Comprehension and Prediction of Astronaut Dynamics

D. Paul Benjamin
John Vincent Monaco
Yixia Lin
Pace University
1 Pace Plaza
New York, NY 10038
212-346-1012
benjamin@pace.edu

Damian M. Lyons
Fordham University
340 JMH, 441 E. Fordham Rd.
Bronx, NY 10458
718-817-4485
dlyons@fordham.edu

Abstract—A robot collaborating with astronauts needs to comprehend and predict their movements. We present an approach to perceiving and modeling astronaut movement using a 3D virtual world. The robot's visual data is registered with the virtual world to construct a model of the astronauts' dynamics and predict future motions using a physics engine. This enables the robot to interact more naturally with the humans and to avoid potentially disastrous mistakes.

This approach is implemented within a cognitive architecture equipped with a learning mechanism. This enables the robot potentially to learn to classify behaviors more quickly and accurately with experience.[1]

We present previous work with simple objects and preliminary experiments with basic human movements including sitting, standing and turning.

1. INTRODUCTION

A truly cognitive architecture has not yet been implemented in robotics. Robots have been programmed to perform specific tasks such as mowing the lawn or navigating in the desert, and these accomplishments can be impressive, but robots still cannot act autonomously to choose tasks and devise ways to perform them. Even when performing their allotted tasks, they lack flexibility in reacting to unforeseen situations. Currently, the design of important perceptual and decision-making structures is done by the programmers before the robot begins its task. The semantics for the symbols and structures the robot uses is determined and fixed by these programmers. This leads to fragmented abilities and brittle performance. The robots cannot adapt their knowledge to the task, cannot solve tasks that are even slightly different from those they have been programmed to solve, cannot communicate effectively with humans about their goals and performance, and just don't seem to understand their environment. This is a principal stumbling block that prevents robots from achieving high levels of performance on complex tasks, especially tasks involving interaction with people.

The ADAPT project (**A**daptive **D**ynamics and **A**ctive **P**erception for **T**hought) is a collaboration of three university research groups at Pace University, Brigham Young University, and Fordham University that is building a robot cognitive architecture that integrates the structures designed by cognitive scientists and linguists with those developed by robotics researchers for real-time perception and control. ADAPT is under development on Pioneer robots in the Pace University Robotics Lab and the Fordham University Robotics Lab. Publications describing ADAPT are [1, 2, 3, 4].

ADAPT models the world as a network of concurrent schemas, and models perception as a problem solving activity. Schemas are represented using the RS (Robot Schemas) language [8,9], and are activated by spreading activation. RS provides a powerful language for distributed control of concurrent processes. Also, the formal semantics of RS [10] provides the basis for the semantics of ADAPT's use of natural language. We have implemented the RS language in Soar, a mature cognitive architecture originally developed at Carnegie-Mellon University and used at a number of universities and companies. Soar's subgoaling and learning capabilities enable ADAPT to manage the complexity of its environment and to learn new schemas from experience.

2. THE ADAPT ARCHITECTURE

Our approach is fundamentally different from other projects, which typically attempt to build a comprehensive system by connecting modules for each different capability: learning, vision, natural language, etc. Instead, we are building a *complete cognitive robotic architecture* by merging RS [8,9], which provides a model for building and reasoning about sensory-motor schemas, with Soar [5], a cognitive architecture that is under development at a number of universities. RS possesses a sophisticated formal language for reasoning about networks of port automata and has been successfully applied to robot planning. Soar is a unified cognitive architecture [7] that has been successfully applied to a wide range of tasks.

Soar's model of problem solving utilizes a single mechanism of subgoaling and chunking to explain human problem solving performance; utilizing Soar as the basis of

[1]978-1-4244-7351-9/11/$26.00 ©2011 IEEE.

ADAPT permits us to unify the mechanisms underlying perception, language and planning. Furthermore, it permits us to explore possible interrelationships between learning in these areas, e.g. how learning language and learning perception may be related. Finally, it permits us to test our architecture on robotic versions of well-known cognitive tasks and explore how robot learning might be related to human learning.

3. ROBOT SCHEMAS

ADAPT's representation of dynamic behaviors is based on the RS (Robot Schemas) language [8,9]. RS is a language with a formal model of robot computation that is based on the semantics of networks of *port automata* [10]. A port automaton (PA) is a finite-state automaton equipped with a set of synchronous communication ports.

RS builds a network of *sensory-motor schemas* to model the dynamics of both the robot and the environment. A schema is an organized network of actions (abstract and/or concrete), percepts, words, facts and beliefs about some aspect of the world. Schemas also contain explicit qualitative temporal information, e.g. about intervals of time during which an action must take place. The schema relates these components to each other and to other schemas.

For example, the schema for a table would contain various images of tables, and would connect them to facts about tables, such as their typical size, and to actions that typically are performed with tables, such as eating at one (which is another schema). The table schema would be connected to the schemas for chairs, dining rooms, conference rooms and many other relevant concepts.

One of the unique advantages of RS is its formal semantics. Each schema has an associated port automaton that defines the semantics of the schema. ADAPT explicitly constructs this PA for each schema and attaches it to the schema. The PA performs two important functions.

First, it provides the basic semantics for the use of natural language. States in the PA provide the semantics for relations and predicates (adverbs and adjectives). Transitions between the states provide the semantics for verbs.

Second, it enables ADAPT to synthesize hierarchies of schemas by decomposing automata. Benjamin[6] has developed an algebraic method of factoring an automaton's associated semigroup. This method is applied in a straightforward way to PAs, factoring each PA into a collection of simpler PAs and decreasing the cost of perception and action.

A large body of work in intelligent systems employs similar notions of schema. Arbib[1] presents a comprehensive theory of schemas.

ADAPT uses schemas by instantiating them to create schema instances that are connected to each other to form a network. Over time, ADAPT maintains this network, monitoring its behavior and adding schema instances and removing old ones as its goals and the environment change.

A network of processes is typically built to capture a specific robot sensory-motor skill or behavior: sensory processes linked to motor processes by data communication channels and sequenced using *process composition operations*.

For example, the following RS statement defines the i-th joint of a robot in terms of three sub-units, which are connected by connection map that connects similarly named output and input ports:

$$\mathrm{Joint}_i(s)() = [\mathrm{Jpos}_i()(x) \mid \mathrm{Jset}_i(s, x)(u) \mid \mathrm{Jmot}_i(u)()\ ]$$

$\mathrm{Jpos}_i()(x)$ continuously reports the position of joint i on output port x

$\mathrm{Jmot}_i(u)()$ accepts a signal on input port u and applies it to the actuator of joint i

$\mathrm{Jset}_i(s, x)(u)$ accepts a set-point on input port s and iteratively inputs a joint position on

port x and outputs a motor signal on port u to drive the joint position to the set-point

This RS schema can then be used as a sub-unit of a guarded move schema that is higher in the hierarchy:

$$\mathrm{Touch}_i = [\mathrm{Tact}_i()(v) \mid \mathrm{Gmove}_i(v)(y) \mid \mathrm{Joint}_i(y)()\ ]$$

Tact_i reports on tactile contact on the i-th join on its port v.

Gmove increments the setpoint of the joint actuator as long as it gets a no-contact signal on port v.

Touch_i implements a guarded move of the i-th link.

RS process composition operations are similar to the well-known CSP algebraic process model. However, unlike CSP, in RS the notation can be seen as simply a shortcut for specifying automata; a process is a port automaton, and a process composition operation is two automata connected in a specific way. Composition operations include sequential, conditional and disabling compositions. To analyze a network of processes, it is necessary to calculate how that network *changes* as time progresses and processes terminate and/or are created. This is the process-level equivalent of the PA transition function, combined with the axioms that define port-to-port communication. This *Process Transition function* can be used to analyze the behavior of RS networks.

RS has been used successfully in industrial settings for applications such as kit assembly. RS thus provides a mature system for robust, real-time distributed control. A short video showing RS controlling a robot arm and camera to track and grasp a moving object is available at http://csis.pace.edu/robotlab/clips/puma.avi.

RS and Soar

RS provides a powerful representational language for the system's dynamics, language and percepts; however, RS does not provide a mechanism for synthesizing the dynamics. Furthermore, RS lacks demonstrated cognitive plausibility, and in particular lacks a learning method.

We have implemented RS in Soar to take advantage of Soar's cognitively plausible problem-solving and learning mechanisms. Soar uses *universal subgoaling* to organize its problem solving process into a hierarchy of subgoals, and uses *chunking* to speed and generalize that process. Universal subgoaling permits Soar to bring all its knowledge to bear on each subgoal. Chunking stores generalized preconditions for search control decisions, so that in future tasks similar search control decisions are made in a single step.

ADAPT's hierarchy of schemas is attached to the top-level state in Soar. ADAPT uses Soar's universal subgoaling to create its hierarchy of schemas, via operators that select, instantiate and interconnect schemas. This permits Soar's chunking method to learn how to build, monitor and modify networks of schemas.

General schemas are kept permanently in working memory, so that they function as a kind of declarative memory. Each schema is linked to other schemas that are relevant, e.g. tables are connected to chairs. This library of schemas in working memory that can be instantiated and activated is similar to the declarative memory in which ACT-R stores its chunks. We follow the example of ACT-R and permit ADAPT to create links between schemas to reflect relationships between them, such as “is a generalization of” and “is the opposite of”, as well as activation links labeled by weights that can be used by spreading activation in a manner similar to ACT-R.

Schemas have *input ports* and *output ports*. The ports of schema instances are connected to create ADAPT's network. Schemas can be decomposed into an assemblage of schemas that represent the world in more detail. Such *assemblage schemas* are attached to their parent schemas, creating a hierarchical structure that represents the world at varying granularities. Schemas that can be directly carried out by our Pioneer robot's Aria software possess an implementation attribute that contains the robot commands.

Each schema instance in the hierarchy has a weight and a priority, which are used by ADAPT to compute what the robot will actually do. Unlike Soar and other cognitive architectures, ADAPT does not select a single schema to execute on the robot. Instead, all active schemas that possess implementations are *blended* by ADAPT's *resolver* to create the actions actually executed on the robot platform. Blending involves weighting each schema implementation by its weight, adding together weighted implementations of equal priority, then disabling lower-priority actions if they contradict higher-priority actions.

For example, suppose a "move-forward" schema and a "turn-right" schema both have equal priority, with "move-forward" weighted by 0.9 and "turn-right" weighted by 0.1". The implementation of "move-forward" is for both wheels to move at speed 100, and the implementation of "turn-right" is for the right wheel not to move and the left wheel to move at speed 100. Then the blend of these two schemas will move the left wheel at speed 100 and the right wheel at speed 90, resulting in a slight angling to the right. If another move schema has a lower priority, it will be disabled. If another schema with a lower priority does not contradict the moves, e.g. a schema that tilts the camera, then it will be

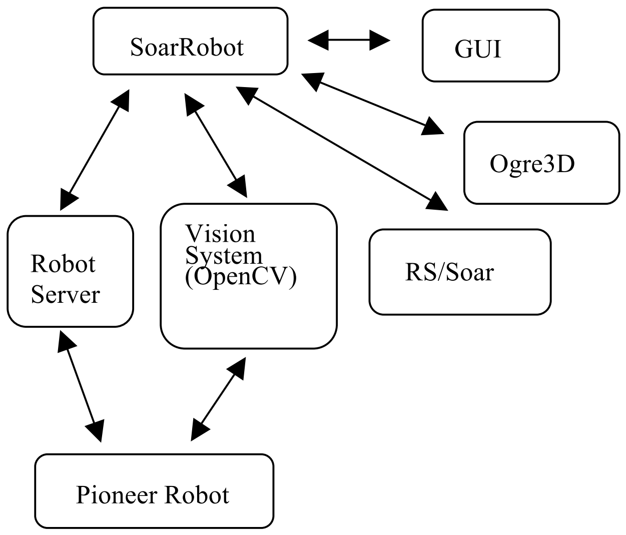

Figure 1. ADAPT’s structure.

executed concurrently.

ADAPT subgoals in the same manner as Soar, but subgoaling does not necessarily choose one schema, but instead assigns weights for blending. Choosing one schema to execute is simply making one schema's weight 1 and all others' weights 0.

ADAPT's chunking mechanism is that of Soar. It creates a new rule (chunk) that summarizes preconditions for a subgoal's result. If the subgoal is selecting, instantiating or removing a schema, then the chunk will perform the operation in a single step. If the subgoal operates on multiple schemas, then the chunk will perform all the schema operations at once; in this way ADAPT can learn truly concurrent operations. For example, a subgoal may be to keep an object in focus while turning right. ADAPT can decide to instantiate a schema to turn the cameras left while it is running a schema to turn right. The decisions to instantiate and start both schemas will appear in one chunk, thus encoding distributed, concurrent control.

Thus, ADAPT's approach to modeling behaviors is to learn port automata from observed behaviors. The methods of learning automata are based on well-known methods of inductive inference of automata.

Soar manipulates a hierarchy of problem spaces, and the formal semantics of RS consists of a hierarchy of port automata. We have merged these architectures in a straightforward way by implementing each RS schema as a Soar problem space. This is done by specifying the state transitions of each schema's port automaton in Soar.

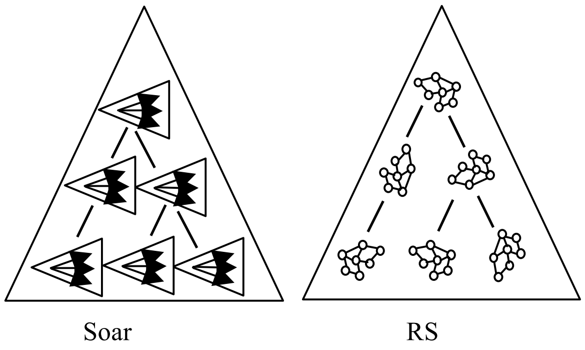

Figure 2. Soar problem spaces implement RS port automata.

Merging RS and Soar in this way combines their strengths. The strengths of RS include its formal mechanism for combining sensing and motion, its ability to reason about the temporal behavior of schemas, and its combination of deliberative planning and reactive behavior. Its weaknesses are the lack of a synthesis mechanism for autonomous formation of sensors or actuators, and the lack of a model for implementation of cognitive abilities such as learning and language.

Soar provides an integrated cognitive model with a full range of cognitive abilities, including perception, deliberative planning, reaction, natural language, learning, and emotion. But Soar lacks parallelism and a temporal mechanism, which severely hampers its usefulness in robotics.

By integrating RS and Soar, we have created an architecture to:

- process a wide range of modes of perceptual input,
- provide a complete range of cognitive abilities, including language and learning,
- reason about time and resources,
- implement parallel, reactive behaviors, and
- learn new high-level sensors and behaviors.

4. Comprehension by Visualization

The central goal of our work is to develop effective methods for robots to comprehend their environment. Our approach models comprehension as a process of trying to recreate the observed dynamics by hypothesizing various sets of goals and beliefs for the agents, generating their dynamics based on these assumptions and comparing it with the observed dynamics. This knowledge-intensive approach to comprehension has a history within AI and in particular in machine learning.

We have extended this approach to apply to comprehension of all observed behaviors, including motion and speech, because we view language comprehension as a special case of behavior comprehension. To say it the other way around, we believe that comprehension of non-speech behaviors is necessary for language comprehension. This necessity stems from two causes. The first is that the semantics of many words (especially verbs) requires comprehension of the activity they denote. The second is that speech is typically enhanced with many non-verbal actions, such as hand movements, facial expressions and postures.

Furthermore, we believe that the comprehension requires *visualization*. We view visualization as consisting of both a perceptual component and a reasoning component. The perceptual component is performed using the same perceptual mechanism that the robot uses to perceive its environment; the difference is that visualization perceives a simulation of the environment. Visual reasoning manipulates and superimposes representations that consist of a combination of symbolic knowledge and 3D animations.

Comprehension by generation requires the robot to be able to create different situations in which it can generate behaviors of robots, people and physical systems, and perceive the results of these behaviors. This requires implementing a virtual world that the robot can control.

ADAPT's virtual world is a multimedia simulation platform capable of realistic simulations of physical phenomena. It combines the various forms of map information found in most robots: topological, metric and conceptual information. ADAPT completely controls this virtual world, and can create arbitrary objects and behaviors in it, including nonexistent objects and behaviors that were not actually observed. Central to ADAPT's use of its virtual world is its ability to view these constructions from any point. This enables ADAPT to create visual representations with desired properties.

This approach to visualization is very different from previous work on reasoning about spatial relationships. ADAPT does not just turn spatial relationships into symbolic terms to be used in reasoning, but instead can reason visually about spatial relationships by constructing instances of those relationships, viewing them from various angles, and superimposing them.

In the current implementation, ADAPT's world model is the Ogre3D open source gaming platform (http://www.ogre3d.org). Ogre gives the robot the ability to create a detailed and dynamic virtual model of its environment, by providing sophisticated graphics and rendering capabilities together with a physics engine based on the PhysX physics engine. Ogre models a wide variety of dynamic environments, including modeling other agents moving and acting in those environments.

ADAPT uses this virtual world in a novel way. Typical robotics architectures connect their sensory mechanisms to their world models, so that sensory data is processed and modeled in the world model. The reasoning engine then operates on the world model to plan the robot's behaviors. This type of architecture treats perception as a separate process from the central reasoning, and typically the implementation reflects this, e.g. a computer vision module processes the vision data and puts symbolic representations of the recognized objects and their relationships in the world model, and the reasoning engine then manipulates these symbols to plan and learn. The reasoning engine does not process the sensory data.

In contrast, ADAPT's virtual world is not connected to its sensory processes. ADAPT's sensory data is placed directly in the reasoning engine (after some low-level processing); the reasoning engine's principal task in ADAPT is to reason about how to model the data. It does this in the following way:

> It creates virtual entities and behaviors in Ogre.
>
> It senses in the virtual world, using the same position and orientation as in the real world, and using the same sensors. For example, if ADAPT is modeling visual data, it grabs graphics input from Ogre, and if it is modeling sonar data, it grabs distance data from Ogre in the directions of the actual sonars.
>
> It compares the virtual sensory data with the real sensory data, using a least-squares measure to find the degree of disagreement.

The reasoning engine searches alternative combinations of virtual entities and behaviors to attempt to minimize the measured disagreement. In this way, *perception becomes a problem-solving process*. This enables all the knowledge of the system to be brought to bear on perception, and unifies the reasoning and learning processes of problem solving with those of perception.

This search can be long and expensive; for this approach to comprehension to be practical, an effective speedup learning mechanism is required to store the results of this search. ADAPT contains a knowledge compilation method that stores generalized results of each successful search. One of the main research goals of our project is to quantify the effectiveness of this approach.

Visualization is also used in ADAPT for *predictive vision*: the robot predicts what it expects to see based on its virtual world and pays attention only to significant differences. This part of the project is detailed in [2].

5. Predictive Vision

As an illustration of the use of this world model, let us consider ADAPT's novel approach to visual comprehension of its environment. ADAPT's vision system consists of two main components, a bottom-up component that is always on, and a top-down goal-directed component controlled by RS/Soar.

The bottom-up component component is simple and fast. It does this by not producing much detail. The idea is for it to produce a basic stereo disparity map, a coarse-grained image flow, and color segmentation in real time. It runs on the robot's onboard computer using Intel's open vision library, and segments the visual data from the robot's two framegrabbers. These "blobs" are transmitted together with stereo disparity data and optical flow to the offboard PC that is running RS/Soar, where it is placed into working memory. This component is always on, and its output is task-independent.

The top-down component executes the more expensive image processing functions, such as object recognition, sophisticated image flow analysis, and application of particular filters to the data. These functions are called in a task-dependent and goal-dependent manner by RS/Soar operators. This greatly reduces their frequency of application and speeds the operation of the vision system significantly.

These two components are not connected to each other; instead, the output of the bottom-up component is used by RS/Soar to determine when to call the top-down operations. RS/Soar compares the bottom-up output to the visual data predicted by the mental model.

The mental model can display the view that the virtual copy of the robot "sees" in the virtual environment. The output of this graphics "camera" in Ogre is segmented and placed into RS/Soar's working memory, together with distance information and motion information from the mental model. Soar operators test for significant differences between the expected view and the actual view, e.g. the appearance of a large new blob or a large change in optical flow. Any significant difference causes an operator to be proposed to attend to this difference.

For example, if a new blob appears, an operator will be proposed to look at this blob and try to recognize it. If this operator is selected (if there is nothing more important to do at the moment) then RS/Soar will instruct the robot to turn its cameras towards this blob, and then call its recognition software to process the rectangular region of the visual field that contains the blob.

Once the object is recognized, a virtual copy is created in Ogre. The object does not need to be recognized again; as long as the blobs from the object approximately match the expected blobs from Ogre, ADAPT assumes it is the same object. Recognition becomes an explicitly goal-directed process that is much cheaper than continually recognizing everything in the environment. The frequency with which these expensive operations are called is reduced, and they are called on small regions in the visual field rather than on the whole visual field.

Thus, ADAPT's vision system spends most of its time verifying hypotheses about its environment, instead of creating them. The percentage of its time that it must spend attending to environmental changes depends on the dynamic nature of the environment; in a relatively static environment (or one that the robot knows well from experience) there are very few unexpected visual events to be processed, so visual processing operators occupy very little of the robot's time.

Another advantage of this approach to visual comprehension is that it opens the possibility of learning recognition strategies.

Figure 3 shows the block diagram of our vision system [11,12]. The real and synthetic images of the scene as viewed by the robot are compared. If the scenes are considered the same but from different viewpoints, then the viewpoint of the camera in the simulation is changed, and the simulation generates an image taken by the camera at the new location. If an unexpected object is seen in the real image, an object is introduced at the corresponding position in the simulated scene. The region of the real image responsible for the difference is used as video texture on the object and a new synthetic image generated. The information on whether there is no difference, an unexpected object, or an object missing between the image pairs is made available to action planning. This loop of difference detection and simulation modification is used to keep the simulation synchronized to the observed environment. For prediction purposes, the simulation can be allowed to 'fast forward' in time, so that the expected position, for example, of a target can be calculated and then compared to observations.

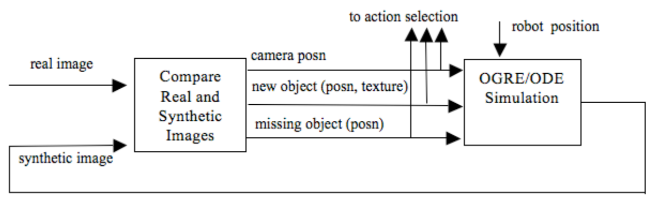

Figure 3: Block diagram of the loop integrating simulation and observation

Fig. 4 shows a real (4(A)) and synthetic (4(B)) view of the same scene taken with the artificial camera at approximately the same location and orientation as the camera in the real scene.

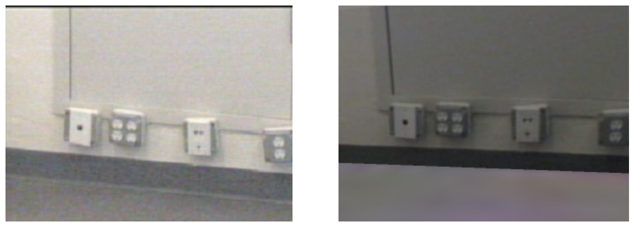

Figure 4: Real (A) and synthetic (B) views of the same scene from approximately the same position and orientation

The view presented here is a close up of one wall of a room. The scene is also modeled graphically using OGRE. Sections of the graphical scene have been tiled with video texture manually extracted from Fig. 4(A). The use of video texture should make it easier to directly compare the real image and synthetic image to answer the following questions:

1. Do they represent the same scene from the same viewpoint?
2. Do they represent the same scene from slightly different viewpoints?
3. Do they represent the same scene but with some number of different objects?
4. Do they represent different scenes?

6. RECOGNIZING DYNAMICS

Given the above method that can compute the differences between individual frames from the real and virtual worlds, we apply it to recognizing dynamics using comprehension by visualization as described in Section 4. We begin with an initial rendering of the real world into the virtual world. With each successive frame grabbed from the real cameras, we generate hypothetical dynamics and create them in the virtual world and compare the predicted visual difference. The hypothetical dynamic that generates the closest observed match is selected as the observed dynamic.

This process is repeated over a window of time (currently two seconds). If one hypothesis is closest at least half the time it is identified as the correct dynamic.

Our initial experiments were the dynamics of balls: rolling bouncing, spinning. Each of these dynamics was associated with real parameters reflecting velocity and height (for bouncing). These experiments were encouraging (more than 50% success rate of identification) but we found that the method of computing the differences included too much detail. We have begun work on a new method that relies more on higher level features of the scenes.

Also, we found that the spherical symmetry of the balls created difficulties in recognizing motion. We believe that working with less symmetric objects will make the recognition more successful.

We have begun a series of experiments on recognizing human movements: walking, standing, sitting, turning. Using hand-constructed schemas for each of these dynamics, we are currently learning classifications from comprehension through generation. This process is slow, an we are evaluating the effectiveness of Soar's speedup learning to improve speed.

7. Conclusion

We have sketched the overall design of a new approach to the comprehension of dynamics that is part of a robotic cognitive architecture. A powerful 3D multimedia world model is used to render dynamics. This gives the robot the ability to visualize alternative evolutions of the dynamics and to classify them. The implementation of the basic components is complete. We have begun experiments on to evaluate the architecture.

Further information on this work, including video clips showing the robot moving under the control of schemas and the use of the world model, can be downloaded from the website for the Pace University Robotics Lab: http://csis.pace.edu/robotlab

References

[1] Benjamin, D. Paul, Damian Lyons and Deryle Lonsdale, "Embodying a Cognitive Model in a Mobile Robot", Proceedings of the SPIE Conference on Intelligent Robots and Computer Vision, Boston, October, 2006.

[2] Benjamin, D. Paul, Damian Lyons and Thomas Achtemichuk, "Obstacle Avoidance using Predictive Vision based on a Dynamic 3D World Model", Proceedings of the SPIE Conference on Intelligent Robots and Computer Vision, Boston, October, 2006.

[3] Benjamin, D. Paul, Damian Lyons and Deryle Lonsdale, "Designing a Robot Cognitive Architecture with Concurrency and Active Perception", Proceedings of the AAAI Fall Symposium on the Intersection of Cognitive Science and Robotics, Washington, D.C., October, 2004.

[4] Benjamin, D. Paul, Damian Lyons and Deryle Lonsdale, " Cognitive Robots: Integrating Perception, Action and Problem Solving in Behavior-Based Robots", AAMAS-2004 Proceedings, pp. 1308-1309, 2004.

[5] Laird, J.E., Newell, A. and Rosenbloom, P.S., "Soar: An Architecture for General Intelligence", *Artificial Intelligence* **33**, pp.1-64, 1987.

[6] Lyons, D.M. and Hendriks, A., "Exploiting Patterns of Interaction to Select Reactions", Special Issue on Computational Theories of Interaction, Artificial Intelligence **73**, 1995, pp.117-148.

[7] Newell, Allen, *Unified Theories of Cognition*, Harvard University Press, Cambridge, Massachusetts, 1990.

[8] Lyons, D.M., "Representing and Analysing Action Plans as Networks of Concurrent Processes", IEEE Transactions on Robotics and Automation, June 1993.

[9] Lyons, D.M. and Arbib, M.A., "A Formal Model of Computation for Sensory-based Robotics", IEEE Transactions on Robotics and Automation **5**(3), Jun. 1989.

[10] Martha Steenstrup, Michael A. Arbib, Ernest G. Manes, *Port Automata and the Algebra of Concurrent Processes*. JCSS 27(1): 29-50, 1983.

[11] "Integrating Perception and Problem Solving to Predict Complex Object Behaviors", by Damian M. Lyons, Mohamed Chaudhry, D. Paul Benjamin, Marius Agica, John Vincent Monaco, Conference on Multisensor, Multisource Information Fusion, SPIE, April 2010.

[12] "Robot Video Tracking by Comparing Real and Simulated Video Scenes", by Damian M. Lyons and D. Paul Benjamin, Conference on Intelligent Robots and Computer Vision, SPIE, San Jose, Calif., January 2009.

Biography

__Paul Benjamin__ is Professor of Computer Science and Director of the Robotics Lab at Pace University in New York City. After receiving his PhD from New York University, he worked in industry in robotics and as project leader in artificial intelligence and machine learning for six years before moving to academia. He has been the Principal Investigator on a number of grants from the NSF, AFOSR, ARO, DOE and DARPA. These projects include development of the ADAPT cognitive robot architecture in collaboration with Fordham University and Brigham Young University and the CSISM cybersecurity agent in collaboration with BBN Technologies. He holds patents in database security and intrusion detection. His research focuses on issues of problem solving and representation in cognitive architectures. He edited the first book in this area, "Change of Representation and Inductive Bias". He and his research group at Pace are developing cognitive agents in both robotics and cybersecurity.