

Control of a Six-Legged Mobile Robot Using the Soar Cognitive Architecture

Scott D. Hanford^{*}, Oranuj Janrathitikarn[†], and Lyle N. Long[‡]
The Pennsylvania State University, University Park, PA, 16802

This paper describes the development of the Cognitive Robotic System (CRS) for studying intelligent and autonomous unmanned vehicles. This system uses a multi-threaded software system written using Java to integrate the Soar cognitive architecture with a six-legged mobile robot and its sensors and motors. The results of experiments in which the Cognitive Robotic System was used to control the hexapod on missions of GPS navigation with obstacle avoidance are described. The CRS was designed so that its capabilities can be expanded in the future through the inclusion of 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) are essentially remotely piloted vehicles that possess very little intelligence or autonomy. However, 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 DARPA Grand Challenge and the 2007 DARPA Urban Grand Challenge. 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.”² Achieving intelligence in unmanned vehicles may be an even more challenging problem than achieving autonomy in these vehicles. 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.³

In an effort to improve the intelligence and autonomy of future mobile robots, the use of intelligent systems software has become more popular.⁴⁻⁶ Software for intelligent systems can be classified as either symbolic or subsymbolic. One way to explain the distinction between symbolic and subsymbolic systems is the relationship between the level of semantic interpretation (the level at which meaningful concepts are described) and the formal representation for the system’s processing.⁷ For symbolic systems, the level of semantic interpretation and the formal representation for processing are both at the level of symbols: the symbols are the meaningful concepts and the processing involves manipulation of symbols. For subsymbolic systems, the formal representation for processing occurs below the level at which the concepts of interest are described. For example, the level of semantic interpretation for a digital image could be objects in the image (e.g., doors or trashcans). A symbolic processing system would manipulate symbols that represent these objects in the image while a subsymbolic processing system would manipulate the pixels that make up these objects. Common subsymbolic systems include neural networks and examples of common symbolic systems are logical reasoning systems and cognitive architectures such as Soar. Subsymbolic systems have proven to be useful for pattern recognition and signal processing and have exhibited significant abilities to learn, while symbolic systems have proved useful for reasoning, planning, and other high-level behaviors important for unmanned vehicles. For real-world environments, learning at the subsymbolic level

^{*} NSF Graduate Research Fellow, Aerospace Engineering, 229 Hammond Building, AIAA Student Member.

[†] Graduate Student Researcher, Aerospace Engineering, 229 Hammond Building, AIAA Student Member.

[‡] Distinguished Professor, Aerospace Engineering, Mathematics, and Bioengineering, 229 Hammond Building, AIAA Fellow.

(i.e., gradual tuning of parameters such as connection weights in neural networks with increased experience) has been more common than learning at the symbolic level (i.e. learning new behaviors or symbolic rules).

A. Subsymbolic systems

Although neural networks have been used to control robots,⁸ they may be most useful for perception. An interesting use of neural networks in mobile robots has been to locate doors by using the neural network to detect the presence of doors in images.⁹ Monasterio et al. used neural networks and ultrasound sensors for both perception (verifying locations of open doors in a mobile robot's environment) and control (driving the robot through the open doors).¹⁰

Spiking neural networks are neural models capable of including time effects (not just mean firing rate as in traditional artificial neural networks). Spiking neural networks have been used to drive a mobile robot and move its arm by recognizing simple instructions such as "turn left" or "lift" spoken by its operator.¹¹ These simple control instructions were then carried out by the robot. These networks have also been used for character recognition.¹² Long and Gupta described how spiking neural networks and Hebbian learning can be used for vision systems.¹³ They also showed how artificial neural networks can be made scalable on massively parallel computers.¹⁴

Other subsymbolic systems that have been useful for unmanned vehicles include computer vision systems with image processing algorithms and fuzzy logic systems. These two systems have been used to perform tasks such as finding doors.¹⁵

B. Symbolic systems

Cognitive architectures are examples of symbolic systems that have been used in intelligent systems. These systems were originally developed to model human performance and have displayed impressive capabilities for high-level behaviors in various domains. These capabilities have resulted in interest in using cognitive architectures for robotics. However, most of these architectures were not necessarily designed for use in the real-world environments that unmanned vehicles encounter so researchers interested in using cognitive architectures for unmanned vehicles have often designed systems around the cognitive architectures to facilitate their use in these vehicles.

Soar is a well-known cognitive architecture that has previously been used to control mobile robots and *simulated* unmanned vehicles.¹⁶ Soar has been used with a mobile robot equipped with sonar sensors and a robotic gripper to locate and pick up cups.¹⁷ Jones et al. used the Soar architecture to autonomously fly U.S. military fixed-wing aircraft during *simulated* missions.¹⁸ Another project used Soar agents to control *simulated* air and ground vehicles.¹⁹

The Intelligent Systems Section at the Navy Center for Applied Research in Artificial Intelligence has used cognitive architectures to build robots that can think and act like humans in order to facilitate better and easier collaboration between humans and robots.²⁰ ACT-R, ACT-R/S (an extension of ACT-R that includes a theory of spatial reasoning), and Polyscheme 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.²⁰ Additionally, Kennedy et al. 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.²¹ 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 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.^{22,23} 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 such as a contact or force sensor to determine when the foot has touched the ground). Active perception, such as vision, 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 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.^{24,25} Subsymbolic processing is used to process all sensor inputs. An example of this processing is determining if points measured from a laser rangefinder can be grouped as lines that may indicate a

landmark.²⁶ Symbolic processing exists in the form of a production system (similar to ACT-R and Soar) that has access to high-level symbolic memory, which includes knowledge stored in the semantic network (common sense information), and results from the subsymbolic processing.

II. Soar for Unmanned Vehicles

Features of intelligent systems software packages that are useful for intelligent unmanned vehicles include: simple communication between the software and environment through many sensors and motors, a mix of reactive and deliberative behaviors, definition of a learning mechanism, and the ability for collaboration with other vehicles or software systems.²⁷ Although there are several packages that have been used for research in intelligent unmanned vehicles, Soar has the potential to meet these criteria and the research described in this paper explored the use of Soar for mobile robots.

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 about its environment is stored in working memory, which can be thought of as a form of short-term memory. In a Soar model, 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.

The execution of a Soar agent or model is governed by a decision cycle (Fig. 1) in which the agent chooses an appropriate action to take based on the information it knows about its current situation and its long-term knowledge. This cycle that can be broken into five parts:²⁸

- 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.
- 3) In this 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. This mechanism supports both the hierarchical execution of complex goals (as abstract operators or problem spaces) and a mix between reactive selection of an operator (when sufficient knowledge is available to choose one) and on-demand planning that can occur when a lack of knowledge results in an impasse. 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.

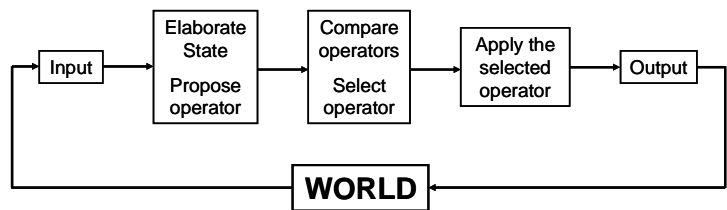


Figure 1. The five stages of Soar's decision cycle for a Soar model that interacts with the external world.

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

- The creation of impasses and automatic subgoals support hierarchical decomposition of complex goals or abstract operators. This mechanism allows complex operators to accomplish goals by using several simpler operators.
- Problem spaces are convenient for Soar models that have a large amount of long-term knowledge and are therefore capable of working on several types of problems or goals. For these models, even though a large number of operators are likely to be eligible for proposal based on the model's current state, only a small number of the operators eligible for proposal are likely to be useful for accomplishing the model's current

goal. Organizing problem solving into problem spaces allows only operators relevant to the current goal to be considered. This concept will be explained in detail later.

- The Soar Markup Language (SML) has been developed to allow simplified interaction of a Soar agent with external environments through sensors and motors.²⁹ Also, since SML can be used in popular programming languages including Java and C++, integration with other software systems is possible.
- The Soar architecture is also capable of handling the large models 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.¹⁸ Soar uses the Rete algorithm, which is efficient for a large number of working memory elements, to determine which of the agent's productions are matched based on the current contents of working memory.³⁰
- 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.¹⁸ Another project used Soar agents to control unmanned air and ground vehicles to cooperatively perform tasks such as communications relay, waypoint following, and conducting surveillance during simulated missions.¹⁹
- 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.
- 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 since Soar does not have a simple mechanism for unlearning incorrect knowledge.

The above characteristics suggest that Soar is suitable for high-level cognitive tasks such as reasoning and planning for unmanned vehicles. However, the uncertainty in perception associated with real-world environments may make using Soar's learning mechanism less useful than in other environments where uncertainty is less prevalent. Combining Soar with other software systems could improve unmanned vehicle performance: subsymbolic systems that have proven to be useful for pattern recognition and signal processing could reduce the uncertainty associated with perception, increasing the usefulness of Soar's learning mechanism as well as providing their own learning capabilities.

The purpose of this paper is to describe the development of an architecture, the Cognitive Robotic System, to explore the use of the Soar cognitive architecture for intelligent and autonomous unmanned vehicles and to demonstrate the application of the developed architecture to a practical unmanned ground vehicle mission, navigating to a target GPS location while avoiding obstacles.

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 the main 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.³¹

Legged robots can be categorized based on their number of legs.³² Two-, four-, and six-legged systems are more typical than one-, three-, five-, and eight+ legged systems.³³ Hexapods (six-legged systems) have taken biological inspiration from insects. Having more legs than two-legged (bipeds) and four-legged (quadrupeds) robots requires hexapod robots to have more hardware and makes the hexapod less agile than bipeds and quadrupeds. The greatest advantage of hexapods is that they have more desirable stability properties than bipeds and quadrupeds.³³ 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.³⁴ 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. In order 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. 2, 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 board (www.via.com.tw/en/), 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, including USB, serial, and IIC ports. Memory (512 MB), a hard-drive, and a wireless card were added to the board.

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 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 IIC 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 uses two sonar sensors, an electronic compass, and a GPS receiver. Figure 3 shows the physical connections of the hardware currently used on the hexapod.³⁵ Janrathitikarn and Long have also investigated the use of touch sensors on the hexapod's feet.³⁶ 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 (<http://www.robot-electronics.co.uk>) provides measurements of the hexapod's heading relative to the earth's magnetic field. Since 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 IIC capability of the BrainStem module was used to communicate with the compass and receive the heading of the hexapod with a range between 0 and 359.9 degrees.

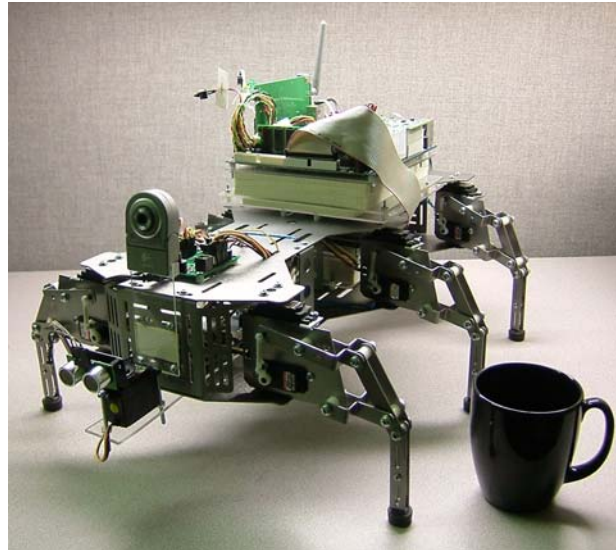


Figure 2. A picture of the hexapod next to a coffee mug.

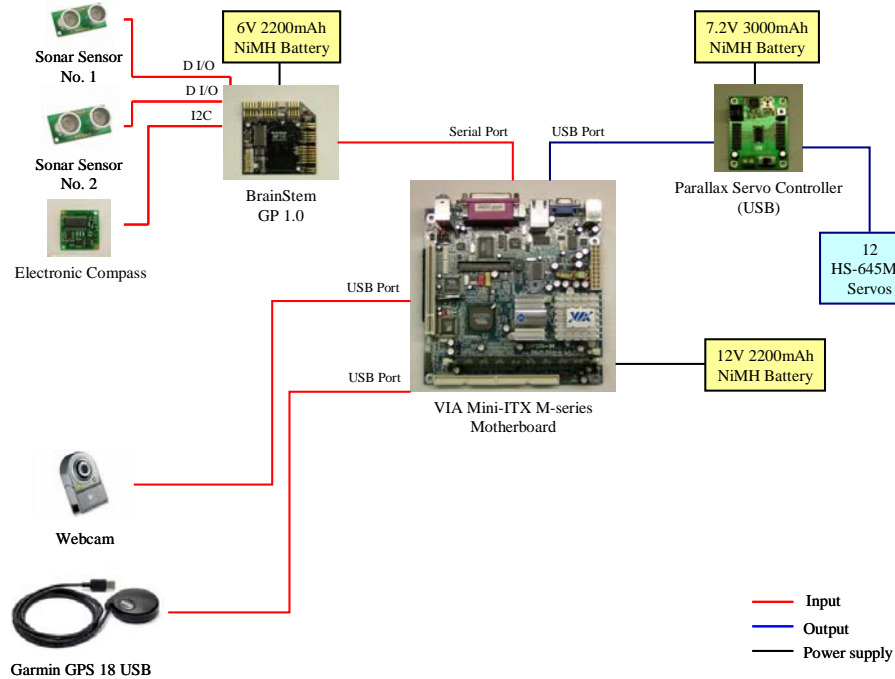


Figure 3. A schematic of the hardware used onboard the hexapod and their connections. Inputs to the BrainStem module and Via Motherboard are shown as red lines. Output from the Via Motherboard and Parallax Servo Controller are shown by the blue lines. The black lines represent power supplied to the BrainStem module, the Via Motherboard, and the Parallax Servo Controller.³⁵

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.”

IV. Software for Hexapod

Several software packages have been installed on the hexapod’s VIA board computer to develop an architecture for research in intelligent and autonomous unmanned vehicles:³⁵

- Java 2 Standard Edition (J2SE) 5.0 Java Communications API (javax.comm)
- Soar (version 8.6.3, including SML)
- BrainStem acronym Java package
- GPS software

A. 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.

B. 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 multithreading capability that allows concurrent computing. 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.

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).

- Several Java classes were developed 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 GPSThread 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 relevant data, and sends it to the GPSThreadReader thread. After receiving the GPS data, the GPSThreadReader 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 requested anytime from the main thread.
- 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, reading and writing to the scratchpad memory on the BrainStem module, etc. 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.
- 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.
- 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 productions, 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.

C. Soar Problem Spaces and Productions

The Soar agent developed to go to a target GPS location while avoiding obstacles implements two complex 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 complex operators are selected, an impasse occurs that results in a problem space, or substate, that is named avoid or goToTarget, respectively. The goal of this problem space is to accomplish the goal of avoiding an obstacle or moving the hexapod to the target GPS location. In this new problem space, low-level operators that are relevant to the goal of avoid or goToTarget can be proposed, selected, and applied to achieve these two goals (Fig. 4).

In the avoid problem space, the turn-left and turn-right low-level operators are eligible to be proposed. 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.

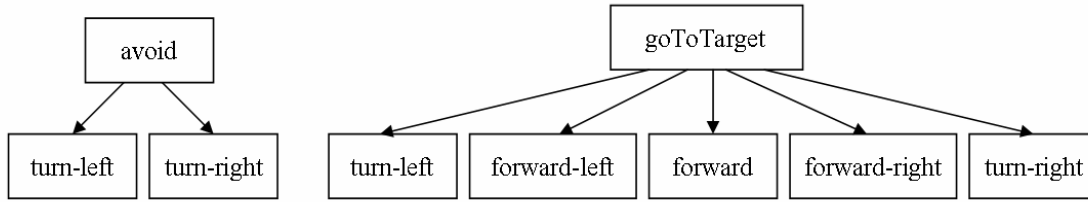


Figure 4. The hierarchical structure of the avoid and goToTarget problem spaces and their low-level operators is shown.

In the goToTarget problem space, 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 target location and the current heading measured by 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 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 5 shows the ranges of values of deltaAngle for which each of the five operators will be proposed.

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 problem space changes. For example, if the hexapod is moving forward in the goToTarget problem space and detects an obstacle ahead and to the right, the avoid operator will be selected. In the avoid problem space, 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 memory 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 in order to go to the target until ten operators have been chosen since the turn left operator was chosen in the avoid problem space. 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.

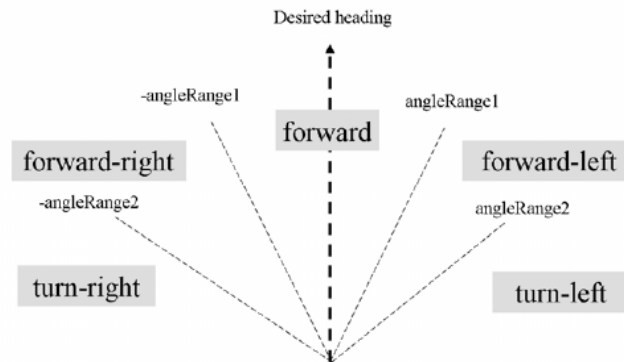


Figure 5. This figure shows the decision criteria used to propose the five operators in the goToTarget problem space. The desired heading is shown to be straight up and has a value of zero in this figure. If the hexapod's heading, and therefore the value of deltaAngle, is greater than $-\text{angleRange1}$ 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 $-\text{angleRange2}$, the turn-right operator will be proposed. If the value of deltaAngle is greater than $-\text{angleRange2}$ and less than $-\text{angleRange1}$, the forward-right operator will be proposed.

V. 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 6 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 Cognitive Robotic System 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 in the main thread.

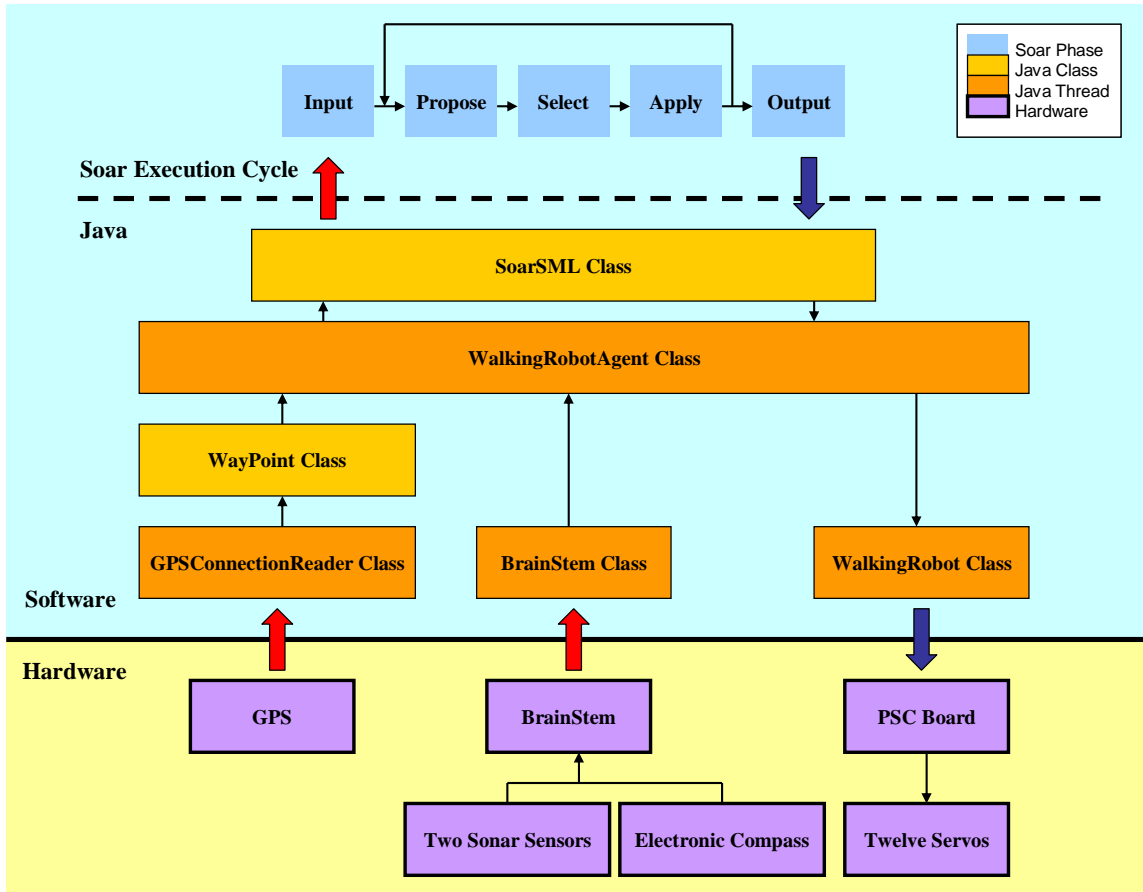


Figure 6. The data flow between all hardware and software components of the CRS are shown.

Results 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 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. 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.

VI. Robotic control architecture

The Cognitive Robotic System has been developed around Soar (Fig. 7) and can be extended to produce more sophisticated robot behaviors and include other software systems and algorithms. Starting in the bottom left of Fig. 7, information from the world can be processed using subsymbolic processing techniques such as neural networks, fuzzy logic, or image processing. In the current version of the CRS, subsymbolic processing of sensor input has not been implemented and raw data is given to the Soar agent. In the future, processing data from sensors at this stage

could be a valuable tool to deal with uncertainty. After the Soar agent receives data in its input phase, Soar operators corresponding to the robot behaviors in the rectangular box are proposed in parallel such that all operators that are appropriate for the current situation will be proposed. The current set of Soar operators can produce two behaviors: avoid obstacles and go to GPS location. Additional robot behaviors can be included by adding more knowledge, in the form of operators, to the Soar agent.

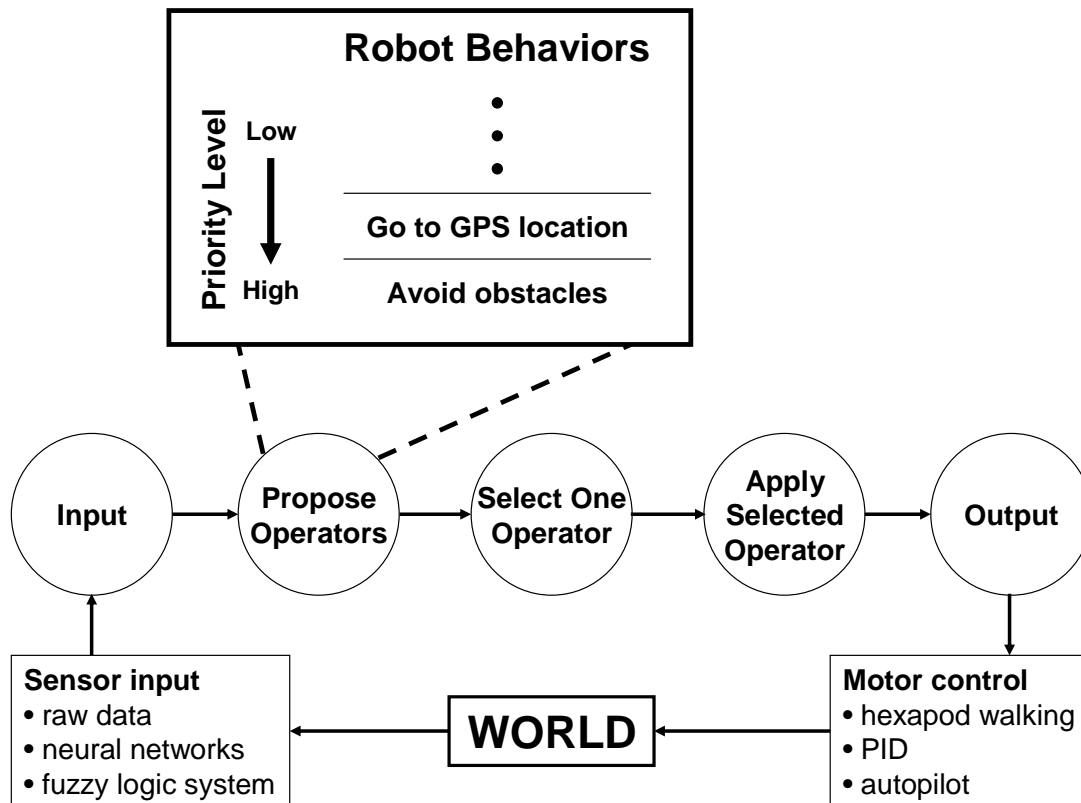


Figure 7. A schematic of the Cognitive Robotic System. The execution cycle of the CRS includes a phase for sensor input from the world, the five phases of the Soar decision cycle (shown in circles), and a motor control phase. The knowledge needed to perform the robot behaviors listed in the rectangle above the “Propose Operators” circle is programmed as Soar productions. The priority level of these behaviors is used by Soar to select a single operator if multiple operators have been proposed.

After all the appropriate operators have been proposed, Soar compares the priority level of these operators and selects one operator to apply. This cycle of propose, select, and apply continues until output is produced. Currently, the output from Soar is in the form of commands to the hexapod, but it could also include inputs to a motor control algorithm, inputs to an autopilot on an unmanned air vehicle, or a command to perform a task such as taking a picture of a specific landmark.

VII. Results

Experiments were performed with three different types of obstacles to test whether the Cognitive Robotic System 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 schematic of the parking lot is depicted in Figs. 8, 9, and 10. 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’

hexapod was able to turn left away from the obstacle. After the obstacle was no longer detected, the hexapod resumed walking to the target in 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. 10. 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. 10). 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. 10 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. 10, until the test was terminated unsuccessfully.

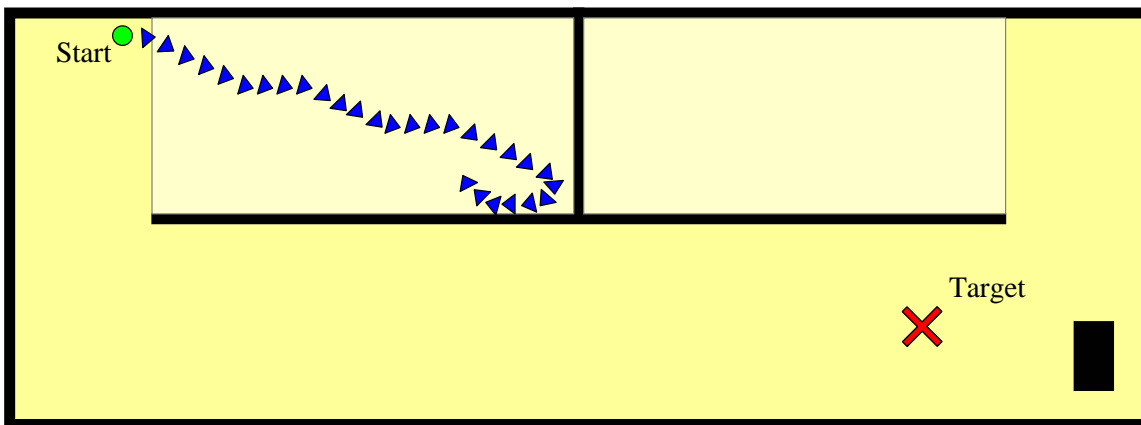


Figure 10. The results of the third test are shown. The hexapod started in the upper left corner of the figure at the green circle and was not able to successfully reach the target location because it became stuck in the cul-de-sac in the upper left corner of the figure.

These tests have shown that our control architecture is capable of guiding the hexapod to a GPS target location with relatively simple obstacles. However, the architecture 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 architecture in the form of additional sensors or more sophisticated behaviors (e.g., Soar productions to perform map building or wall following) would increase the possibility of successfully completing tests such as the one depicted in Fig. 10.

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 can not be used, to locate landmarks such as doors that are useful for navigation in indoor environments.

VIII. Conclusion

This paper described the development of the Cognitive Robotic System for studying intelligent and autonomous unmanned vehicles. Java has been used to integrate a popular cognitive architecture, Soar, with the sensors and motors of a hexapod robot. This system has been tested on a practical mission of GPS navigation with obstacle avoidance. The CRS has also been implemented on another unmanned vehicle and tested on the same mission with similar results.

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, the addition of Soar productions to increase the sophistication of the robot's behavior, and the addition of sensors to increase the situational awareness of the hexapod. Another potential area of future work is collaborative missions using multiple robots running Soar or even other robots using different control architectures.

Acknowledgments

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

References

- ¹Bekey, G.A. *Autonomous Robots: From Biological Inspiration to Implementation and Control*, The MIT Press, Cambridge, MA, 2005.
- ²Gottfredson, L.S., "Mainstream Science on Intelligence: An Editorial With 52 Signatories, History, and Bibliography," *Intelligence*, Vol. 24, No. 1, 1997, pp. 13-23.
- ³Jackel, 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.
- ⁴*Journal of Aerospace Computing, Information, and Communication*, December 2004, special issue on Intelligence in Aerospace Systems.
- ⁵*Journal of Aerospace Computing, Information, and Communication*, May 2007, special issue on Achieving Intelligence in Aerospace Systems.
- ⁶Sinsley, G., Long, Lyle N., Niessner, A.F., and Horn, J.F., "Intelligent Systems Software for Unmanned Air Vehicles," *Proceedings of the 46th AIAA Aerospace Sciences Meeting*, AIAA Paper No. 2008-0871, Jan. 7-10, 2008 (to be presented).
- ⁷Smolensky, P., "Connectionist Modeling: Neural Computation/Mental Connections." *Neural Connections, Mental Computation*, edited by L. Nadel, L.A. Cooper, P. Culicover, and R.M. Harnish., Bradford/MIT Press, Cambridge, MA, 1989.
- ⁸Zalzala, A.M.S. and Morris, A.S. (ed.), *Neural Networks for Robotic Control: Theory and Applications*, Ellis Horwood, New York, 1996.
- ⁹Cicirelli, G., D'Orazio, T.; Distanto, A., "Target Recognition by Components for Mobile Robot Navigation," *Journal of experimental & theoretical artificial intelligence*, Vol. 15, No.3, Jul-Sept, 2003, pp. 281-297.
- ¹⁰Monasterio, I.; Lazkano, E.; Rañó, I.; Sierra, B., "Learning to Traverse Doors using Visual Information," *Mathematics and Computers in Simulation*, 2002, pp. 347-356.
- ¹¹Panchev, C. and Wernter, S., "Temporal Sequence Detection with Spiking Neurons: Towards Recognizing Robot Language Instruction," *Connection Science*, Vol. 18, 2006, pp. 1-22.
- ¹²Gupta, A. and Long, L.N., "Character Recognition using Spiking Neural Network," *Proceedings of the IEEE Neural Network Conference*, Aug. 12-17, 2007.
- ¹³Long, L.N. and Gupta, A., "Spiking Neural Networks with Hebbian Learning for Image Processing," *Proceedings of the 46th AIAA Aerospace Sciences Meeting*, AIAA Paper No. 2008-0885, Jan. 7-10, 2008 (to be presented).
- ¹⁴Long, L.N. and Gupta, A., "Scalable Massively Parallel Artificial Neural Networks," *Journal of Aerospace Computing, Information, and Communication (JACIC)*, (to be published).
- ¹⁵Muñoz-Salinas, R.; Aguirre, E.; Garcia-Silvente, M., "Detection of Doors using a Genetic Visual Fuzzy System for Mobile Robots," *Autonomous Robots*, Vol. 21, 2006, pp. 123-141.
- ¹⁶Laird, J.E., Newell, A., and Rosenbloom, P.S., "Soar: An Architecture for General Intelligence." *Artificial Intelligence*, Vol. 33, No. 3, 1987, pp. 1-64.
- ¹⁷Laird, 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, 1990, pp. 1022-1029.
- ¹⁸Jones, 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.
- ¹⁹Smith, P.R. and Willcox, S.W., "Systems Research for Practical Autonomy in Unmanned Air Vehicles," *Proceedings of Infotech@Aerospace*, 2005.
- ²⁰Trafton, J. G., Schultz, A. C., Cassimatis, N. L., Hiatt, L. M., Perzanowski, D., P., Brock D. P., Bugajska M., & Adams W., "Communicating and Collaborating with Robotic Agents," *Cognition and Multi-Agent Interaction: From Cognitive Modeling to Social Simulation*, edited by R. Sun, 2006, pp. 252-278.
- ²¹Kennedy, W.G., Bugajska, M., Marge, M., Adams, W., Fransen, B.R., Perzanowski, D., Schultz, A.C., & Trafton, J.G., "Spatial Representation and Reasoning for Human-Robot Collaboration," *Proceedings of the Twenty-Second Conference on Artificial Intelligence*, AAAI Press, 2007, pp. 1554-1559.
- ²²Benjamin, 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.
- ²³Benjamin, 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.
- ²⁴Avery, 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.

- ²⁵Kelley, 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.
- ²⁶Kelley, T.D., "Using a Cognitive Architecture to Solve Simultaneous Localization and Mapping (SLAM) Problems." ARL-MR-0639, Aberdeen Proving Ground, MD, May 2006.
- ²⁷Long, 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.
- ²⁸Laird, J.E., "The Soar 8 Tutorial," University of Michigan, Ann Arbor, MI, May 14, 2006.
- ²⁹"SML Quick Start Guide," 2005, ThreePenny Software LLC.
- ³⁰Forgy, C., "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem," *Artificial Intelligence*, Vol. 19, No. 1, 1982, pp. 17-37.
- ³¹Todd, D.J., *Walking Machines: An Introduction to Legged Robots*, Chapman and Hall, New York, 1985.
- ³²Randall, M.J., *Adaptive Neural Control of Walking Robots*, Professional Engineering Publishing Ltd., London, 2001.
- ³³Gonzalez de Santos, P., Garcia, E., and Estremera, J., *Quadrupedal Locomotion: An Introduction to the Control of Four-Legged Robots*, Springer: Germany, 2006.
- ³⁴Rosheim, M. E., *Robot Evolution: The Development of Anthrobotics*, John Wiley & Sons, Inc., New York, 1994.
- ³⁵Janrathitikarn, 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.
- ³⁶Janrathitikarn, O. and Long, Lyle N., "Gait Control of a Six-Legged Robot on Uneven Terrain using a Cognitive Architecture," *Proceedings of the IEEE Aerospace Conference*, Mar. 1-8, 2008 (to be presented).