird, 2008). Soar has been extended recently to handle problems involving space and motion, using specialized representations and mechanisms inspired by human perception and mental imagery. The extension to Soar is SVS (Spatial/Visual System, Wintermute and Lathrop, 2008). SVS provides the core of an action simulation and execution system.

To examine this system in detail, a case study will be used. Since research in Soar/SVS is moving towards robotics, a modern robotic motion-planning algorithm will be examined. The RRT algorithm (LaValle, 2006) has been used in many recent robots (e.g., Leonard et al., 2008), and works through simulating action. Looking at developments in the field of motion planning that led to RRT and other similar algorithms, we can provide evidence that simulation is an appropriate way to reason about action. We will instantiate RRT in Soar/SVS, showing how all of the parts of the system work together, and the generality this approach affords.

Simulation and Motion Planning

A key aspect of our approach to action and planning is that simulation is used for planning, rather than abstract symbol transformations. Our chief reason for doing this is that it leads to a more general system; however, there is increasing evidence from robotic motion planning research that regardless of generality concerns, simulation is an appropriate way to plan motion.

Motion planning research has usually been pursued outside of the context of creating generally intelligent systems. Earlier approaches focused on efforts to exactly compute optimal paths for particular classes of robots, such as polygon robots that can move in any direction. This involves computing exactly the configuration space of the robot, a space in which any path corresponds to a real path to robot is able to follow. As motion planning has progressed to address problems involving more and more realistic robots, however, this exact computation has become intractable (Lindemann and LaValle, 2003).

One reason for this difficulty is that certain kinds of constraints on motion are infeasible to capture in representations like configuration spaces. Nonholonomic constraints result from systems where the number of controllable dimensions is less than the total number of degrees of freedom. For instance, a car is nonholonomic, since its position can be described by three parameters (x, y, and an angle), but it is only controllable in two dimensions (driving forward and reverse, and steering left and right). Where it is relatively straightforward to calculate the configuration space of a robot that can turn in place, this is not as simple with a car-like robot. In addition to nonholonomic constraints, traditional geometric motion planning approaches also have trouble incorporating dynamic constraints, where the path of the robot is affected by dynamics, such as a car that can't stop without slowing.

Recently, sampling-based approaches have become popular for planning with dynamic and nonholonomic constraints (LaValle, 2006). In sampling-based motion planning, the goal is not to exactly compute the configuration space, but instead to sample it through simulation. While previous approaches required difficult-to-calculate specialized representations that were specific to the particular motion under consideration, motion planning through simulation requires only a basic spatial representation, as details particular to the motion are encapsulated in the controller. If the simulation is accurate, motion plans can be guaranteed to meet nonholonomic and dynamic constraints.

This development from computation of configuration space to sampling reflects only two of many prominent techniques in motion planning. It is also worth mentioning behavior-based approaches, where representations are eschewed, and motion planning emerges from relatively simple mappings from perceptions to actions (e.g., Brooks, 1991). While our approach most certainly involves representations, it allows techniques developed in this tradition to be used as controllers within a broader symbolic AI system (in our case, the equations of Fajen and Warren, 2003, are used).

The RRT Algorithm

RRT (Rapidly-exploring Random Trees, LaValle, 2006) is a sampling-based motion planning algorithm that works by constructing a tree of states of the robot, rooted at the initial state, and adding nodes until that tree reaches the goal. Nodes are generated by extending the tree in random directions, in such a way that it will eventually reach the goal, given enough time. Each path from the root of the tree to a leaf represents a path that the robot could take, constantly obeying all constraints on its motion. The tree is constructed by the algorithm in Figure 2.

Two of the steps in this figure hide the true complexity of the algorithm. The steps to get the closest node to the random state, and to extend that node towards the random state, can both take substantial computation. This computation is also specific to the exact problem being solved, where the rest of the algorithm is general to all motion planning problems.

To determine the closest existing state to a random state, some metric must be determined that can measure the distance between states. In the case of car path planning, a simple metric is the Euclidian distance between the two

```
make tree rooted at initial state
while tree does not reach goal
  generate random state -> Xr
  get closest existing state to Xr -> Xc
  extend Xc towards Xr -> Xn
  if no collision occurred
    add Xn to the tree, connected to Xc
```

Figure 2. Basic RRT Algorithm

states, with the condition that the distance is infinite if the target state is not in front of the source.

The other problem-specific step in the algorithm is extending the chosen node towards the new state, while detecting collisions along the path. A typical approach is to numerically integrate differential equations to simulate motion, resulting in a sequence of states parameterized by time. This simulation must occur within a system capable of detecting collisions.

The Soar/SVS Architecture

The Soar cognitive architecture (Laird, 2008) provides the basis of our system. Work in Soar has traditionally focused on symbolic processing, and that remains a strength of the system. SVS builds on two prior lines of work on extending Soar to encompass non-symbolic spatial and visual representations, SVI (Lathrop, 2008), and SRS (Wintermute and Laird, 2008).

SVS encompasses the capabilities of both of these previous systems. Figure 3 shows the relevant parts of the system for this discussion. Compared to the generic diagram in Figure 1b, the names of some of the parts have changed, and two ways of adding objects to the scene from the top down have been added. SVS contains a long-term memory of the 3D structure of objects and the relationships between them, called the Perceptual LTM. By accessing this memory, Soar can recall objects or entire scenes to its Spatial Scene short-term memory as necessary. Sometimes, it is also necessary to add qualitatively described new objects to the scene, such as a line between two objects. This is called predicate projection (Chandrasekaran, 1998). A complementary process, predicate extraction, allows Soar to determine qualitative information about the contents of the scene, such as whether two objects intersect. This is the high-level perception box from Figure 1b, it contains the fixed processes through which Soar perceives the spatial world.

SVS also contains representations and mechanisms for dealing with 2D visual data, which are not used in this work. An external environment is also not used, all reasoning occurs in scenes retrieved from memory.

SVS and its predecessor systems have been used to study problems such as reasoning about positioning of military scouts to observe enemy units (Lathrop, 2008), and determining the layout of buildings in a real-time strategy game (Wintermute and Laird, 2008). Particularly relevant to this work, we have previously addressed reasoning with motion (Wintermute and Laird, 2008).

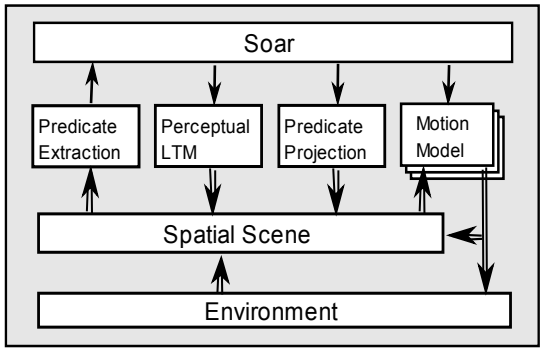

Figure 3. The SVS system.

The interface between SVS and Soar is designed to work at a qualitative level (outside of a few exceptions). This means that information exchanged between Soar and SVS is expressed chiefly in terms of references to known items in one of SVS's memories and a library of fixed qualitative relationships. Detailed quantitative information, such as the continuous coordinates describing the polyhedrons in the scene, is inaccessible to Soar.

One way to add new objects to the SVS scene is by applying a known motion to an existing object. Motions are encapsulated as motion models. These allow stepwise simulations of particular motions to be invoked. To do this, the Soar agent specifics which motion model to apply, which existing object to move, and any other object parameters, such as a goal object. For example, the agent can apply motion **`drive`** to object **`car`**, moving towards object **`goal`**. SVS responds to this command by building an image of **`car`**, driving towards **`goal`**. The symbolic command also has a parameter for time—changing the time steps the motion forward.

Complex termination conditions are not declared to SVS beforehand, instead, the general capabilities that allow Soar to extract information from the scene are used. For example, Soar can query whether objects in the scene are intersecting. If the conditions for car-driving to terminate are that it has collided with an obstacle, Soar can query as to whether the moving car intersects with any obstacles, and stop updating the motion once this becomes true.

Motion models can be used to reason over any motion, including the agent's own actions, actions of others, and environmental movement. In the first case, the motion model will be the same as the actual controller for the motion, just simulating its output in the scene, instead of executing it in the environment. We will use the terms "motion model" and "controller" interchangeably when talking about actions for this reason.

Implementing RRT in Soar/SVS

In order to explore reasoning about action in our system, we have instantiated the RRT algorithm in a Soar agent. The problem we considered is that of planning to drive a car from an initial state to a goal region, while avoiding obstacles in a known environment. The car motion model takes as input the identity of a car in the scene, and the location of a goal. Inside this model, a system of differential equations that describe the configuration of a simple car-like vehicle as a function of the time and goal location is used. When integrated numerically, these equations yield a sequence of configurations, allowing for simulation. These equations were determined by combining a model of human movement (Fajen and Warren, 2003) with a simple car model (LaValle, 2006). The human model controls the intended steering angle of the car, and this steering angle determines the next position of the car. A constant speed is assumed.

The controller simulates motion towards a goal, while maintaining the nonholonomic constraints of the vehicle. Along with geometric models of the car and world in the LTM of SVS, it is the low-level knowledge that was added to the existing SVS system to implement this planner.

Symbolic Soar rules were written to perform the algorithm in Figure 2. The algorithm relies on existing architectural functionality in the interface between Soar and SVS. As a metric for node distance, our implementation used the Euclidean distance, with the condition that the distance is infinite where the goal is not "in front" of the node. SVS includes mechanisms for Soar to extract distances, and to query for an in-front relationship. The motion model described above enables simulation, and SVS supports querying for intersections between objects in the scene, enabling collision detection. The only new mechanism in SVS to support this algorithm was a method to generate random goal points in the scene, which was a simple addition.

Examples of the SVS scene during RRT planning are shown in Figure 4 (top). Soar stores, as a symbolic structure in its working memory, the RRT tree. The nodes in that tree are symbolic indexes into SVS—they point to specific objects in the scene, which can be seen in the figure. Soar proceeds by adding new random point objects to the scene, and querying for the distance from each node to that object, which are then compared to find the closest. A simulation is then instantiated with that node as the initial conditions (creating a new car object in the scene), this simulation is stepped until a certain time is reached, the goal is reached, or Soar detects a collision with an obstacle. In all but the last case, the termination of the simulation results in a new tree node being added. In addition to moving towards random points, with a certain probability the agent instead tries to extend the tree directly towards the overall goal, biasing the tree in that direction.

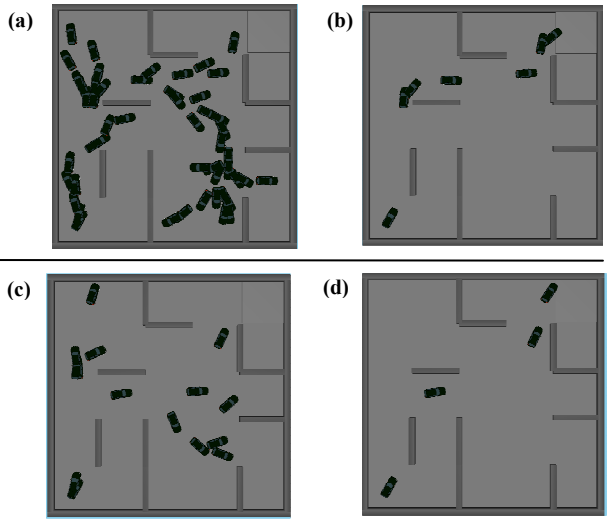

Figure 4. States of SVS Spatial Scene during RRT planning. The problem is to drive a car from lower-left to upper-right.
(a). RRT tree for goal-seeking controller, just before a solution is found.
(b). Sequence of car positions that solve the problem.
(c), (d). Same, for obstacle-avoiding controller.

RRT Planning with Local Obstacle Avoidance

It is clear that the controller in the previous example leaves much room for improvement. The task is to avoid obstacles while driving towards the goal, but the controller knows nothing about obstacles. All obstacle avoidance happens by pruning the tree when simulations collide with obstacles.

As is done in Fajen and Warren (2003), the above controller can be enhanced to be biased to avoid obstacles, in addition to moving toward the goal. Each obstacle affects the steering of the car, with nearer obstacles located towards the front of the car having the most influence. This requires slightly more information to be provided to the controller: the identities of the obstacles. Other than that, the system outside of the controller remains unchanged.

With this new, smarter controller, performance is greatly increased. An example instance is shown in Figure 4 (bottom), note that the number of states in the RRT tree before a solution is found is much less than in the top of the figure. Over 100 trials of the scenario presented in Figure 4, the average number of individual simulations needed to solve the problem was 128 without obstacle avoidance and only 12 when it was present.

Discussion

These case studies exemplify how reasoning and action can be integrated through simulation. In general, problems are decomposed between high-level algorithmic knowledge, instantiated in the symbolic system, and low-level action knowledge, instantiated as motion models. There is a spatial representation allowing real and imaginary situations to be instantiated, and fixed processes which communicate between that representation and the symbolic level. Complex reasoning about action occurs through interplay of low-level simulation and high-level symbolic reasoning over simulation results, detecting spatial interactions and making decisions accordingly.

The interface between the spatial and symbolic levels is the only means by which symbolic processing in Soar can obtain information about the results of lower-level processing. Soar uses the Predicate Extraction system in Figure 3 to query the scene for information such as which objects intersect which, relative directions, and other simple qualitative relationships. The commitment to make this system fixed is important, as it implies that the type of qualitative spatial information available to the symbolic system is the same, regardless of the task. The poverty conjecture of Forbus et al. (1991) states that "there is no purely qualitative, general-purpose, representation of spatial properties", and if this is true, the decision to use a fixed representation, calculated without task knowledge, seems a poor choice.

However, this conjecture might not apply to our system, as the qualitative representation used does not have to be rich enough to enable purely qualitative reasoning (and hence is not really "general purpose"). The essential role of simulation with respect to symbolic reasoning is that of an

action model. The system is able to predict the symbolic consequences of symbolically described actions—for example, inferring that driving a certain car object toward the goal leads to a collision. Since simulation is available, the qualitative representation used needs to be only rich enough to capture important aspects of the current state of the spatial system, not the implications of action.

The alternative to this is to represent the consequences of actions completely symbolically, as in planning systems such as STRIPS. Consider a problem in a variant of the blocks world. In this problem, a robot finds itself in front of a table with blocks on it, and must stack them in a certain way (e.g, A on B, and C on A). Each time it encounters a table, the robot's perception system must encode the problem in such a way that planning can be performed. This encoding must capture both the states of the world (such as on(B,C) and clear(C)), and the way those states are transformed by action. The robot can make certain assumptions about any table it encounters. For example, if a block is clear (has nothing on top of it), it can be moved to the top of a different clear block. However, in a divergence from the standard blocks world, the table is not always clear—in this case, the table is divided into numbered bins, and blocks must be placed entirely inside a bin. Not all of the bins are wide enough to fit a block, though, and the block will fall to the floor if the agent places it there. So depending on the exact table the robot has encountered, the result of moving block A into bin 3 is either inBin(A, bin3), or onFloor(A).

To allow purely symbolic planning in this domain, the perception system of the robot must then compare each bin's size to the width of a block, and generate its state transition rules accordingly. In this way, creating the qualitative representation of the problem requires very task-specific calculations to be performed in the perception system in order for that representation to capture the consequences of actions. In contrast, if actions can be simulated in a spatial representation, this can instead be accounted for by simulating placing a block in the bin, and simply checking if it intersects a wall

While there is no proof that the symbolic representations used in Soar/SVS are general enough that all reasonable tasks can be represented, it is clear that the symbolic representation needed when actions can be simulated is much simpler than the representation needed to solve problems entirely at the symbolic level. The high-level perception system that builds this representation can then be much simpler if simulation is possible.

Similarly, the ability to simulate actions simplifies the requirements of the low-level action system. Consider again the standard planning approach to a problem like those in blocks world. In order to represent the problem completely symbolically, the effect of every possible action on every possible state must be characterized before the problem can be solved. In addition to increasing the need for task-dependant perception, as discussed above, this requires that the controllers used by the system have simple-to-characterize guarantees on their performance. For instance, in blocks world, it is assumed that the controller can reliably perform an action like moving a block from the top of one stack to another for any two stacks present. If actions can instead be simulated, these guarantees are unnecessary—the behavior of the controller does not need to be symbolically characterized before reasoning can begin.

To further illustrate this point, contrast the RRT agent with obstacle avoidance described above with a more traditional graph-search approach. If a robot can be approximated by a circle and can turn in place, concise representations of relevant locations for the robot to move in the world can be derived. Using this representation, a graph of nodes and distances between them, optimal path planning can be done using an algorithm like A*. If this graph is the robot's perception, and its actions move between nodes, reasoning about action is a simple graph search. Consider the complication involved in using this form of perception and action, though. Perception involves computing the configuration space of the robot, and paths within it that lie entirely outside of obstacles. Actions must map directly onto transitions between nodes of the graph, so a controller must be used that guarantees accurate execution of these transitions.

In contrast, the RRT planner needs only very simple perceptions, such as which objects intersect which others, and the general direction and distance between objects. The actions of the robot do not have to be symbolically characterized beforehand for the planner to work. For a controller like the obstacle-avoiding car used above, it would be very hard to characterize in what cases it is able to guide the robot to the goal and in what cases it isn't.

Ignoring other concerns, such as the limited applicability of graph-search planning for more complicated robots, and the suboptimal performance of RRT for simple robots, it is clear that integrating reasoning and action through simulation requires simpler perception and less-constrained action systems than reasoning solely at the symbolic level.

This simplification of perception and action systems afforded by the use of simulation allows for a highly modular overall system. The interface between the symbolic system and spatial representation, which involves complicated transformations between symbolic and quantitative representations, is fixed; new capabilities can be added to the system without modifying this interface. For example, a different approach to path planning has been previously implemented in Soar/SVS (Wintermute and Laird, 2008). In this case, the problem was approached by detecting which obstacles lie between the car and the goal, and deliberately creating waypoints in the scene to divert around them. This is a very different overall algorithm than RRT, but the agent itself differs only in the high-level Soar rules that describe it. Changes in the interface between Soar and the spatial system were not needed, nor were changes in the controller.

This fixed symbolic/quantitative interface also allows for modularity in the controllers used. This is seen in the above examination of the RRT planner, where an obstacle-

avoiding controller was substituted for a simpler goal-seeking controller, while requiring minimal changes to the rest of the system. Controllers in the system take the majority of input and create output in terms of spatial objects. Their interface to the symbolic system simply points them to the relevant objects in the scene and provides time. Controllers can be built which work internally with continuous numbers, the same format in which the scene is represented, so there is no need for a difficult representation conversion at the controllers input and output like there might be if the controller interfaced only to the symbolic system.

Related Work

The integration of reactive behavior and deliberative reasoning in robotics architecture has been investigated, such as by Arkin (1989) and many others. However, most of these systems involve using deliberative reasoning to serialize different reactive behaviors (thus fitting Figure 1a), and do not involve simulation. Some previous systems do use simulation of reactive behavior to guide reasoning. MetaToto (Stein, 1994) emphasizes this, but doesn't focus on enabling general high-level reasoning as our system does. 4D/RCS (Albus, 2003) includes simulation ability, but is more of a scheme for organizing an agent, rather than a commitment to a specific set of components.

From an AI point of view, this work inherits much from research in diagrammatic reasoning (e.g., Chandrasekaran, 1997). Comirit (Johnston and Williams, 2008) is also a similar approach to ours, integrating logic with physical simulation, but is not used for action and doesn't have a spatial representation. Our use of a spatial representation to simplify symbolic reasoning builds on previous work looking at the frame problem (Huffman and Laird, 1992).

Conclusion

We have shown that Soar with SVS is able to use imagery to solve motion planning problems. This was done using the existing fixed functionality of SVS to communicate between Soar and the scene, and adding symbolic knowledge to Soar encoding the high-level RRT algorithm and low-level knowledge in the form of a controller. While this system has not yet been implemented on a real robot, RRT has (e.g., Leonard et al., 2008), so some version of this approach is feasible with current technology.

This system is presented as an example of integrating action and symbolic reasoning through spatial simulation. This allows the symbolic system to reason about the problem, but removes the requirement that the entire problem be represented symbolically. Because of this, the perception system needed is simpler and more problem-independent than would otherwise be required, and controllers can be used without symbolically characterizing them beforehand. A very modular system is then possible, since symbolic and control processing are mediated by the spatial system and its fixed qualitative/spatial interface. While many existing systems use major elements of this approach, this work has attempted to make explicit the argument for how and why this form of integration is a step on the path toward more general AI systems.

Acknowledgements

John Laird provided guidance and support on this project, and assisted in editing. Joseph Xu and Nicholas Gorski provided useful conversations in formulating the ideas behind this paper. This research was funded by a grant from US Army TARDEC.

References

Albus, J.S., 2003. 4D/RCS: A reference model architecture for intelligent unmanned ground vehicles. In *Proceedings of SPIE*.

Arkin, R.C., 1989. Towards the unification of navigational planning and reactive control. In *AAAI Spring Symposium on Robot Navigation*. Stanford, CA.

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

Chandrasekaran, B., 1997. Diagrammatic representation and reasoning: some distinctions. In *AAAI Fall Symposium on Diagrammatic Reasoning*. Boston, MA.

Fajen, B.R. & Warren, W.H., 2003. Behavioral dynamics of steering, obstacle avoidance, and route selection. *J. Experimental Psychology: Human Perception and Performance*, 29(2).

Forbus, K.D., Nielsen, P. & Faltings, B., 1991. Qualitative spatial reasoning: the CLOCK project. *Artificial Intelligence*, 51(1-3).

Huffman, S. & Laird, J.E., 1992. Using Concrete, Perceptually-Based Representations to Avoid the Frame Problem. In *AAAI Spring Symp. on Reasoning with Diagrammatic Representations*.

Johnston, B. & Williams, M., 2008. Comirit: Commonsense Reasoning by Integrating Simulation and Logic. In *Proc. First Conference on Artificial General Intelligence*.

Laird, J.E., 2008. Extending the Soar Cognitive Architecture. In *Proc. First Conference on Artificial General Intelligence*.

Lathrop, S.D., 2008. *Extending Cognitive Architectures with Spatial and Visual Imagery Mechanisms*. PhD Thesis, University of Michigan.

LaValle, S.M., 2006. *Planning Algorithms*, Cambridge U. Press.

Leonard, J. et al., 2008. A perception-driven autonomous urban vehicle. *Journal of Field Robotics*, 25(10)

Lindemann, S.R. & LaValle, S.M., 2003. Current issues in sampling-based motion planning. In *Proceedings of the International Symposium of Robotics Research*. Springer.

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

Stein, L.A., 1994. Imagination and situated cognition. *Journal of Experimental and Theoretical Artificial Intelligence*, 6.

Wintermute, S. & Laird, J.E., 2008. Bimodal Spatial Reasoning with Continuous Motion. In *Proceedings of AAAI-08*. Chicago.

Wintermute, S. & Lathrop, S.D., 2008. AI and Mental Imagery. In *AAAI Fall Symposium on Naturally Inspired AI*.