JOURNAL OF AEROSPACE COMPUTING, INFORMATION, AND COMMUNICATION
Vol. 6, February 2009

Control of Mobile Robots Using the Soar Cognitive Architecture

Scott D. Hanford* , Oranuj Janrathitikarn† , and Lyle N. Long‡
The Pennsylvania State University, University Park, Pennsylvania 16802

DOI: 10.2514/1.37056

This paper describes the development of a system that uses computational psychology (the Soar cognitive architecture) for the control of unmanned vehicles. A multithreaded software system written using Java and integrated with the Soar cognitive architecture has been implemented on two types of mobile robots. Soar can be used as a general purpose robotic intelligence system and can handle a wide variety of sensor inputs and motor-control outputs. The use of existing computational psychology methods (such as Soar) may be a more efficient approach to developing robotic software, rather than developing new software for every new unmanned vehicle application. Results from the application of this software system (named the cognitive robotic system, or CRS) to a practical unmanned ground vehicle mission, navigating to a target GPS location while avoiding obstacles, are described. The CRS has been designed so that its capabilities can be expanded in the future through the inclusion of additional sensors and motors, additional software systems, and more sophisticated Soar agents.

I. Introduction

MOBILE robots and unmanned vehicles have continued to become more prevalent for a variety of civilian and military applications. Several well-known and highly successful unmanned vehicles (such as the Predator and GlobalHawk unmanned air vehicles), however, are essentially remotely piloted vehicles that possess very little intelligence or autonomy. As unmanned vehicles are required to perform more difficult missions or a larger number of missions in remote environments, intelligence and autonomy will become more important to their success. Autonomy has been described as the ability to operate "in the real-world environment without any form of external control for extended periods of time" [1]. Highly publicized examples of success in increasing the autonomous capabilities of unmanned vehicles were the 2005 Defense Advanced Research Projects Agency (DARPA) Grand Challenge, the 2007 DARPA Urban Grand Challenge, and NASA's Mars Exploration Rovers. Another DARPA program, the Learning Applied to Ground Vehicles program (LAGR), has encouraged the development of certain aspects of intelligence, particularly learning, for unmanned vehicles [2].

Intelligence can be thought of as "a very general mental capability that, among other things, involves the ability to reason, plan, solve problems, think abstractly, comprehend complex ideas, learn quickly and learn from experience" [3].

Received 22 February 2008; revision received 4 November 2008; accepted for publication 5 November 2008.

* NSF Graduate Research Fellow, Aerospace Engineering, sdh187@psu.edu, AIAA Student Member.
† Graduate Student Researcher, Aerospace Engineering, ozj100@psu.edu, AIAA Student Member.
‡ Distinguished Professor, Aerospace Engineering, Mathematics, and Bioengineering, lnl@psu.edu, AIAA Fellow.

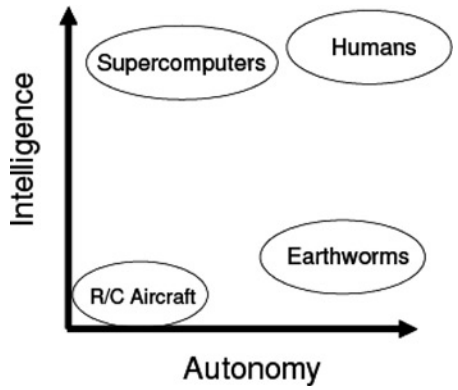

Fig. 1 Intelligence vs. autonomy for a few systems [4].

Figure 1 shows that intelligence and autonomy are two distinct characteristics: systems such as supercomputers have the capacity for intelligence but are incapable of autonomy because they are not embedded in the real world with sensors and motors. On the other hand, simple organisms such as earthworms are autonomous, but demonstrate minimal intelligence. Achieving intelligence in unmanned vehicles may be an even more challenging problem than achieving autonomy in these vehicles. This will happen, however, including the eventual robotic system that becomes self-aware or conscious [5].

A. Robotic Autonomy Architectures

Robotic control architectures are generally classified as reactive, deliberative or hybrid systems. Deliberative architectures (e.g. NASREM [6] and the Real-time Control System [7] (RCS)) are characterized by the sense-plan-act paradigm and the development of a world model, which the architecture uses to reason about the actions it should take. Reactive architectures (e.g. [8]) tightly couple sensors and motors and do not maintain a centralized model of their world. Some examples of hybrid systems, which combine characteristics of reactive and deliberative systems, are the AuRA [9], 3T [10], and ASE [11] architectures. Humans are essentially hybrid systems that can react quickly to stimuli such as pain, but can also reason and plan. Some examples of robotic control architectures that have been used for control of aerospace systems are described below.

The Jet Propulsion Lab has developed a three-layer control architecture called the Autonomous Science Experiment (ASE) [11] autonomously to collect data of scientific interest. The top layer uses the Continuous Activity Scheduling Planning Execution and Response [12] system for planning and the middle layer uses the Spacecraft Command Language [13] for translating tasks from the top layer into commands that the bottom layer of flight software can perform. The ASE architecture has been successfully implemented on the Earth-Observing One (EO-1) spacecraft to autonomously detect and observe interesting scientific events while overcoming challenges such as memory, power, and processing capabilities associated with deployment on a spacecraft.

NASREM [6] is a hierarchical control architecture designed for the space station Flight Telerobotic Servicer using the RCS [7]. NASREM has three hierarchical legs (sensor processing, world modeling, and task decomposition) which have access to a global memory database. Each leg of the hierarchy has a range of functions that vary in complexity from low-level actions such as motor control to high-level activity such as mission planning. The architecture can simultaneously make autonomous decisions and accept operator input at each level of the hierarchy.

3T [10] is a robot control architecture that uses three tiers or layers to combine deliberation and reactivity and has been used on several hardware platforms, including a prototype of a two-armed robotic manipulator for maintenance of a space station. The bottom, or skill, layer uses a skill manager to coordinate reactive skills to interface with the robot motors and sensors. The sequencing layer uses Reactive Action Packages [14] (RAPs) to control series of real-time behaviors or skills. The top layer is a planner that uses Adversarial Planner [15] to transform abstract goals into detailed subgoals that can be specified using RAPs or sequences of RAPs.

Of course there are many other robotic systems, many of which are described in well-known textbooks [1,16,17].

B. Computational Psychology Approaches to Intelligent Systems

In an effort to improve the intelligence and autonomy of mobile robots, the use of tools from computational psychology has become popular [18,19,20,21,22,23,24,25]. The most well-known approaches are called cognitive architectures. Newell proposed the development of cognitive architectures that would define the fixed processes that their designers believe were important for intelligent behavior [26]. Because "no theory in the foreseeable future can hope to account for all of cognition" [27], many cognitive architectures have been developed. These different architectures often focus on modeling different aspects of cognition and model cognition at different levels of abstraction. Although there are many examples of cognitive architectures, this paragraph briefly describes three of the most well-known architectures: Soar [28], ACT-R [27], and EPIC [29]. Soar and ACT-R were originally developed with a focus on modeling central cognition such as problem solving and learning. Soar uses a production system in which all of its knowledge about how to behave (production rules) and all of its knowledge about its current state are applied to demonstrate flexible and goal-driven behavior. The strengths of the Soar architecture include modeling decision-making at a level of abstraction very useful for performing high-level control of unmanned vehicles. Soar has historically been a symbolic architecture; however, the architecture has recently been extended to include subsymbolic mechanisms [30]. ACT-R (Adaptive control of thought-rational) models intelligent behavior using modules associated with regions of the brain and a central production system that has access to a subset of the information from the other modules. In addition to the symbolic processing that occurs in the production system, ACT-R also performs subsymbolic processes such as memory activation or assigning utility values to production rules. Models created using ACT-R have successfully modeled many aspects of human cognitive performance. The EPIC (Executive Process-Interactive Control) architecture has emphasized that, in addition to central cognition, perception and action have an important role in cognition and task performance. EPIC has detailed mechanisms for sensory information processing (visual, auditory, and tactile) and motor (ocular, vocal, and manual) activity. EPIC is capable of modeling multitasking behavior, which is especially useful in the area of human computer interaction [31]. It is also possible to integrate components from different cognitive architectures: for example, the EASE architecture integrated some of the strengths of ACT-R, Soar, and EPIC [32].

The ability of cognitive architectures to model intelligent behavior in diverse domains has resulted in interest in using these architectures for robotics. However, most of these architectures were not necessarily designed for use in the real-world environments that unmanned vehicles encounter. It can be particularly difficult to generate meaningful symbols for the symbolic components of the cognitive architectures to reason about from (potentially noisy) sensor data or to perform some low-level tasks important for unmanned vehicles such as control of motors using cognitive architectures. To deal with some of the challenges of using cognitive architectures to control unmanned vehicles in real environments, researchers have often integrated additional systems with the cognitive architecture that complement the strengths of the cognitive architecture. The next four paragraphs describe systems that have used cognitive architectures to control unmanned vehicles.

The Soar cognitive architecture has previously been used to control mobile robots and *simulated* unmanned vehicles [33,34,35]. Soar has been used with a mobile robot equipped with sonar sensors and a robotic gripper to locate and pick up cups [33]. Jones et al. have used the Soar architecture to autonomously fly U.S. military fixed-wing aircraft during missions in a *simulated* environment in the TacAir-Soar project [34]. The Soar agents in this project demonstrated autonomous and intelligent behavior in a complex, albeit *simulated*, environment with real-time constraints. Another project used Soar agents to control *simulated* air and ground vehicles [35].

The Intelligent Systems Section at the Navy Center for Applied Research in Artificial Intelligence has used multiple cognitive architectures in its efforts to build robots that can think and act like humans to facilitate better and easier collaboration between humans and robots [36]. ACT-R, ACT-R/S (an extension of ACT-R that includes a theory of spatial reasoning), and Polyscheme [37] (a cognitive architecture which can integrate several specialists to integrate different types of reasoning) have been used to perform research in the area of robots and spatial reasoning through the game of hide and seek and a perspective taking task. Additionally, Kennedy et al. [38] developed a robot capable of approaching or following a target (as in a reconnaissance mission) independently or collaboratively with humans through voice, gestures, and movement. To accomplish this task, which is greatly dependent of the robot's ability in spatial reasoning, a three-layer architecture was developed. The bottom layer included sensors, effectors, and a non-cognitive system for standard robotic tasks (path planning, obstacle avoidance, etc.). The middle layer was a spatial support layer and included a cognitive map. The top, or cognitive, layer included an ACT-R model.

ADAPT [39,40] is a robotic cognitive architecture that has been embedded in the Soar architecture by integrating sensorimotor activity using the RS (robot schemas) language, Soar and its theory of cognition, a theory of decomposition and reformulation, and a natural language system. The development of ADAPT has been motivated by the desire to address two inadequacies of cognitive architectures for robotics research: true concurrency and active perception. Concurrency allows the coordination of several behaviors (e.g., moving a robotic foot until it touches the ground requires simultaneous control of the motors required to move the foot and monitoring a sensor to determine when the foot has touched the ground). Active perception can be considered perception in pursuit of a goal. Two examples of active perception useful for a robot would be the selective processing of only the important areas of an image and adjusting the parameters of a camera to focus on a particular object in the robot's environment.

The Symbolic and Subsymbolic Robotic Intelligent Control System (SS-RICS) aims to integrate several successful AI techniques (symbolic and subsymbolic processing, machine learning, and semantic networks) for control of mobile robots [41,42,43]. SS-RICS has been built around ACT-R at the symbolic level and can use AI techniques at both the subsymbolic (neural networks) and symbolic (machine learning) levels. Subsymbolic processing is used to process all sensor inputs. An example of this processing is determining if points sensed using a laser rangefinder can be considered as a group that may indicate an important item (e.g., a corner or a wall) in the robot's environment [44]. Symbolic processing occurs in the form of a production system (similar to ACT-R and Soar) that has access to high-level symbolic memory that includes results from processing at the subsymbolic level.

The purpose of this paper is to describe the development of a system that uses the Soar cognitive architecture to control unmanned vehicles and to demonstrate the application of Soar to a practical unmanned ground vehicle mission, navigating to a target global positioning system (GPS) location while avoiding obstacles.

II. Soar for Unmanned Vehicles

The Soar cognitive architecture possesses several capabilities that make it a promising candidate for use in unmanned vehicles, including simple communication between the architecture and environment through many sensors and motors, a mix of reactive and deliberative behaviors, definition of a learning mechanism, and the ability to collaborate with other vehicles or software systems [45]. A Soar agent attempts to use its knowledge about its current situation, or state, to apply operators to change the state in an effort to reach its goal. Information about the current state of the Soar agent and its environment is stored in working memory, which can be thought of as a form of short-term memory. In a Soar agent, long-term knowledge is encoded as production rules. These rules include elaborations that add information to the current state of working memory and operators that make changes to working memory to move the agent closer to reaching its current goal. Operators can be organized into problem spaces that are relevant to some specific problem. For example, an agent could have a problem space called avoid-obstacle, in which operators describe how it should move if an obstacle is sensed that needs to be avoided.

The execution of a Soar agent is governed by a decision cycle (Fig. 2) in which the agent chooses an appropriate action to take based on the information it knows about its current situation (working memory) and its long-term knowledge (production rules). This cycle can be broken into five parts [46]:

1) In the input phase, Soar can receive information from a simulated or real external environment (e.g. sensor input).
2) In the second phase, the current state is elaborated and operators are proposed based on the contents of working memory.

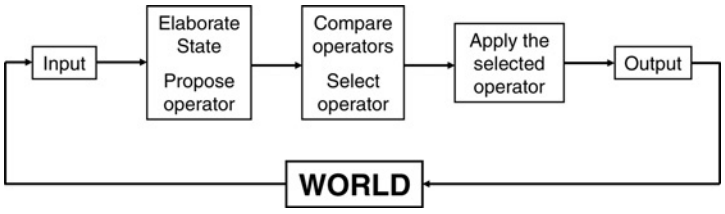

Fig. 2 The five stages of Soar's decision cycle for a Soar model that interacts with the external world [46].

3) In the third phase, one operator is selected. If more than one operator has been proposed in the second phase, Soar has a mechanism to attempt to choose the most appropriate operator for the current situation.
4) In the application phase, the selected operator makes changes to working memory that move the agent closer to achieving its current goal.
5) In the output phase, Soar can send information to its external environment (e.g. motors and servos).

If a Soar agent does not have enough knowledge to select the best operator in that phase of its reasoning cycle, an impasse occurs in which Soar automatically creates a subgoal to determine the best operator to select. This mechanism supports the hierarchical execution of complex goals (as abstract operators in subgoals) and allows Soar agents to both reactively select an operator (when sufficient knowledge is available to choose one) and deliberatively reason about which operator to select (when the agent does not have enough knowledge to reactively select one). Chunking, the learning mechanism in Soar, also uses the creation of subgoals by storing the results of the problem solving that is used to achieve the subgoal as a new production rule.

The Soar architecture has several characteristics that make it desirable for use in unmanned vehicles:

1) The creation of impasses and automatic subgoals support hierarchical decomposition of complex goals or abstract operators. This mechanism allows high-level abstract operators to be implemented in subgoals, in which a new state is created where simpler low-level operators (operators that create actions the Soar agent can perform; e.g. giving motor commands) are used to achieve the goal of the abstract operator [34]. The use of subgoals is convenient for Soar agents that have a large amount of long-term knowledge and are therefore capable of working on several types of problems or goals. For these agents, even though a large number of operators are likely to be eligible for proposal based on the agent's current state, only a small number of these operators are likely to be useful for accomplishing the agent's current goal. Organizing problem solving hierarchically using abstract operators and subgoals allows only operators relevant to the current goal to be considered. The use of abstract operators and subgoals in the Soar agent developed for this research will be discussed later.
2) The Soar Markup Language (SML) has been developed to allow simplified interaction of a Soar agent with external environments through sensors and motors [47]. Also, because SML can be used with popular programming languages such as Java and C++, integration with other software systems is possible.
3) The Soar architecture is also capable of handling the large agents important for intelligent autonomous behavior in complex environments: an agent in TacAir-Soar had 5200 production rules, 450 total operators, and 130 abstract operators that were used to fly U.S. military fixed-wing aircraft [34]. Soar uses the Rete algorithm, which is efficient for large numbers of working memory elements, to determine which of the agent's production rules have their conditions for firing matched based on the current contents of working memory [48].
4) Soar agents have also demonstrated collaborative behaviors in simulated external environments. TacAir-Soar agents performed flight missions that required interaction between agents through simulated radio systems [34]. Another project used Soar agents to control unmanned air and ground vehicles cooperatively to perform tasks such as communications relay, waypoint following, and conducting surveillance during simulated missions [35].
5) The Soar architecture can provide a mix of reactive and deliberate behaviors. Soar agents can use a combination of reactive selection of an operator and deliberative on demand planning that can occur when an agent does not have enough knowledge to select an operator.
6) Soar's learning mechanism (chunking) can store the results of on-demand planning, allowing the agent to be more reactive the next time a similar situation is encountered. However, chunking in an environment in which there is uncertainty in the Soar agent's model of this environment (for example, from noisy sensors) is a difficult problem because Soar does not have a simple mechanism for unlearning incorrect knowledge.

The above characteristics suggest that Soar is suitable for high-level control (such as reasoning and planning about how to solve problems) of unmanned vehicles. However, there are also characteristics useful for control of unmanned vehicles that do not match the strengths of Soar. For example, Soar is not well-suited to perform a large amount of numerical calculations, perform low-level control of motors, or solve optimization problems. Combining Soar with other software systems (including systems such as neural networks, robot control algorithms, state estimation

techniques, and object recognition methods) that complement the strengths of Soar could increase the usefulness of Soar for the control of unmanned vehicles.

This paper will describe the cognitive robotic system (CRS), which has been developed to use the Soar cognitive architecture for the control of unmanned vehicles. The CRS has been implemented on two unmanned ground vehicles for the problem of navigating to a GPS location while avoiding obstacles. Figure 3 shows how the CRS uses a Soar agent's decision cycle to make high-level decisions about how to control an unmanned vehicle. Starting in the bottom left of Fig. 3, sensors are used to obtain information from the world. The sensor information is sent to the Soar agent during its input phase. In the Propose Operators phase of the Soar decision cycle, Soar operators corresponding to the robot behaviors in the rectangular box (Fig. 3) are proposed in parallel; all operators that are appropriate for the current situation will be proposed. The current Soar agent can produce two behaviors: avoid obstacles and go to a GPS location. (Additional robot behaviors can be included by adding more knowledge, in the form of production rules, to the Soar agent.) After all of the appropriate operators have been proposed, Soar compares the priority level of these operators and selects one operator to apply. The propose, select, and apply phases are repeated until output is produced. The output from the Soar agent is used to determine the unmanned vehicle's action. The cycle continues with new information about the unmanned vehicle's world being obtained by the sensors.

The current version of the CRS focuses on using the symbolic processing capabilities of Soar to make high-level decisions. Currently, only minimal subsymbolic processing of raw sensor data is performed before information is sent to the Soar agent during its input phase. Additionally, simple algorithms are used in the 'Commands' box (Fig. 3) to use commands from Soar to move the robots. The rest of this paper describes the implementation of the CRS to use the Soar cognitive architecture on two types of unmanned ground vehicles to accomplish a mission of navigating to a desired GPS location while avoiding obstacles.

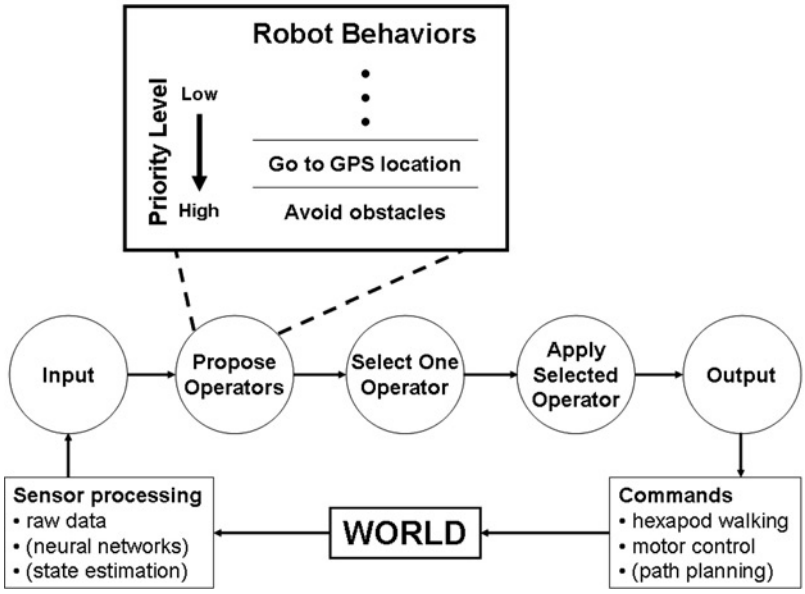

Fig. 3 A schematic of the cognitive robotic system. The execution cycle of the CRS includes a phase for sensor input from the world, the five phases of the Soar decision cycle (shown in circles), and a motor control phase. The knowledge needed to perform the robot behaviors listed in the rectangle above the "Propose Operators" circle is programmed as Soar production rules. The priority level of these behaviors is used by Soar to select a single operator if multiple operators have been proposed. In the 'Sensor processing' and 'Commands' boxes, examples of algorithms that have not been implemented yet, but could be in the future, are included in parentheses.

III. Hexapod Robotic Platform

A. Walking

Walking robots can have some advantages over their more common wheeled counterparts. Walking robots can traverse irregular terrain, walk up and down stairs, and step over small obstacles and holes. With advanced walking algorithms, walking robots can also travel over rough terrain at high speed while maintaining a smooth motion of their body. Legged robots may be able to move on soft ground where wheeled robots might slip and legged robots can also be designed to disturb the ground less than wheeled robots. In addition, walking robots can sometimes maneuver around small spaces better than wheeled robots [49].

Legged robots can be categorized based on their number of legs [50]. Two-, four-, and six-legged systems are more typical than one-, three-, five-, and eight+ legged systems [51]. Hexapods (six-legged systems) have taken biological inspiration from insects. Although having more legs than two-legged (bipeds) and four-legged (quadrupeds) robots requires hexapod robots to have more hardware, hexapods benefit from having more desirable stability properties than bipeds and quadrupeds [51]. The typical six-legged gait is a tripod gait, in which the front and back legs of one side and the middle leg of the other side make up one tripod. One tripod remains in contact with the ground while the other tripod is moving [52]. As a result, the tripod gait is both statically and dynamically stable while one-leg-standing bipeds or two-leg-standing quadrupeds are only dynamically stable. To focus on the software for intelligent and autonomous control, a hexapod was used for this research because of its beneficial stability properties.

B. Hexapod Hardware

The hexapod robot used in this research is the HexCrawler, a robotic kit from Parallax Inc. (www.parallax.com). This hexapod, shown in Fig. 4, has dimensions of 20″ by 16″, is 6″ tall when standing and 4.9″ tall when squatting, and has a ground clearance of 3.5″. The hexapod weighs 4 lbs and its payload capacity is 7.5 lbs. Each leg of the hexapod is controlled by two servo motors: one that allows motion in the horizontal plane and one that allows motion in the vertical plane.

A small PC, the VIA EPIA Mini-ITX M-Series (www.via.com.tw/enboard), has been installed on the hexapod. This PC is less than 7″ by 7″ and has a 600 MHz VIA Eden processor. The VIA board has many input/output devices,

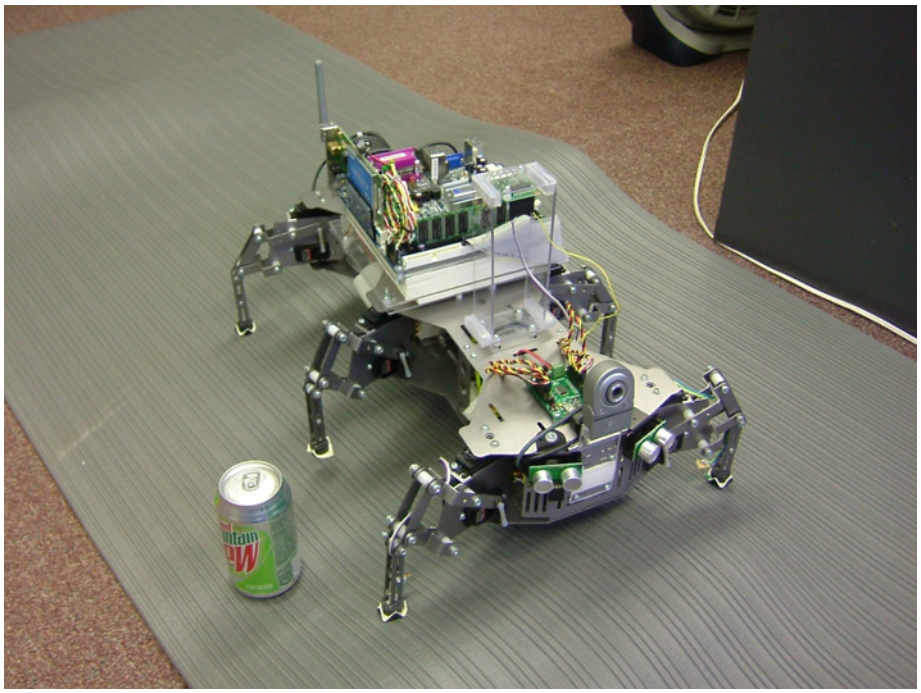

Fig. 4 A picture of the hexapod next to a can of soda.

including USB, serial, and I2C (IIC) ports. Memory (512 MB), a hard disk drive, and a wireless card were added to the board. The operating system used on the Via board is Windows XP.

The Parallax Servo Controller (PSC) with a USB interface is used to control each of the twelve servo motors on the hexapod legs. The PSC-USB board is 1.8″ by 2.3″ and can control servos with a range of 180 degrees of rotation and with a resolution of 0.18 degrees.

The BrainStem GP 1.0 module (www.acroname.com) is used as an interface between several of the sensors and the onboard PC. The BrainStem is a 2.5″ by 2.5″ board with multiple input/output capabilities. The board has a 40 MHz RISC processor (PIC 18C252 microcontroller), four outputs to servo motors, five 10-bit analog inputs, five digital input/output (I/O) pins, two I2C ports, one serial port, one GP2D02 connector (for use with IR sensors), and power connections for servo motors and the onboard logic (the microcontroller and analog and digital pins). The BrainStem module is also capable of running up to four Tiny Embedded Application (TEA, a subset of the C programming language) programs concurrently.

Currently, the hexapod senses its environment using two sonar sensors, an electronic compass, and a GPS receiver. Figure 5 shows the physical connections of the hardware currently used on the hexapod [53]. Janrathitikarn and Long have also investigated the use of touch sensors on the hexapod's feet [54]. The sonar sensors, the Devantech SRF05 Ultrasonic Ranger (www.robot-electronics.co.uk), can measure the distance to the closest object in its beam pattern with a range of 3 cm to 4 m. This sonar sensor has an operational mode that allows a single digital pin to be used to both initiate the sonic pulse and measure the echo, reducing the number of digital I/O pins that must be used to interface this sensor with the BrainStem.

A Devantech CMPS03 Electronic Compass by Robot Electronics provides measurements of the hexapod's heading relative to the Earth's magnetic field. Because the electronic compass operates by using the Earth's magnetic field, it is sensitive to all electromagnetic fields in its environment, including those from other electronic devices, metallic objects, etc. Consequently, the compass must be carefully mounted on the robot at a position which is minimally affected by the electromagnetic fields of other devices. The I2C capability of the BrainStem module was used to communicate with the compass and receive the heading of the hexapod with a range of 0 to 359.9 degrees.

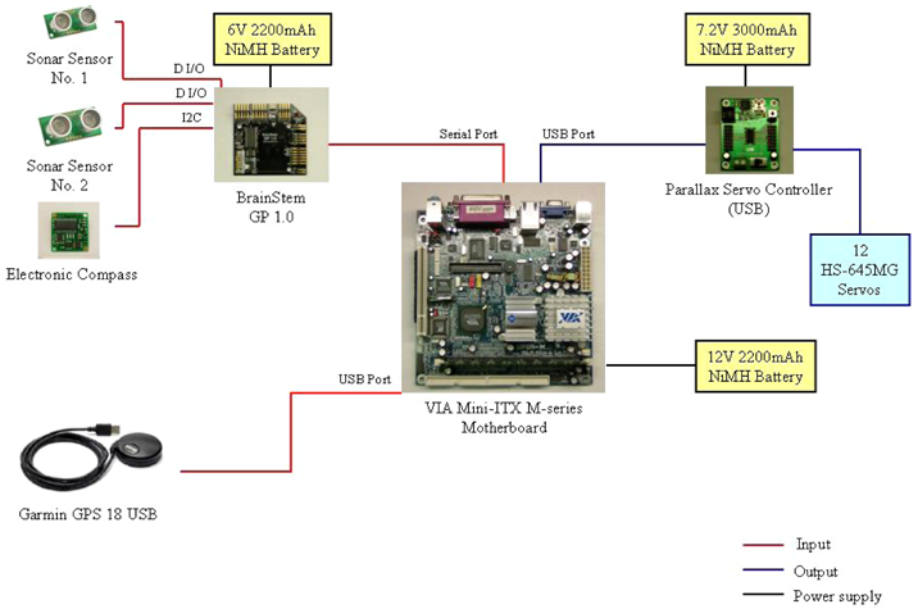

Fig. 5 A schematic of the hardware used onboard the hexapod and their connections. Inputs to the BrainStem module and VIA Motherboard are shown as red lines. Output from the VIA Motherboard and Parallax Servo Controller are shown by the blue lines. The black lines represent power supplied to the BrainStem module, the VIA Motherboard, and the Parallax Servo Controller [43].

The GPS 18 USB (www.garmin.com) is a 12-parallel-channel, WAAS-enabled GPS receiver that can receive information from up to 12 satellites to update the current latitude and longitude of the receiver. This GPS receiver connects to the VIA board using a USB 2.0 port. Outputs from the GPS receiver are sent to the VIA board in the standard NMEA sentence format through a virtual communication port using the interface software from Garmin called "Spanner."

C. Software for Hexapod

Several software packages have been installed on the hexapod's VIA board computer [42]:

1) Java 2 Standard Edition (J2SE) 5.0
2) Java Communications API (javax.comm)
3) Soar (version 8.6.3, including SML)
4) BrainStem acroname Java package
5) GPS software

1. TEA Code

The BrainStem is capable of running four TEA programs concurrently using a virtual machine on the BrainStem module. The TEA language was designed for small programs that simplify the applications on the host computer (the VIA board in our project) that communicate with the BrainStem module. For the current hexapod system, two TEA programs are run concurrently that continuously record measurements from the two sonar sensors and electronic compass and place the results in an area of memory (called the scratchpad) on the BrainStem that can be accessed by both the TEA programs and applications on the host computer.

2. Java Code

The Java programming language is used as middleware between the hardware and the Soar architecture. Java is an object-oriented programming (OOP) language, which is very useful for maintaining and extending software systems. Java is platform independent so the same Java code can run on all machines using the Java Virtual Machine (JVM). Java also has a multithreading capability that allows concurrent computing, which is used in the CRS. Although Java does have some characteristics that are not desirable for real-time applications (automatic garbage collection and lack of an accurate real-time clock), the Real-Time Specification for Java (RTSJ) and Sun Java Real-Time System 2.0 (Java RTS) provide promise for future real-time Java systems. The RTSJ was not used for the current implementation of the CRS, but could be used in future versions.

Several Java classes and packages to interface with the hardware and software systems onboard the hexapod have been developed using the Eclipse Integrated Development Environment (IDE).

1) Several Java classes are used to receive data from the GPS unit and process the data before it is sent to the Soar agent. The WayPoint class stores data such as the current and target GPS positions and calculates the vector to the target from the current position. The GPSConnection class runs as a separate thread that establishes a connection to the GPS receiver using a virtual communication port, acquires the GPS data, extracts the latitude and longitude data, and sends it to the GPSConnectionReader thread. After receiving the GPS data, the GPSConnectionReader thread stores it in a WayPoint object as the current robot position so the vector from the current position to the target position can be calculated by the Waypoint object. The results of these calculations can be accessed anytime by the main Java thread.
2) Acroname, Inc., the BrainStem manufacturer, provides a Java package called acroname for communicating with BrainStem modules from a PC. This package includes a class with low-level methods for interacting with BrainStem hardware such as reading and writing to digital pins and reading and writing to the scratchpad memory on the BrainStem module. An additional Java class called BrainStem was written to control all BrainStem interactions by using the low-level methods included in the acroname package. The BrainStem class is also implemented as its own thread. In this thread, communication is established between the Brainstem module and the PC, the two TEA programs for reading the sonar sensors and electronic compass are started on the BrainStem module, and results from the sonar sensors and electronic compass are read from the scratchpad memory on the BrainStem.

3) The WalkingRobot class is implemented in a thread that communicates with the Parallax Servo Controller to control the movement of the hexapod. A WalkingRobot object has a Hexapod object, which in turn has six Leg objects. Each Leg object has two Servo objects. The WalkingRobot class has methods such as forward(), turn-left(), etc. that control each leg's servos using specified sequences of servo motor movements to perform the desired hexapod motion.
4) The SoarSML class provides methods for communicating with Soar using the Soar Markup Language (SML). SML can be used to perform all the steps necessary to use Soar from Java, including creating a Soar kernel and agent, loading the Soar production rules, defining the structure of the input link, updating the values of Soar's input link, running the Soar agent, and retrieving commands from Soar's output link.

3. Soar Production Rules and Operators

The production rules for the Soar agent include elaborations, which update information about the agent's state, and those which propose, compare, and apply operators.

The Soar elaborations for this agent use information (e.g., from the input link) to "create" new descriptions of the agent's state that can be useful for the proposal, selection, and proposal of operators. These elaborations are selected independently of the current operator and take information from Soar's input link (e.g. data from sensors) and determine things such as the presence of obstacles, the current distance and heading to target, and the difference between the current heading of the hexapod and the heading to the target. Jones et al. called this the implicit goal of maintaining situational awareness [34]. For the current implementation of the CRS, these elaborations function as a simple state estimator in the Soar agent.

The Soar agent developed to go to a target GPS location while avoiding obstacles implements two abstract operators: avoid and goToTarget. The avoid operator is proposed when at least one sonar sensor detects an obstacle and the goToTarget operator is proposed when no obstacles are detected by the sonar sensors. When the avoid or goToTarget abstract operators are selected, the Soar agent implements them in subgoals, where a new state is created and named avoid or goToTarget, respectively. In these subgoals, relevant low-level operators can be proposed, selected, and applied to accomplish the goal of the selected abstract operator. The hierarchy of operators for this Soar agent is shown in Fig. 6.

In the avoid subgoal, the turn-left and turn-right low-level operators are eligible to be proposed and are used to avoid obstacles sensed using the sonar sensors. If the hexapod senses an obstacle on its left, the turn-right operator is proposed. If the hexapod senses an obstacle on its right, the turn-left operator is proposed. If obstacles are sensed to the left and right, both the turn-right and turn-left operators are proposed and one operator will be selected by Soar at random (a more sophisticated Soar agent could use information about its environment to help select an operator instead of choosing one randomly). The avoid abstract operator (and therefore the avoid subgoal) will no longer be applicable (conditions for proposal of avoid operator will not be met) when no obstacles are present. Because no obstacles are present, the goToTarget abstract operator will be proposed.

In the goToTarget subgoal, the turn-left, forward-left, forward, forward-right, and turn-right operators are eligible to be proposed. The proposal of these five operators depends on the difference between the heading required to go to the target location and the current heading of the hexapod. This value will be referred to as the deltaAngle. Two more variables used in the proposal of these five operators are angleRange1 and angleRange2. These variables represent

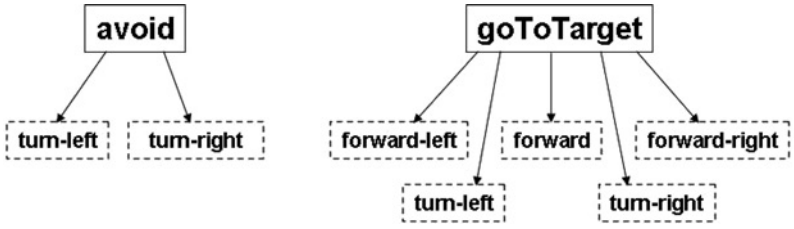

Fig. 6 The hierarchical structure of the avoid and goToTarget abstract operators and their low-level operators is shown. The abstract operators are shown in the boxes with solid borders and the low-level operators are shown in boxes with dashed borders.

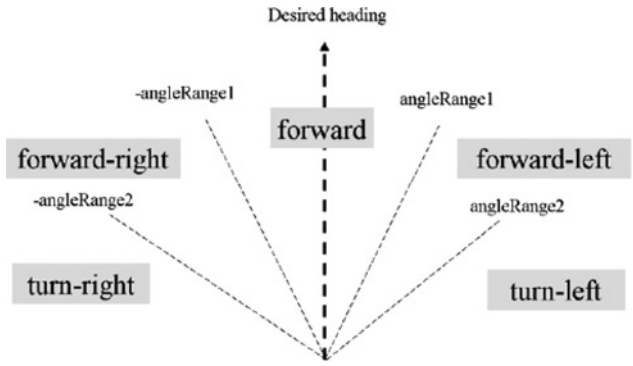

Fig. 7 This figure shows the decision criteria used to propose the five operators in the goToTarget subgoal. The desired heading is shown to be straight up and has a value of zero in this figure. If the hexapod's heading, and therefore the value of deltaAngle, is greater than –angleRange1 and less than angleRange1, the hexapod's heading is close to the desired heading and the forward operator will be proposed. If the value of deltaAngle is greater than angleRange1 and less than angleRange2, the forward-left operator will be proposed. If the value of deltaAngle is greater than angleRange2, the turn-left operator will be proposed. If the value of deltaAngle is less than –angleRange2, the turn-right operator will be proposed. If the value of deltaAngle is greater than –angleRange2 and less than –angleRange1, the forward-right operator will be proposed.

threshold values that, along with the value of deltaAngle, determine which of the five operators will be proposed. For example, if the value of deltaAngle is less than the value of angleRange2 and greater than the value of angleRange1, the forward-left operator will be proposed. Figure 7 shows the ranges of values of deltaAngle for which each of the five operators will be proposed. In addition, the forward operator is always proposed in case the selection of a turn-operator would undo a recent action of the robot.

The Soar agent also keeps a history of the last ten operators selected during its execution. The last ten operators are recorded so that these operators will not be selected again unless they are the only operators that have been proposed. Recording the last ten operators becomes beneficial when the abstract operator changes. For example, if the hexapod is moving forward in the goToTarget subgoal and detects an obstacle ahead and to the right, the avoid operator will be selected. In the avoid subgoal, the hexapod will turn left until the obstacle is no longer detected. At this point, the goToTarget problem space will be selected again. If the hexapod did not have any knowledge of its past operators, it would attempt to turn right to accomplish its goal of reaching the target. However, this could cause the hexapod to encounter the obstacle it just tried to avoid. With the memory of its last ten operators, the hexapod will go forward, waiting to turn to the right to go to the target until ten operators have been chosen because the turn left operator was chosen in the avoid subgoal. This recording of operators can help the hexapod get away from an obstacle it has avoided and maybe even escape from some cases of cul-de-sac environments.

D. System Integration

During execution of the hexapod's missions, four Java threads are concurrently running to: 1) receive GPS data, 2) acquire data from the BrainStem, 3) send output to the PSC Board, and 4) control all processes including the interface with Soar. Figure 8 shows the data flow between all software and hardware devices in the system. The hardware is shown on the bottom layer and includes three devices that communicate with Java threads: the GPS sensor, the BrainStem module, and the PSC board. The software in the CRS has primarily two layers: Java and Soar (there are also two TEA programs running on the BrainStem module). The main thread, WalkingRobotAgent, coordinates communication between five classes: GpsConnectionReader, BrainStem, WalkingRobot, SoarSML, and Waypoint. The GpsConnectionReader, BrainStem, and WalkingRobot classes are threads that are created when the main thread is started and the SoarSML and Waypoint classes are objects executed in the main thread. More details, including class and sequence diagrams, are available in [53].

Measurements from the two sonar sensors and the electronic compass connected to the BrainStem module are received in Java using the BrainStem class. Information from the GPS unit is sent to the GpsConnectionReader class

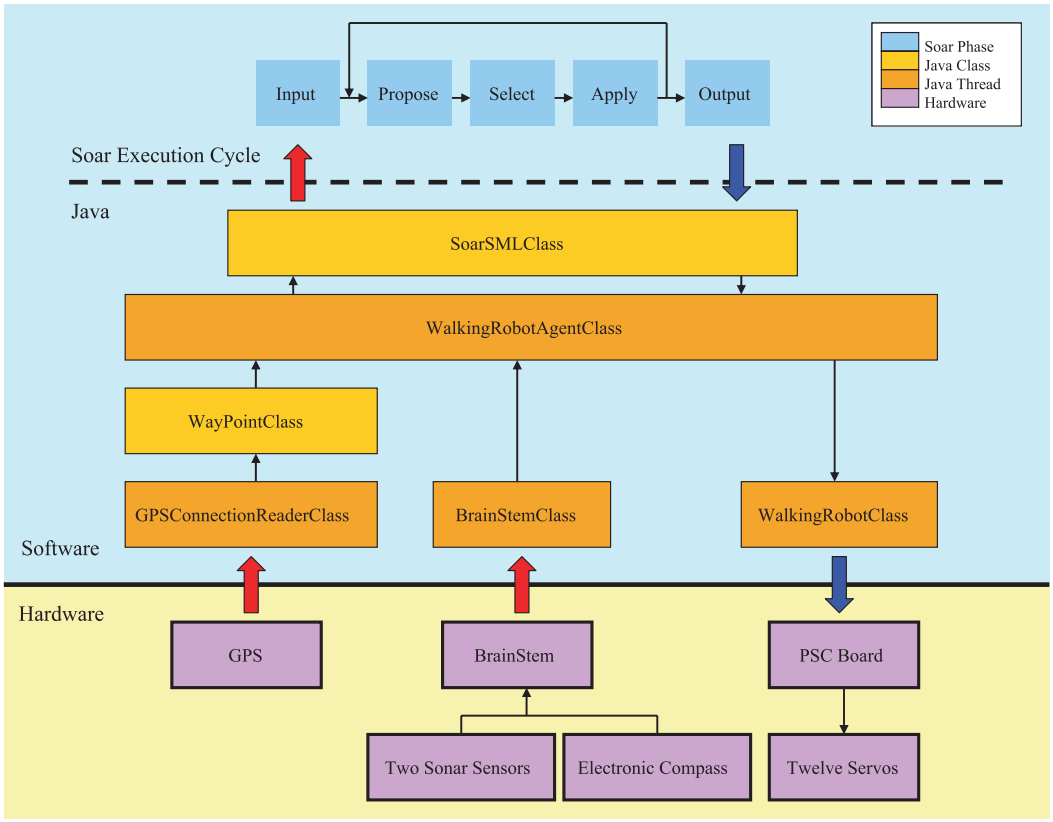

Fig. 8 The data flow between all hardware and software components of the CRS is shown.

and then to the WayPoint class, where it is processed. In the WalkingRobotAgent class, the SoarSML class is used to send the sensory information from the Waypoint and BrainStem classes to the Soar agent during its input phase. The SoarSML class then tells the Soar agent to run its decision cycle (propose, select, and apply) until it produces output. While the Soar agent runs its decision cycle, information is being obtained from sensors using the TEA programs running on the BrainStem module and other Java threads are processing sensor data. The SoarSML class then retrieves the output and sends it to the WalkingRobotAgent class. This output is then sent to the WalkingRobot class, which gives motor commands to the PSC board that controls twelve servos on the hexapod.

E. Results

Experiments were performed with three different types of obstacles to test whether the CRS could control the hexapod to navigate to a GPS location while avoiding obstacles. These tests were performed at the top level of a parking garage at the Pennsylvania State University. A picture of the testing environment is shown in Fig. 9 and a schematic of the parking lot is depicted in Figs. 10–12. The thick black line in the figures represents the walls on this level of the parking garage. In addition to the walls, there is a stairway from the next lower level in the bottom right of the figures that provided another obstacle to the hexapod. The lower half, the far left of the upper half, and the far right of the upper half of the figures (dark yellow area) are level. The upper-left part of the figures is a ramp up to the additional parking spaces. The upper-right part of the figures is a ramp down to the lower level of the parking lot. The red 'X' in the lower right corner indicates the target GPS location. For these tests, the target was considered to be reached when the hexapod was within 3 m of the specified GPS location.

For the first test (Fig. 10), the hexapod was placed in the narrow (~0.8 m wide) space between the stairway located in the lower right lower corner of the figure and the outer wall of the parking garage. The robot tried to adjust its heading to the target, but it detected the stairway on the left and processing in the avoid problem space instructed the

Fig. 9 A picture of the top level of the parking garage on which the tests described in this paper were performed. This picture was taken from a location in the lower left corner of Figs. 10–12 and 17.

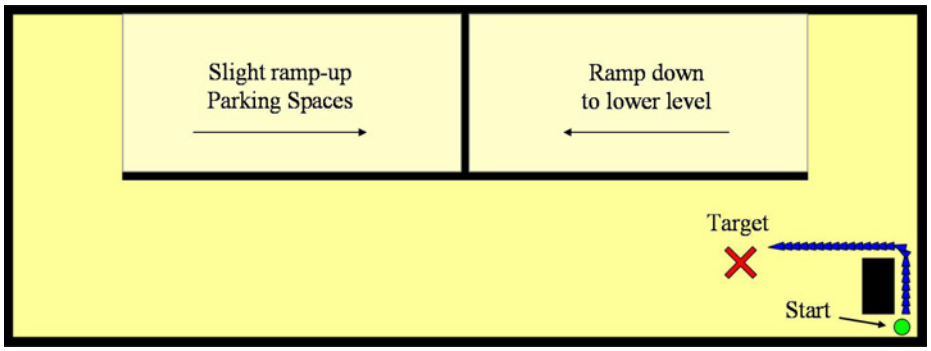

Fig. 10 The hexapod started this test at the green circle, went through the narrow path between the stairway and the outer wall, and reached the target location represented by the red 'X.'

robot to turn right. The hexapod walked forward for ten Soar decision cycles until the turn-right operator was deleted from the hexapod's memory of its previous operators. The goToTarget problem space commanded the robot to turn left to the target again. If it turned left and still sensed the obstacle of the stairway, the avoid subgoal was selected. This process was repeated until the robot walked out of the narrow space between the stairway and the outer wall of the parking garage. When the hexapod turned left and did not sense the stairway, it successfully walked to the target location.

In the next test, the hexapod started from the lower left corner of the parking lot (Fig. 11). There were eleven obstacles placed between its starting position and the target. The obstacles were placed far enough apart so that the hexapod could move between any of the obstacles. At the start of the test, the hexapod detected no obstacles and headed to the target. When an obstacle was detected in front of the hexapod, the avoid operator was selected and the hexapod was able to turn left away from the obstacle. After the obstacle was no longer detected, the hexapod re-entered the goToTarget problem space. The Soar agent alternated between the avoid and goToTarget problem spaces until the hexapod successfully reached the target location.

The results of the third test are shown in Fig. 12. The hexapod started in the upper left corner and moved to the target in the goToTarget problem space until it sensed an obstacle (the wall in the middle of the upper part of Fig.12).

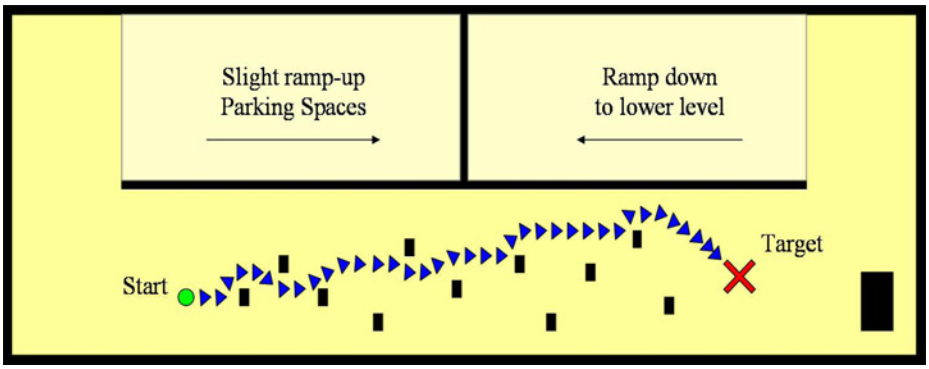

Fig. 11 The results of the second test are shown. The hexapod started in the lower left corner of the figure at the green circle and was able to successfully avoid the obstacles and reach the target location by switching between the avoid and goToTarget problem spaces.

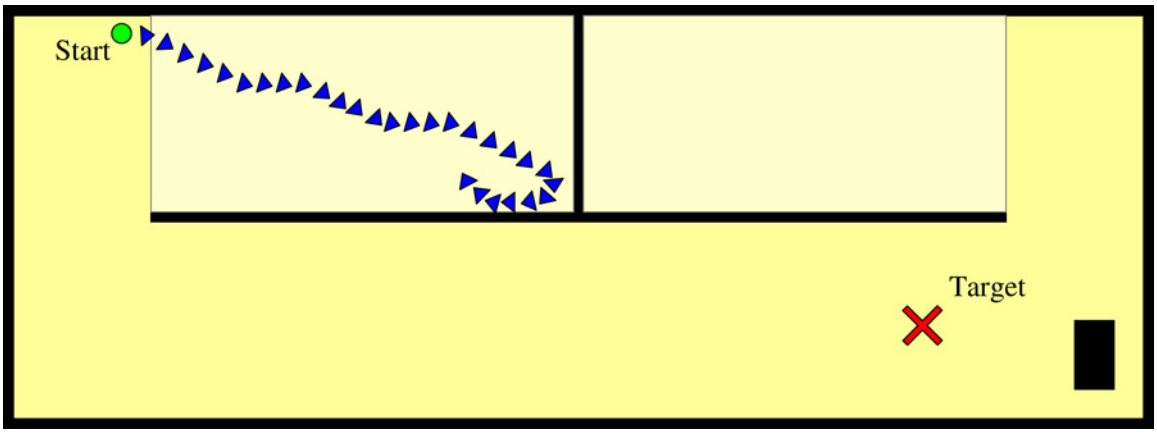

Fig. 12 The results of the third test are shown. The hexapod started in the upper left corner of the figure at the green circle and was not able successfully to reach the target location because it became stuck in the cul-de-sac in the upper left corner of the figure.

The hexapod then turned right in the avoid problem space. When it no longer detected the obstacle, it returned to the goToTarget problem space. The target was then behind and to the right of the hexapod so the hexapod turned to the right. For clarity, the path of the hexapod for the rest of the test is not shown, but the hexapod attempted to walk to the target until it encountered the wall in the middle of the upper part of Fig. 12 again. The hexapod then again used the avoid problem space to move itself out of the corner. The hexapod repeated this pattern of behavior, continuing to move into and out of the corner of the cul-de-sac in the upper left corner of Fig. 12, until the test was terminated unsuccessfully.

These tests have shown that the Soar agent is capable of guiding the hexapod to a GPS target location with relatively simple obstacles. However, the agent was not able to control the hexapod to a target location in the presence of more complex obstacles, such as a cul-de-sac. Additions to the CRS in the form of additional sensors and more sophisticated behaviors allow the robot to successfully complete the mission in the presence of a cul-de-sac, as described in Sec. IV.G.

The current target accuracy of three meters was chosen after testing the accuracy level of the GPS unit at the parking garage used for the tests. However, this GPS unit may not provide enough accuracy for some practical applications. Although the level of accuracy could be increased through the use of expensive differential GPS, it may be more interesting to supplement the current GPS with a vision system. Instead of simply giving the hexapod the

mission of navigating to a target GPS location, the mission could be to find an object of interest at a GPS location. The hexapod could then navigate (within the accuracy of the GPS unit) to the GPS location of the object and then use its vision system to identify and finish walking to the object. Vision systems with some object identification abilities could also be of great use in indoor environments, where GPS cannot be used, to locate landmarks such as doors that are useful for navigation in indoor environments [43].

IV. SuperDroid Robotic Platform

In an attempt to demonstrate the applicability of the CRS to different types of unmanned vehicles, the CRS was implemented on a wheeled robot called the SuperDroid. Also, to achieve more successful results on the cul-de-sac test shown in Fig. 12, extensions were made to the CRS in the form of additional sensors and a more knowledgeable Soar agent.

A. Hardware for the SuperDroid platform

The SuperDroid (Fig. 13) has an aluminum base which was purchased from SuperDroid Robots (www.superdroidrobots.com). With 6.75″ diameter wheels, the robot has a width of 21″ wide and a length of 21.5″.

Most hardware components used for the implementation of the CRS on the SuperDroid (Fig. 14) are the same components used on the hexapod and have been described in Sec. III.B. The hardware components used on both the SuperDroid and hexapod are the BrainStem GP 1.0 module, the sonar sensors, the electronic compass, and the GPS receiver.

Instead of the Servo Controller Board used on the hexapod, the MD22 Motor Controller from Devantech (www.robot-electronics.co.uk) was used to control the SuperDroid's motors. This motor controller has a mode of operation where it is able to receive a steer command and a throttle command from the servo motor outputs on the BrainStem. Two independent motor commands (voltages) are sent from the MD22 Motor Controller: one command goes to the two motors on the left side of the SuperDroid while the other command is sent to the two motors on the right side of the SuperDroid. (With an additional MD22 controller, all four motors could be given independent commands.)

A laptop running Windows XP was used on the SuperDroid instead of the VIA board used on the hexapod. Several laptops have been used to control the SuperDroid; the most recent is a Lenovo ThinkPad with an Intel Core2 Duo 2.0 GHz processor and 2 GB of RAM.

In addition to the sensors used on the hexapod, a GP2D12 infrared distance sensor from Sharp was placed on each side of the SuperDroid. These sensors output an analog voltage proportional to the measured distance to the closest obstacle within the sensor's range of 10-80 cm. These sensors are interfaced with the BrainStem module using its analog input pins.

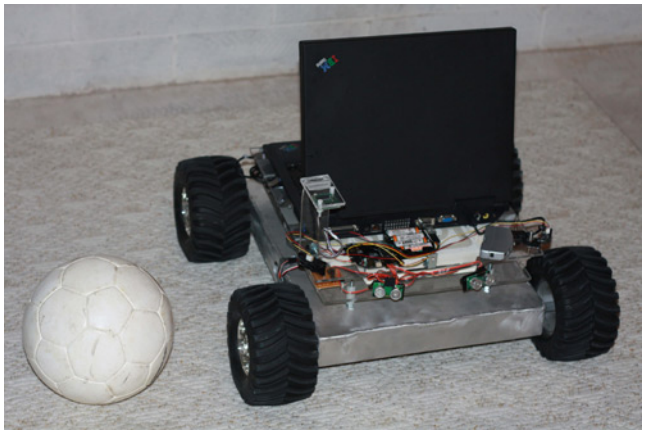

Fig. 13 A picture of the SuperDroid robot, shown next to a soccer ball.

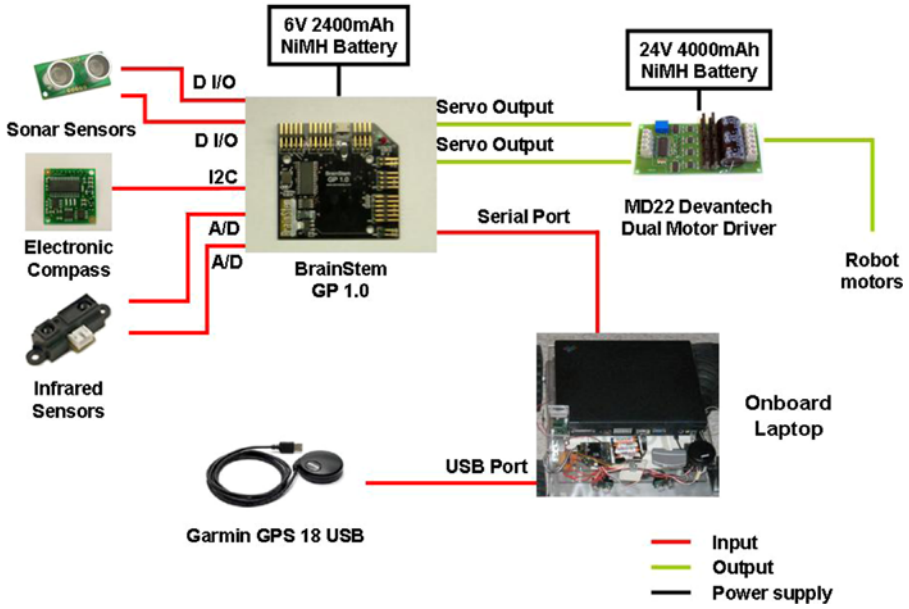

Fig. 14 A schematic of the hardware used onboard the SuperDroid and their connections. Inputs to the BrainStem module and onboard laptop are shown as red lines. Output from the onboard laptop and Devantech Motor Driver are shown by the green lines. The black lines represent power supplied to the BrainStem module and the Devantech Motor Driver.

B. Software for the SuperDroid

1. Java and TEA Programs

Much of the software used on the SuperDroid is the same as or very similar to the software used on the Hexapod. The same software packages that were installed on the VIA board on the hexapod (Sec. III.C.) were installed on the laptop on the SuperDroid. The same TEA code used on the BrainStem module on the hexapod to obtain measurements from the two sonar sensors and electronic compass was also used on the BrainStem module on the SuperDroid.

Many of the Java classes and packages used on the hexapod were also used on the SuperDroid. The Java classes used on the Hexapod to interface with the GPS receiver (WayPoint, GPSConnection, and GPSConnectionReader) were used again. The BrainStem class and acroname package were also used. In addition to their functions for the Hexapod, this class and package were also used to send motor commands from the BrainStem module to the MD22 motor controller. Because the BrainStem class and acroname package were used to send motor commands, the WalkingRobot class was not used on the SuperDroid. The SoarSML class described in Sec. III.C. was used for the SuperDroid, with the added capability of sending information about the measurements from the infrared sensors to the Soar agent.

2. Soar Agent

The revised Soar agent expands upon the Soar agent described in Sec. III.C. by using additional abstract operators and sensor information to navigate to a target GPS location while avoiding more complex obstacles than the original Soar agent could. There is a new abstract operator, followWall, to help the robot navigate to a target location in the presence of complex obstacles, including cul-de-sacs. In addition to the sensors used by the Soar agent described in Sec. III (two sonar sensors, compass, and GPS receiver), the new Soar agent also used two infrared distance sensors to measure the distances to the closest objects on the right and left sides of the robots. These infrared sensors are used to detect obstacles and, in the followWall abstract operator, to measure the distance to the wall being followed. Elaborations were also added to the Soar agent to determine if objects were close enough to the infrared distance sensors to be considered obstacles that should be avoided.

1) Remembering recently applied operators

The new Soar agent also remembers more about the operators it has previously applied than the agent described in Sec. III.C., letting the Soar agent consider in more detail what it has done recently. This agent records the names of the last 25 operators (forward, turn-right, etc.) it has applied, what subgoal they were applied in (goToTarget, avoid, etc.), and what actions (turn command, throttle command) were associated with the operators. These 25 recorded operators are used for several reasons:

a) The most recent operator that caused the robot to turn left or right (if it has been one of the five most recently applied operators) is recorded as the last-turn-operator. If no operators that cause the robot to turn have been applied in the last five operators, no last-turn-operator is recorded. The information associated with last-turn-operator is used to avoid undoing recent actions, similar to how the earlier Soar agent used the record of its last ten operators.

b) The most recent operator applied in an avoid problem space is called the last-avoid-operator and is recorded for up to 10 cycles (if none of the 10 most recent operators have been applied in the avoid problem space, there is not considered to be a last-avoid-operator).

c) The record of the 25 most recent operators is also used for recognizing when obstacles are being seen repeatedly (e.g., walls or some other obstacle or group of obstacles that the robot is having trouble avoiding) on one side of the robot. Within the record of these applied operators, the Soar agent looks for a sequence of two operators in which the first operator was applied in the goToTarget subgoal and had an action of turning the robot and the immediately following operator was applied in the avoid subgoal (indicating that the performance of the operator in the goToTarget subgoal resulted in seeing an obstacle). If there are three consecutive occurrences of this sequence with the same turn command (i.e. all three occurrences have the operator applied within the goToTarget subgoal tell the robot to turn in the same direction), the Soar agent assumes that it is having trouble avoiding an obstacle such as a wall or a group of smaller obstacles and uses this assumption in its decision making (and proposes the followWall abstract operator to avoid the obstacle).

2) Operator hierarchy

The updated Soar agent uses three abstract operators at the top-level of the hierarchy (the followWall abstract operator also has two abstract operators) to move to a desired location: avoid, goToTarget, and followWall (Fig. 15). The avoid abstract operator is proposed when at least one sonar or infrared sensor detects an obstacle and the robot is facing towards the target (it has a heading within 90 degrees of the desired heading). The goToTarget operator is proposed when no obstacles are detected by the sonar and infrared distance sensors. The followWall abstract operator is proposed for two different situations: 1) the robot is repeatedly encountering an obstacle on one of its sides or 2) the robot senses an obstacle while facing away from the target (it has a heading greater than 90 degrees away from the desired heading). The followWall behavior has higher priority than goToTarget if both are proposed at the same time.

3) The goToTarget subgoal

The conditions for proposal and application of the low-level operators in the goToTarget subgoal are the same for the current Soar agent as in the goToTarget subgoal for the Soar agent described in Sec. III.C.

4) The avoid subgoal

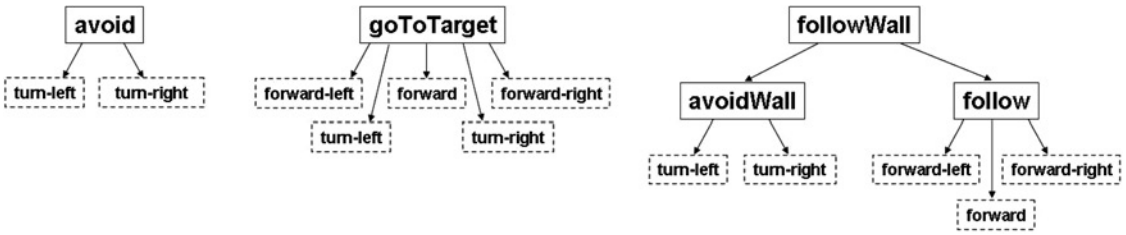

Fig. 15 The hierarchical structure of the operators and their low-level operators is shown. The abstract operators are shown in the boxes with solid borders and the low-level operators are shown in boxes with dashed borders.

Similar to the avoid subgoal for the Soar agent described in Sec. III.C., turn-right and turn-left operators are eligible to be proposed in the avoid subgoal for the updated Soar agent. However, the avoid subgoal for the current agent also uses information from the infrared sensors. If either the left sonar sensor or the left infrared sensor detect an obstacle that needs to be avoided, the turn-right operator is proposed. Likewise, if either the right sonar sensor or the right infrared sensor detects an obstacle that should be avoided, the turn-left operator is proposed.

When both the turn-right and turn-left operators have been proposed, the current Soar agent reasons about which operator to select instead of choosing at random as the original Soar agent did. First, the Soar agent considers if obstacles were sensed on either side of the robot by the infrared sensors: if there is an obstacle on one side of the robot but not the other, the robot will choose the turn operator that results in turning away from that obstacle. If both or neither infrared sensors sense obstacles on the sides of the robot, the Soar agent then uses its memory of the last-avoid-operator, if there is a last-avoid-operator recorded, to choose the turn operator by selecting the operator that is the same as the last-avoid-operator. At this point, if the Soar agent still has not been able to make a choice between the turn operators, the turn-right or turn-left operator is selected randomly.

5) The followWall subgoal

Once the followWall operator has been selected and the Soar agent is in the followWall subgoal, the agent must decide if it should follow a wall on its right or left; this decision is made by considering which side of the robot the sensed obstacle is on that resulted in the selection of the followWall operator. In the followWall subgoal, the Soar agent attempts to keep the robot a set distance from the closest obstacle sensed by the infrared distance sensor on the side of the robot on which the wall is being followed.

The Soar agent uses two abstract operators to accomplish its goal of following a wall. The first is the avoidWall operator, which is proposed if either of the sonar sensor senses an obstacle. In this avoid subgoal, the turn-right and turn-left operators are used to turn away from the side of the robot on which the wall is being followed. For example, if the robot is following a wall on its right side and senses an obstacle using either of its sonar sensors, the Soar agent instructs the robot to turn left, away from the wall it is following. If no obstacles are detected by the sonar sensors, the follow abstract operator is proposed. In the follow subgoal, the infrared distance sensor on the side of the wall being followed (left side sensor for left wall following, right side sensor for right wall following) is used to determine the distance between the robot and the closest obstacle and then how the robot should use the forward, forward-left, and forward-right operators to maintain the desired distance from the wall. The goal of following a wall is considered to be accomplished (and the agent therefore exits the followWall subgoal) when the robot's heading is within 30 degrees of the heading to the target and no obstacles are sensed by any of the infrared or sonar sensors.

C. System Integration

Figure 16 shows the flow of information between the hardware and software components used by the CRS to control the SuperDroid. This figure is similar to the figure describing the system integration for the hexapod in Fig. 8; the differences are caused by the additional sensors used on the SuperDroid and the motor requirements for two different types of robots (wheeled vs. legged). The addition of infrared distance sensors is shown in the hardware layer. Because only two (servo) motor commands are needed to drive the SuperDroid (versus 12 motors that must be controlled for the hexapod), the dedicated servo controller board (the PSC board in Fig. 8) used on the hexapod is not necessary. Instead of using the WalkingRobot class to send motor commands to the PSC board, the BrainStem class is used to send motor commands from the BrainStem board to the MD22 motor driver board. So, for the CRS implementation on the SuperDroid, the BrainStem class communicates with both sensors (infrared, sonar, compass) and motors (via the motor driver board).

The time for a Soar agent to produce output after it receives new sensor information (which can consist of several decision cycles in Soar) varies depending on how much information the Soar agent makes available about its reasoning. With minimal information output from the Soar agent, the Soar agent produces output in the form of a command in 30–50 ms on the most recent laptop used on the SuperDroid robot. The entire cycle of the CRS takes about 300 ms.

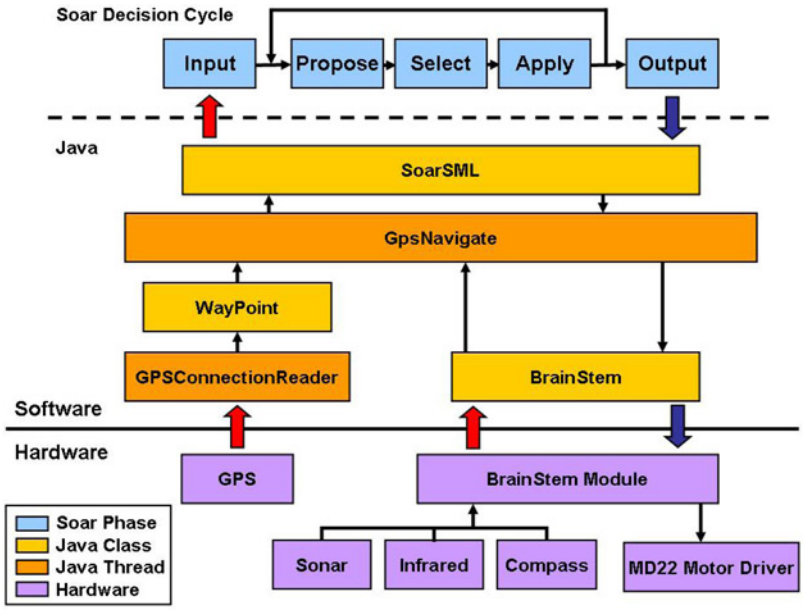

Fig. 16 The data flow between all hardware and software components of the CRS implementation on the SuperDroid is shown.

D. Results

The test shown in Fig. 12 was repeated using the CRS to control the SuperDroid with the changes described above and the results are shown in Fig. 17. The SuperDroid was placed at the location marked at the green circle labeled 'start' for this test. The Soar agent used the goToTarget subgoal to turn to the right and start moving toward the target.

1) At the point labeled '1' in Fig. 17, the robot sensed the wall to its left as an obstacle and used the avoid subgoal to turn to the right until it had moved far enough so that the wall was not considered an obstacle. At this point (no obstacles detected), the goToTarget abstract operator was selected. In the goToTarget subgoal, the forward-left and forward operators are proposed: the forward-left operator because the target is to the left of the robot and the forward operator because it is always proposed in the goToTarget subgoal in case the only other option is to undo a recently performed action. Because the robot just turned right to avoid an obstacle, the forward operator was selected instead of the forward-left operator to avoid undoing the recently applied turn-right operator. The Soar agent selected the forward operator for a total of five decision cycles, choosing to select this operator instead of the forward-left

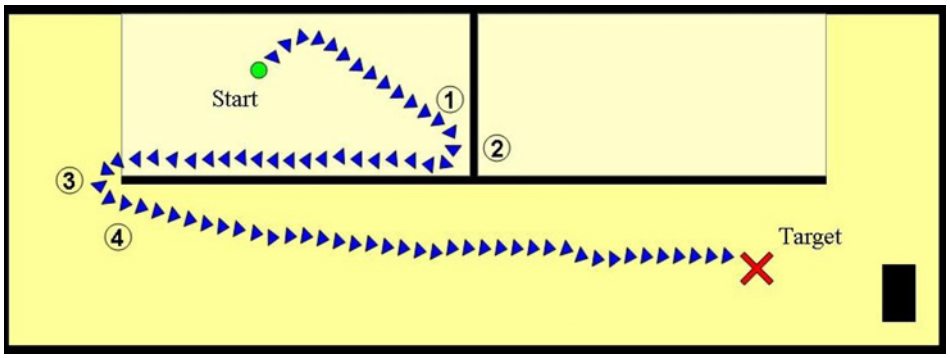

Fig. 17 The results of the test of the SuperDroid, in which the improved Soar agent navigates to the target location in the presence of a cul-de-sac, are shown.

operators that were also proposed because the last-avoid-operator was remembered for five cycles. After five cycles, the last-avoid-operator was forgotten and the forward-left operator was chosen. Within two decision cycles, the robot was again close enough to the wall for the wall to be considered an obstacle, resulting in the use of the avoid subgoal to move away from the wall again using the turn-right operator.

2) The process described in the preceding paragraph repeated itself until point '2' in Fig. 17, when the agent decided that the SuperDroid was repeatedly encountering and having trouble avoiding an obstacle on its left and proposed and selected the followWall abstract operator. In the followWall subgoal, the agent then chose to follow the wall on its left side. The agent was able to turn the corner near point '2' (in Fig. 17) using the avoidWall subgoal after the horizontal wall (in Fig. 17) had been sensed by the robot's sonar sensors.

3) The Soar agent remained in the followWall subgoal while it followed the wall on the left side of the robot and continued to follow the wall around the 180 degree turn in the wall near point '3' in Fig. 17.

4) Once the robot reached point '4' in Fig. 17, the Soar agent exited the followWall subgoal because its heading was within 30 degrees of the heading to the target and no obstacles were sensed. The agent then entered the goToTarget subgoal and used the low-level operators for this subgoal to successfully navigate to the target location.

This section has described the use of the CRS on a wheeled robot. Two main extensions to the CRS were implemented on the SuperDroid to allow the SuperDroid to navigate to a target location in the presence of complex obstacles. First, infrared distance sensors were added to increase the awareness the Soar agent had of its environment. Second, production rules were added to the Soar agent that allowed the agent to use another operator (followWall) to avoid obstacles and to remember and reason more about its recent actions and their results. This reasoning resulted in the Soar agent trying to avoid choosing operators that could undo the recent actions of the SuperDroid and trying to recognize situations in which the SuperDroid was having trouble avoiding an obstacle. These extensions to the CRS resulted in the successful completion of the navigation mission in the presence of a cul-de-sac obstacle.

V. Conclusion

This paper described the development of a robotic system to use the Soar cognitive architecture for the control of unmanned vehicles. Java has been used to integrate Soar with the sensors and motors of two types of unmanned ground vehicles. A Soar agent was developed to test the CRS on a practical mission of GPS navigation with obstacle avoidance. The Soar agent was able to successfully navigate even a cul-de-sac type obstacle.

The Soar architecture has been successfully used by other researchers to control unmanned vehicles (e.g. [34,35]), but this research has mainly been in simulated environments. Control of unmanned vehicles is more challenging in real-world environments because of noise in sensor and motor systems and the challenge of finding meaningful and appropriate symbols from sensor data. The Soar architecture allows the use of automatic subgoals to execute goals hierarchically, can achieve both reactive and deliberative behavior, and is capable of supporting agents with large amounts of knowledge in the form of production rules. The ability to integrate Soar with other software systems whose strengths complement Soar's strengths for controlling unmanned vehicles is also very important.

The CRS system developed here is flexible enough to work on different unmanned vehicle missions. Cognitive architectures are designed to be domain-independent and the general mechanisms of Soar and the CRS would be applicable to more complex unmanned vehicle missions. The use of the CRS to perform the navigation mission discussed in this paper does not preclude the use of specialized path planning approaches. For example, the Soar agent could try using its current production rules to accomplish the navigation mission, but realize if it is having difficulty completing the mission that it could decide to use a path planning algorithm to calculate a path the robot could use to complete its mission.

Although the research described in this paper represents abilities important for a robot performing missions in a real-world environment, it does not yet exploit all the capabilities of Soar or take advantage of all of the systems that can be incorporated into the CRS. The CRS can be extended in several ways, including the addition of subsymbolic processing components (such as well-established algorithms in areas of unmanned vehicle control that do not match Soar's strengths (e.g., specialized path planning algorithms) or other cognitively-inspired subsymbolic systems (e.g., spiking neural networks)), the addition of Soar production rules to increase the sophistication of the robot's behavior, and the addition of sensors to increase the situational awareness of the Soar agent. Another potential area of future work is collaborative missions using multiple robots running Soar, robots using Soar and robots using different control architectures, or robots using Soar and humans (to interact in a way natural to humans).

Acknowledgment

The support of the NSF Graduate Research Fellowship Program is gratefully acknowledged.

References

[1] Bekey, G. A., *Autonomous Robots: From Biological Inspiration to Implementation and Control*, MIT Press, Cambridge, MA, 2005.

[2] Jackel, L. D., Krotkov, E., Perschbacher, M., Pippine, J., and Sullivan, C., "The DARPA LAGR Program: Goals, Challenges, Methodology, and Phase I Results," *Journal of Field Robotics*, Vol. 23, No. 11-12, 2006, pp. 945–973. doi: 10.1002/rob.20161

[3] Gottfredson, L. S., "Mainstream Science on Intelligence: An Editorial With 52 Signatories, History, and Bibliography," *Intelligence*, Vol. 24, No. 1, 1997, pp. 13–23. doi: 10.1016/S0160-2896(97)90011-8

[4] Hanford, S. D., and Long, L. N., "Evaluating Cognitive Architectures for Unmanned Autonomous Vehicles," Evaluating Architectures for Intelligence: Papers from the 2007 Association for the Advancement of Artificial Intelligence Workshop (Technical Report WS-07-04), Vancouver, Canada, July, 2007.

[5] Long, L. N., and Kelley, T. D., "The Requirements and Possibilities of Creating Conscious Systems," to be presented at the 2009 AIAA InfoTech@Aerospace Conference, Seattle, WA, April 6–9, 2009.

[6] Albus, J. S., McCain, H. G., and Lumia, R., "NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM)," NIST/TN-1235, National Institute of Standards and Technology, Gaithersburg, MD, 1989.

[7] Albus, J. S., "The NIST Real-time Control System (RCS): An Approach to Intelligent Systems Research," *Journal of Experimental and Theoretical Artificial Intelligence*, Vol. 9, No. 2–3, 1997, pp. 157–174. doi: 10.1080/095281397147059

[8] Brooks, R., "A Robust Layered Control System for a Mobile Robot," *IEEE Journal of Robotics and Automation*, Vol. 2, No. 1, 1986, pp. 14–23.

[9] Arkin R. C., and Balch T., "AuRA: Principles and Practice in Review," *Journal of Experimental & Theoretical Artificial Intelligence*, Vol. 9, No. 2–3, 1997, pp. 175–189. doi: 10.1080/095281397147068

[10] Bonasso, R. P., Firby, R. J., Gat, E., Kortenkamp, D., Miller, D. P., and Slack, M. G., "Experiences with an Architecture for Intlligent, Reactive Agents," *Journal of Experimental & Theoretical Artificial Intelligence*, Vol. 9, No. 2–3, 1997, pp. 237–256. doi: 10.1080/095281397147103

[11] Chien, S., et al., "Using Autonomy Flight Software to Improve Science Return on Earth Observing One," *Journal of Aerospace Computing, Information, and Computing*, Vol. 2, April 2005, pp. 196–216.

[12] Chien, S., Knight, R., Stechert, A., Sherwood, R., and Rabideau, G., "Using Iterative Repair to Improve Responsiveness of Planning and Scheduling," *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling*, April 2000.

[13] Buckley, B., and Van Gaasbeck, J., "SCL: An Off-The-Shelf System for Spacecraft Control," NASA N95-17243, Nov. 1, 1994.

[14] Firby, R. J., "Adaptive Execution in Complex Dynamic Worlds," Ph.D. Dissertation, Yale University, New Haven, CT, 1989.

[15] Elsaesser, C., and Slack, M. G., "Integrating Deliberative Planning in a Robot Architecture," *Proceedings of the AIAA/NASA Conference on Intelligent Robots in Field, Factory, Service, and Space*, 1994.

[16] Murphy, R., *Introduction to AI Robotics*, MIT Press, Cambridge, MA, 2000.

[17] Kortenkamp, D., Bonasso, R. P., and Murphy, R., *Artificial Intelligence and Mobile Robots*, AAAI Press/MIT Press, Menlo Park, CA, 1998.

[18] Figueroa, F., "Introduction: Achieving Intelligence in Aerospace Systems," *Journal of Aerospace Computing, Information, and Communication*, Vol. 4, No. 5, 2007, pp. 751–752.

[19] Sauter, J. A., Matthews, R., Parunak, H. V. D., Brueckner, S. A., "Effectiveness of Digital Pheromones Controlling Swarming Vehicles in Military Scenarios," *Journal of Aerospace Computing, Information, and Communication*, Vol. 4, No. 5, 2007, pp. 753–769.

[20] Barber, D. B., Griffiths, S. R., McLain, T. W., Beard, R. W., "Autonomous Landing of Miniature Aerial Vehicles," *Journal of Aerospace Computing, Information, and Communication*, Vol. 4, No. 5, 2007, pp. 770–784.

[21] Schulz, H.-W., Buschmann, M., Krüger, L., Winkler, S., Vörsmann, P., "Towards Vision-Based Autonomous Landing for Small UAVs—First Experimental Results of the Vision System," *Journal of Aerospace Computing, Information, and Communication*, Vol. 4, No. 5, 2007, pp. 785–797.

[22] Ahner, D. K., "Real-Time Planning and Control of Army UAVs Under Uncertainty," *Journal of Aerospace Computing, Information, and Communication*, Vol. 4, No. 5, 2007, pp. 798–815.
[23] Miller, J. A., Minear, P. D., Niessner, A. F., DeLullo, A. M., Geiger, B. R., Long, L. N., Horn, J. F., "Intelligent Unmanned Air Vehicle Flight Systems," *Journal of Aerospace Computing, Information, and Communication*, Vol. 4, No. 5, 2007, pp. 816–835.
[24] Wray, R. E., Chong, R. S., "Comparing Cognitive Models and Human Behavior Models: Two Computational Tools for Expressing Human Behavior," *Journal of Aerospace Computing, Information, and Communication*, Vol. 4, No. 5, 2007, pp. 836–852.
[25] Lohn, J., Hornby, G., Linden, D., "Tools for Automated Antenna Design and Optimization," *Journal of Aerospace Computing, Information, and Communication*, Vol. 4, No. 5, 2007, pp. 853–864.
[26] Newell, A., *Unified Theories of Cognition*, Harvard University Press, Cambridge, MA, 1990.
[27] Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., and Qin, Y., "An Integrated Theory of the Mind," *Psychological Review*, Vol. 111, No. 4, 2004, pp. 10361060.
doi: 10.1037/0033-295X.111.4.1036
[28] Laird, J. E., Newell, A., and Rosenbloom, P. S., "Soar: An Architecture for General Intelligence," *Artificial Intelligence*, Vol. 33, No. 3, 1987, pp. 1–64.
[29] Kieras, D. E., and Meyer, D. E., "An Overview of the EPIC Architecture for Cognition and Performance with Application to Human–Computer Interaction," *Human–Computer Interaction*, Vol. 12, No. 4, 1997, pp. 391–438.
[30] Laird, J. E., "Extending the Soar Cognitive Architecture," *Artificial General Intelligence 2008—Proceedings of the First AGI Conference*, by P. Wang, B. Goertzel, and S. Franklin (eds), Vol. 171, *Frontiers in Artificial Intelligence and Applications*, IOS Press, Amsterdam, Netherlands, 2008, pp. 224–235.
[31] Taatgen, N. A., and Anderson, J. R., "Constraints in Cognitive Architectures," *The Cambridge Handbook of Computational Psychology*, edited by Ron Sun, Cambridge Univ. Press, New York, NY, 2008, pp. 170–185.
[32] Chong, R. S., "Inheriting Constraint in Hybrid Cognitive Architectures: Applying the EASE Architecture to Performance and Learning in a Simplified Air-Traffic Control Task," AFRL-HE-WP-TR-2005-0120, Wright-Patterson AFB, OH, 2004.
[33] Laird, J. E., and Rosenbloom, P. S., "Integrating Execution, Planning, and Learning in Soar for External Environments," *Proceedings of the Eighth Conference on Artificial Intelligence*, AAAI Press, Menlo Park, CA, 1990, pp. 1022–1029.
[34] Jones, R. M., Laird, J. E., Nielsen, R. E., Coulter, K. J., Kenny, R., and Koss, F. V., "Automated Intelligent Pilots for Combat Flight Simulation," *AI Magazine*, Spring 1999, pp. 27–41.
[35] Smith, P. R., and Willcox, S. W., "Systems Research for Practical Autonomy in Unmanned Air Vehicles," *Proceedings of the AIAA Infotech@Aerospace Conference*, AIAA, Washington, DC, AIAA paper 2005-7082, 2005.
[36] Trafton, J. G., et al., "Communicating and Collaborating with Robotic Agents," *Cognition and Multi-Agent Interaction: From Cognitive Modeling to Social Simulation*, edited by R. Sun, Cambridge Univ. Press, New York, NY, 2006, pp. 252–278.
[37] Cassimatis, N. L, "A Cognitive Architecture for Integrating Multiple Representation and Inference Schemes," Ph.D. Dissertation, Massachusetts Institute of Technology, Cambridge, MA, 2002.
[38] Kennedy, W. G., et al., "Spatial Representation and Reasoning for Human–Robot Collaboration," *Proceedings of the Twenty-Second Conference on Artificial Intelligence*, AAAI Press, Menlo Park, CA, 2007, pp. 1554–1559.
[39] Benjamin, P., Lyons, D., and Lonsdale, D., "Designing a Robot Cognitive Architecture with Concurrency and Active Perception," *Proceedings of the AAAI Fall Symposium on the Intersection of Cognitive Science and Robotics*, October, 2004.
[40] Benjamin, P., Lonsdale, D., and Lyons, D., "Embodying a Cognitive Model in a Mobile Robot," *Proceedings of the SPIE Conference on Intelligent Robots and Computer Vision*, October, 2006.
[41] Avery, E., Kelley, T., and Davani, D., "Using Cognitive Architectures to Improve Robot Control: Integrating Production Systems, Semantic Networks, and Sub-Symbolic Processing," Behavior Representation in Modeling and Simulation, May 15–18, 2006.
[42] Kelley, T. D., "Developing a Psychologically Inspired Cognitive Architecture for Robotic Control: The Symbolic and Subsymbolic Robotic Intelligence Control System (SS-RICS)," *International Journal of Advanced Robotic Systems*, Vol. 3, No. 3, 2006, pp. 219–222.
[43] Kelley, T. D., Avery, E., and Long, L. N., "A Hybrid Symbolic and Sub-Symbolic Intelligent System for Mobile Robots," to be presented at the 2009 AIAA InfoTech@Aerospace Conference, Seattle, WA, April 6–9, 2009.
[44] Kelley, T. D., "Using a Cognitive Architecture to Solve Simultaneous Localization and Mapping (SLAM) Problems," ARL-MR-0639, Aberdeen Proving Ground, MD, May 2006.
[45] Long, L. N., Hanford, S. D., Janrathitikarn, O., Sinsley, G. L., and Miller, J. A., "A Review of Intelligent Systems Software for Autonomous Vehicles," *Proceedings of the IEEE Symposium Series in Computational Intelligence*, April 1–5, 2007.
[46] Laird, J. E., "The Soar 8 Tutorial," University of Michigan, Ann Arbor, MI, May 14, 2006.

[47] "SML Quick Start Guide," 2005, ThreePenny Software LLC.
[48] Forgy, C., "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem," *Artificial Intelligence*, Vol. 19, No. 1, 1982, pp. 17–37.
doi: 10.1016/0004-3702(82)90020-0
[49] Todd, D. J., *Walking Machines: An Introduction to Legged Robots*, Chapman and Hall, New York, NY, 1985.
[50] Randall, M. J., *Adaptive Neural Control of Walking Robots*, Professional Engineering Publishing Ltd, London, 2001.
[51] Gonzalez de Santos, P., Garcia, E., and Estremera, J., *Quadrupedal Locomotion: An Introduction to the Control of Four-Legged Robots*, Springer-Verlag, Berlin, 2006.
[52] Rosheim, M. E., *Robot Evolution: The Development of Anthrobotics*, Wiley, New York, NY, 1994.
[53] Janrathitikarn, O., "The Use of a Cognitive Architecture to Control a Six-Legged Robot," M.S. Thesis, Department of Aerospace Engineering, The Pennsylvania State University, University Park, PA, 2007.
[54] Janrathitikarn, O., and Long, L. N., "Gait Control of a Six-Legged Robot on Unlevel Terrain using a Cognitive Architecture," *Proceedings of the IEEE Aerospace Conference*, Mar. 1–8, 2008.

Christopher Rouff
Associate Editor