IEEE Aerospace Conference, Big Sky, Montana, Mar. 1-8, 2008.

Gait Control of a Six-Legged Robot on Unlevel Terrain Using a Cognitive Architecture

Oranuj Janrathitikarn and Lyle N. Long
Department of Aerospace Engineering
The Pennsylvania State University
University Park, PA 16802
814-865-1172
ozj100@psu.edu, lnl@psu.edu

Abstract—This paper discusses the use of the Soar cognitive architecture to control gait selection of a six-legged robot using force sensors attached to its feet. The hardware platform also incorporated sonar sensors, a GPS receiver, and a webcam. The Soar cognitive architecture was used to control the robot, and the Java programming language was used as middleware between Soar and the hardware components. The force sensors were attached and the force profile was examined. Soar productions for controlling the gait were developed. Experiments were performed on terrain which has random holes and obstacles. Two cases were conducted: walking using Soar to select the gait, and using Soar with the chunking turned on for learning.

TABLE OF CONTENTS

1. INTRODUCTION

Mobile robots have become more and more useful. They play dominant roles in remote and dangerous operations. For instance, two Mars Exploration Rovers, Spirit and Opportunity, have been performing missions on Mars since 2004 [1]. Although most mobile robots are wheeled robots due to their design and construction advantages, legged robots sometimes have benefits over the wheel robots especially on natural terrain. Leg mechanisms have evolved through natural selection so that animals can travel in any kind of terrain and access essentially any place on the earths surface [2]. With a well-designed leg mechanism, legged robots could traverse unlevel and rough terrain more effectively than wheeled robots. Legged robots can jump or step over holes and small obstacles. They can turn around within a small space too. In addition, legged robots have fewer tendencies to damage the ground than wheeled robots [3]. On the other hand, legged robots require complex control algorithms particularly in gait selection and control. Each leg must continuously load and unload the weight within a short time [2]. Legged robots may also have some problems, for example, they can slip on wet surfaces, or one of their legs can get stuck in a hole [3]. In order to develop legged robots to operate in a dynamic environment, more research needs to be done. But the use of intelligent software such as cognitive architectures can help.

The most important key to control legged robots is the gait selection and control. If legged robots sense adequate information from the environment, they should be able to appropriately adjust the leg locomotion to correspond to the external world. Most legged robots use pre-defined leg sequences for traditional control of foot placement planning. In an outdoor environment, legged robots should respond to the unexpected terrain. In addition, they should have the ability to learn from its own experience to improve its walking ability over time. A number of legged-robot locomotion research studies have been performed by either running simulations or experimenting with various robot platforms. The simulation of quadruped search-based foot placement demonstrated a successful traversing of surfaces with the near-leg-length height obstacles [4]. The combination of a local height map and a foot placement search algorithm designed for MoRETA, the quadruped rover, was simulated [5]. The LittleDog quadruped from Boston Dynamics was experimented with on rough terrain using deliberative and reactive modules [6]. Another experiment on the LittleDog on rough terrain was done by using an algorithm to stabilize the center-of-gravity trajectory [7]. The controller used the heading and drift errors measured from a magnetic compass and a camera was implemented and tested on a hexapod [8]. The Q-multiactor learning algorithm for posture control of a hexapod was simulated [9]. However, most of the methodologies in these studies are platform specific and none of them have implemented a cognitive architecture into their hardware and software systems.

Existing intelligent software systems implement different paradigms [10]. Cognitive architectures are an effort to replicate human cognitive processes, problem solving and learning capabilities, and generalize them into a single software package. The State, Operator And Result (Soar) architecture is one of the most well-known cognitive

1

[1] 1-4244-1488-1/08/$25.00©2008 IEEE.
[2] IEEEAC paper#1115, Version 3, Updated November 2, 2007.

architecture software systems and is the best candidate for the Unified Theories of Cognition (UTC) proposed by Allen Newell [11]. It provides many potential features for robotic platforms: acquiring input, reasoning, generating output, and learning. TACAIR-SOAR successfully demonstrated autonomous and intelligent behavior in a military flight simulation [12]. In addition, Soar was implemented on a Puma robot arm and a small robot [13]. This paper reports on the implementation of the Soar architecture to control leg locomotion of a hexapod platform. The ultimate goal is to create an intelligent autonomous system to collaborate with other UGVs.

The six-legged robot platform was assembled and the sensors were installed. Two sonar sensors were mounted in front of the hexapod for obstacle avoidance. A GPS receiver was attached for determining location. An electric compass was used to measure the heading. Force sensors were attached to the feet to check whether the feet are contacting the ground. The sensor data was collected using Java software (developed by the authors), which was then linked to the Soar cognitive architecture. Soar operates the high-level control while Java interfaces with all hardware and connects high-level and low-level communication together. The implementation of the Soar architecture on the hexapod was tested with obstacle avoidance and moving to specific GPS target missions [14]. This paper focuses on using Soar to control leg locomotion on the unlevel terrain by using the force sensors.

This paper also discusses the main features of the Soar architecture. The hardware platform used in this research is described too. Then the details of the hardware and software system integration are provided. Finally, the results of the experiments are described and discussed.

2. THE SOAR ARCHITECTURE

The Soar architecture [15, 16] has been available since 1983 and it is available to download from http://sitemaker.umich.edu/soar/home. Three functional constraints of Soar are to present dynamics and goal driven behavior, to learn from experience, and to demonstrate real-time cognition. Soar is a rule-based system. Its rules, called productions, are the lowest-level memory which contain problem-solving knowledge and is accessible by users. Soar productions create the production memory which is analogous to the long-term memory of humans. The information on the current situation, which is called a state, is stored in short-term memory or working memory. Based on the problem space hypothesis, Soar operates by using its knowledge to select and apply operators to a state. The operators transform the current state to the new one. The transition of states continues until the goal has reached. Soar is very fast and efficient due to the use of the Rete algorithm [16].

Soar operates by executing the continuous cycle comprised of five phases [16]: input phase, operator proposal phase, operator selection phase, application phase, and output phase. In the input phase, Soar receives sensor information from the external world. In the operator proposal phase, the state information is updated by state elaboration rules and any applicable operator is proposed. In the decision phase, Soar selects one of the proposed operators. If Soar cannot select any new operator, it will reach an impasse and create a substate or subgoal to evaluate these operators. A stack of substates can be created and Soar will execute the oldest or highest one first. Then Soar will either modify the working memory in the apply phase or send the output to the external environment in the output phase.

One of the most interesting features of Soar is its learning mechanism or chunking. This mechanism enables Soar to create its own long-term knowledge or productions called a chunk [15, 16]. When Soar has insufficient knowledge to solve the current problem, it reaches an impasse which means a chance to learn. After the impasse was resolved, Soar generalizes and summarizes the substates, and records them into the production memory. In the future, if the same condition occurs, the situation will be immediately solved by using the chuck without reaching an impasse.

One of the significant capabilities of Soar is the excellent input/output interface. Soar can interact with the external world by using the Soar Markup Language (SML). Similar to XML packets, SML sends and receives Soar commands between the Soar architecture and the clients which can be either the external environment or Soar debuggers [17]. ClientSML is a set of classes containing the low-level details of XML commands available in C++, Java, and Tcl programming language. By using SML, Soar can easily communicate with the environment.

3. HARDWARE PLATFORM

An advanced, unassembled six-legged robotic platform by Parallax, Inc. [18], the HexCrawler, is the test platform used in this research. The hexapod's chassis is made from anodized aluminum and has slots for attaching any add-on device. The front bracket has two 26.6-degree-offset-angle sensor wings which provide broad field-of-view for the hexapod. The overall dimensions of the hexapod are approximately 16 inches in width measured from leg to leg, 20 inches in length, and 4.9 inches in height when squatting. The ground clearance is 3.5 inches. Its weight is roughly 4 pounds including all servos. Its payload capacity is 7.5 pounds and the payload space is 282 square inches [19]. Each leg of the hexapod is controlled by one horizontal and one vertical servo. A total of twelve analog Hitec HS-645MG Ultra Torque Metal Gear servos [20] generate the hexapod locomotion. The output torque of each servo is 133 ounce-inches at 6.0 volts. The speed of each servo is 300

degrees per second at 6.0 volts and no load. Figure 1 shows the hexapod with all the equipment on-board.

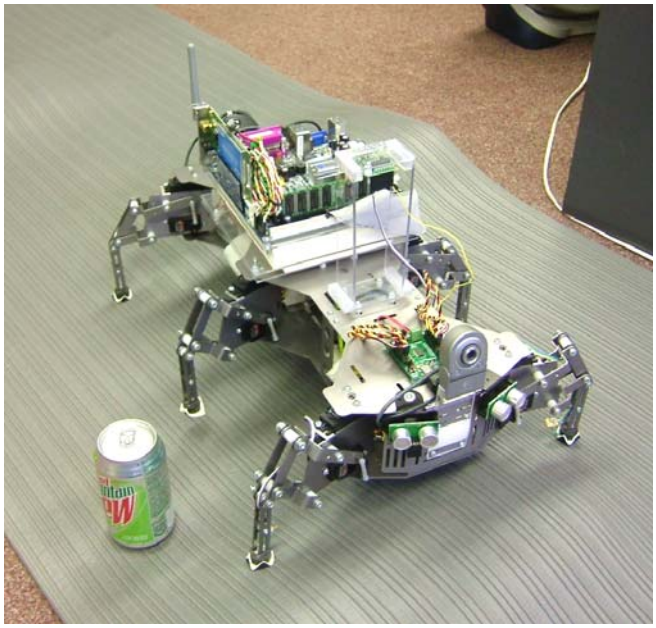

Figure 1 – The hexapod with all equipment on-board

The on-board computer on the hexapod is a Mini-ITX M-Series motherboard from VIA Technologies, Inc [21]. This ultra-compact x86 platform PC has a size of 7 inches square. It has an embedded 600 MHz VIA Eden™ processor which has low power consumption and does not require a fan. The board has many I/O ports including USB 2.0 ports and one RS-232 serial port. A 40-Gb IDE hard drive, 512-Mb memory, and a wireless LAN PCI adapter are also connected to the motherboard. The operating system is Windows XP; however, Linux could also be used.

The Parallax Servo Controller (PSC) by Parallax, Inc. [18] with a USB interface controls the position, rate, offset, and post delay of each servo. This board can control servos within a range of 180 degrees of rotation with a resolution of 0.18 degrees. The position of a servo may be requested at any time. The baud rate is adjustable between 2400 and 38K baud. This 1.8-inch-by-2.3-inch board has sixteen servo connectors, on/off and reset switches, connectors for networking with another PSC board, a connector for servo power supply, a USB connector, an on-board processor and a FT232BM chip which allows the USB-to-serial communication. The USB port also supplies the power for the processor while the separate battery is required for servos.

The BrainStem GP 1.0 module by Acroname, Inc. [22] is the interface between the sensors and the motherboard. The size of the BrainStem is 2.5 inches square. It has a 40 MHz RISC processor (PIC 18C252 microcontroller), four servo outputs, five 10-bit analog inputs, five digital input/output pins, two IIC ports, one serial port, one IR-sensor connector, and two power connections: servo power supply and the board power supply. Eleven 1K EEPROM slots are available for storing Tiny Embedded Application (TEA) software programs which are a subset of C. The BrianStem can run TEA code directly from the EEPROM slots. The BrainStem has three modes of operations: slave, TEA, and reflex. In slave mode, a host computer can directly send commands to the BrainStem by using supporting libraries in C, C++, and Java. In TEA mode, the BrainStem can run TEA applications from the EEPROM slots. In the reflex mode, the BrainStem can generate an automatic response according to the dynamic input changes without communication between the BrainStem and the host computer. For example, when the sonar sensor detects an obstacle, the servo can be stopped immediately. However, this mode is useful only when an output device is controlled by the BrainStem. All three operation modes can transfer small amounts of data among each other by using the 32-byte RAM buffer called scratchpad.

Two Devantech SRF05 Ultrasonic Rangers by Robot Electronics [23] were attached on the hexapod's front sensor wings; one on the left and another one on the right. Their size is 0.75 inches by 1.75 inches by 0.5 inches. Each sensor emits a conical beam, waits for its echo, and sends the duration to a controller. These sensors can detect any obstacle within a range from one inch to 13 feet.

A Devantech CMPS03 Electronic Compass by Robot Electronics [23] measures the bearing for the hexapod. Its size is 1.25 inches square. The compass operates by reading values from two magnetic field sensors perpendicularly installed to each other and computing the bearing relative to the magnetic field of the earth. Since the electronic compass is sensitive to electromagnetic fields, it must be carefully mounted in a position which has the least interference from other on-board devices.

A GPS 18 USB by Garmin International, Inc. [24] provides the robot's position on the earth. The GPS is 2.4 inches in diameter and is 0.8 inches in thickness. This FAA Wide-Area-Augmentation-System (WAAS) enabled GPS receiver has accuracy less than 15 meters in position for the GPS Standard Positioning Service (SPS) and less than 3 meters in WAAS mode. This GPS connects to a computer via a USB port and sends the information in a NMEA sentence format via a virtual communication port.

Two analog force sensors [25] were attached at the bottom of the two front feet of the hexapod. Each flat force-sensing resistor has an area of 0.2 square inches. The sensitivity level of the sensors is adjustable to the weight of different robots by using the external resistor. These sensors are connected to the BrainStem's analog input pins.

A color VGA CMOS webcam from Logitech [26] is attached on the front of the hexapod. It can capture still images of 1.3 megapixels and capture a video at up to 15

frames per second. This webcam will be used in future work.

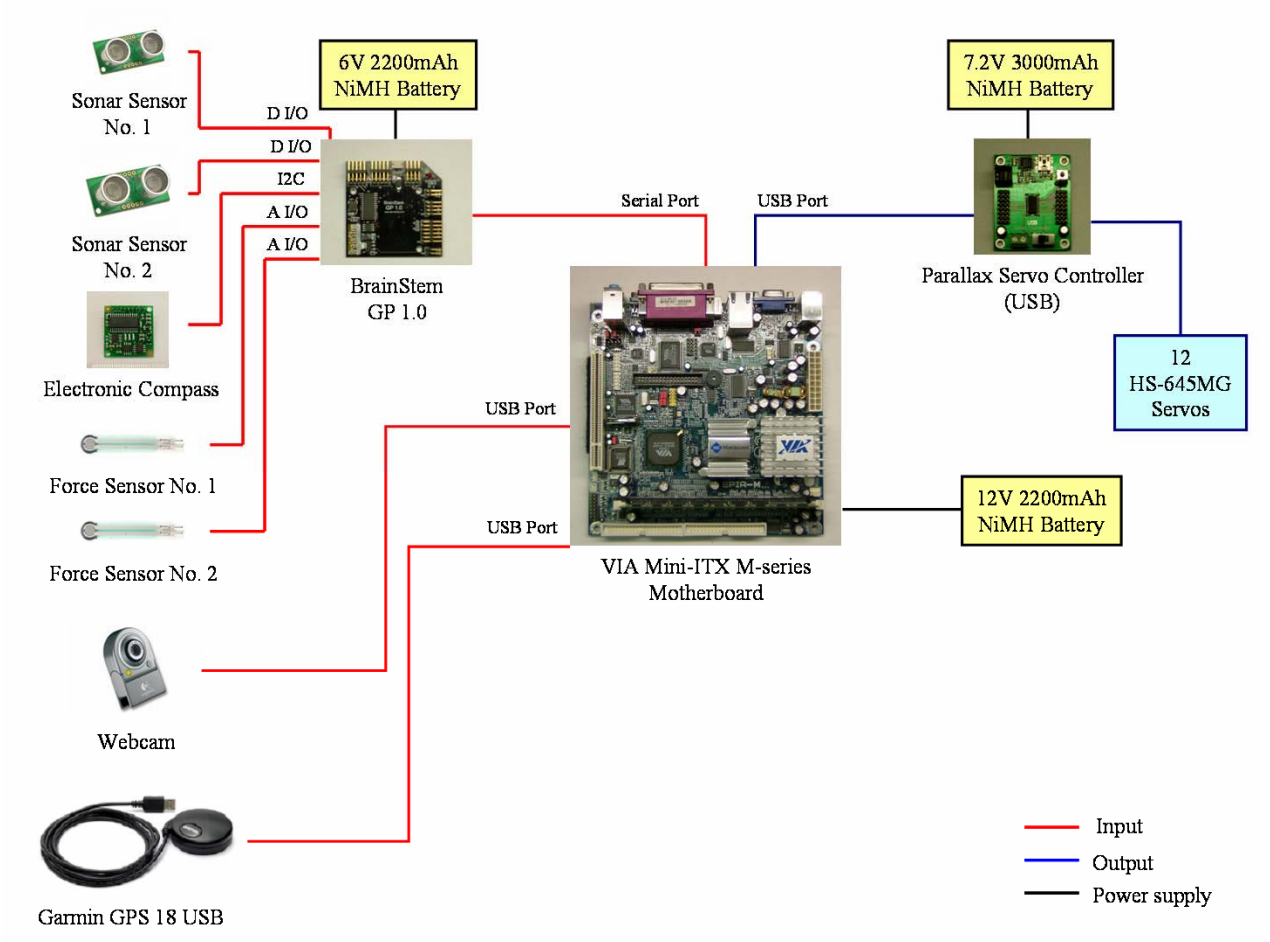

Figure 2 – Hardware connection diagram

The hexapod requires three battery units for the motherboard, the BrianStem module, and the twelve servos. The power unit for the onboard PC has a PW70 12V Mini-ITX converter and a battery. A ten-cell 12V 2200mAh rechargeable NiMH battery back provides power for the motherboard and the hard drive. A four-cell 6V 2200mAh NiMH rechargeable battery pack is the BrianStem power supply. A 7.2V 3000mAh NiMH battery pack supplies the power for all twelve servos via the PSC Board. Figure 2 shows the connection diagram of all hardware installed on the hexapod.

4. System Integration

The Java programming language is used as an interface between the hardware and the Soar architecture. There are many advantages of using Java in controlling a hardware system. First, the Java language is portable and platform independent. It was designed with the concept of "Write-Once-Run-Anywhere" (WORA) [27] which makes all Java classes reusable in any platform in future works. Second, Java has large collections of Application Programming Interfaces (APIs) which allow users to add various capabilities e.g. creating Graphical User Interface (GUI) by using Abstract Windowing Toolkit (AWT), communicating between two or more computers on the network by using Remote Method Invocation (RMI), interfacing with RS-232 ports and USB ports by using the Java Communication API (javax.comm) and the Java API for USB (jUSB) [28] respectively. Finally, the included multithreading capability for concurrent computing (Thread API) distinguishes Java from C and C++ [29]. In spite of its advantages, Java has some drawbacks especially in real-time applications. Users have limited memory allocation capabilities and have little access to the garbage collector which is automatically run as a background thread. In addition, users need to implement their own timing methods because Java does not provide a precise real-time clock [29]. However, the Real-Time Specification for Java (RTSJ) [30] and Sun Java Real-Time System 2.0 [27] have been developed for real-time applications (but it was not used here).

The Java™2 Platform, Standard Edition (Java SE) 5.0 was installed on the robotic system. Java Communications 2.0 API for the Windows platform is used to interface with the PSC Board (USB) and the GPS. The Java software packages for communicating with BrainStem modules

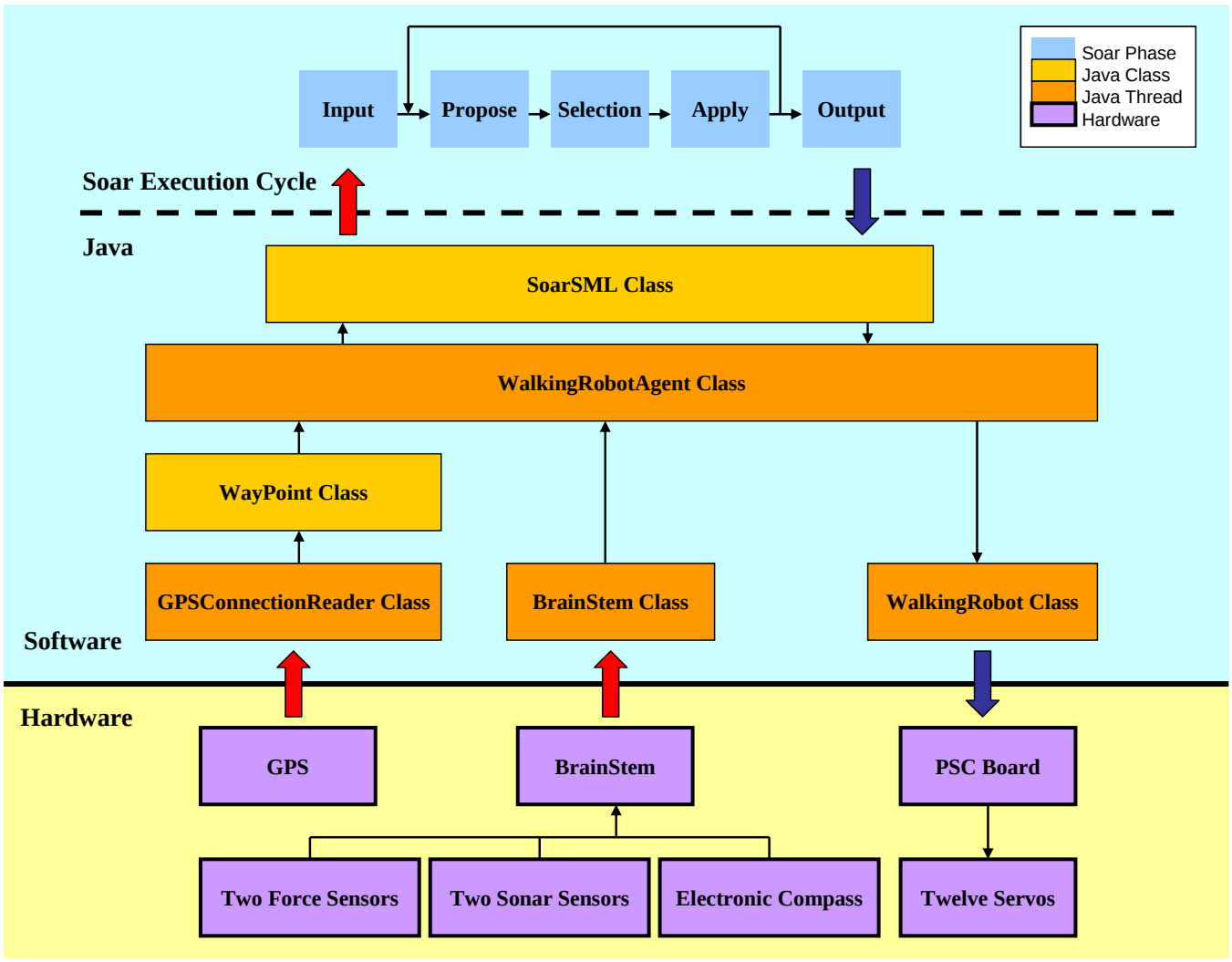

Figure 3 – Diagram of Hardware and Software Interactions

provided by Acroname and Soar Suite 8.6.3 are installed in the system too. A total of four Java packages were developed. The first package is for interfacing with PSC Board, controlling all servos, and generating leg locomotion by using the object-oriented approach. The second package is for communicating with BrainStem. Methods for data acquisition were implemented. The third package is for requesting and extracting data from the GPS. The fourth package is used to interface with the Soar architecture and to control all classes in the previous three packages. During execution, four Java threads are concurrently running to extract GPS data, to acquire data from the BrainStem, to send output to the PSC Board, and to control all processes including the interface with Soar.

Figure 3 shows the interaction between hardware and software on the robotic system. The data flow of the system begins when the sensor information is sent through either the GPS receiver or the BrainStem module by the Java interfaces. Then Java classes will process the information and send them to the input links of the Soar architecture. As soon as Soar detects the changes of its input links, the execution process will begin. Soar execution cycles will continuously operate until it has the output. Finally, the Java code receives the Soar output to control the hexapod locomotion via the PSC Board. All of these processes are controlled by the WalkingRobotAgent class.

The integrated hardware and software system showed in Figure 3, excluding the force sensors, was successfully tested in a parking lot at the Pennsylvania State University, University Park, PA [14]. Figure 4 shows the result of the test mission on the parking lot layout. The parking lot size is 36 m by 160 m. The upper section of the parking lot is the ramp up and down while the lower section is the flat surface. The hexapod started from the lower-left area and walked towards the target marked by the red cross on the lower-right area. The target position was specified by the GPS location. The black squares represent the obstacles randomly spreaded along the way. The hexapod successfully walked to the target while avoiding those

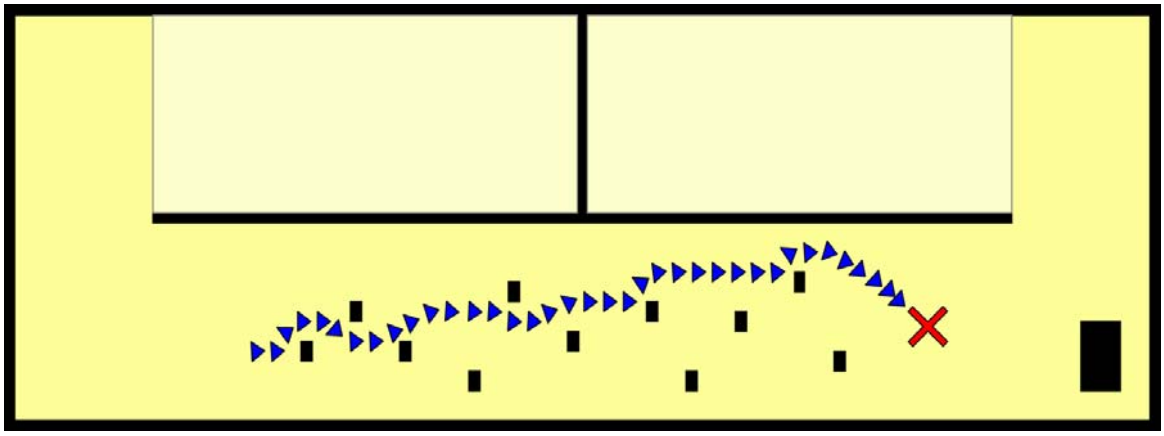

Figure 4 – The Hexapod's Path Controlled by Soar Architecture [14]

obstacles. The two sonar sensors helped provide obstacle avoidance. The robot was programmed to stop when it was within 3 meters of the target point. This experiment demonstrated the Soar architecture could be used to control the hexapod and the use of Soar subgoal to hierarchically control the separated tasks. This is very effective compared to developing all software for mobile robots from scratch every time. Additional sensors and additional Soar rules can be easily added.

5. RESULTS

Two force sensors were attached to the front left and right feet of the hexapod. In order to obtain the general and precise sensor information for the gait analysis, both sensors were connected to the analog-to-digital converter ports on the PCI-1716 high-resolution multifunction card controlled by the xPC Target 3.3 to measure the real-time data from

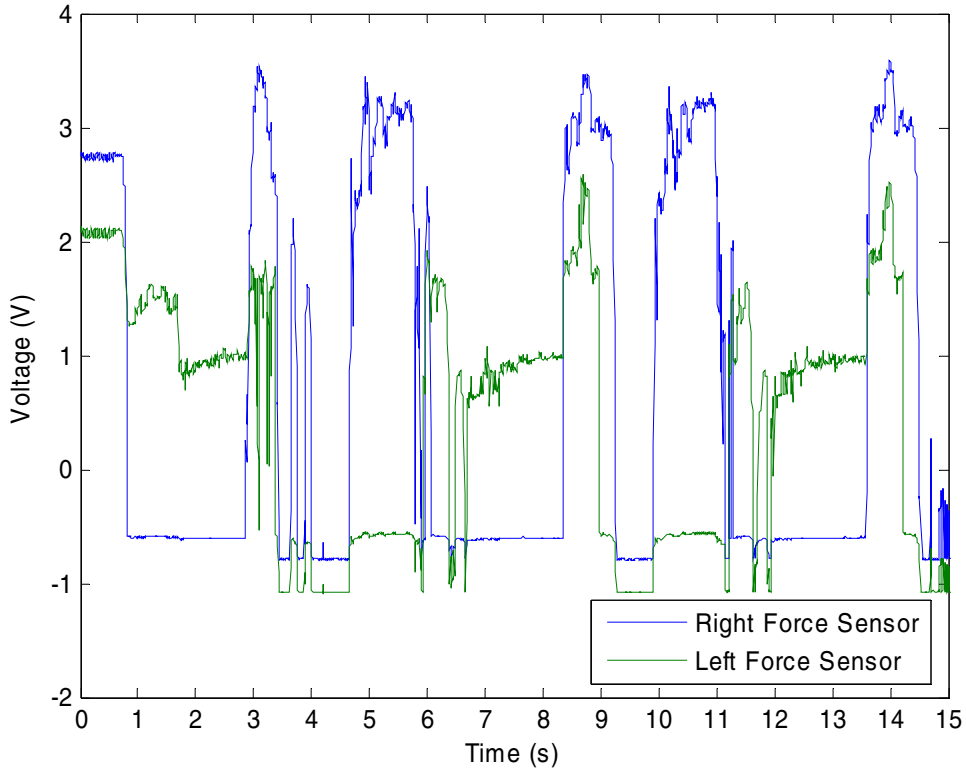

Figure 5 – Plot of Voltage from Left and Right Force Sensors of Walking Hexapod

the sensors. The system read the voltage from each sensor every 5 milliseconds for 15 seconds while the hexapod was walking forward on the hard, flat surface. The result is shown in Figure 5. The blue line represents the voltage of the right force sensor and the green one represents that of the left force sensor. The value is high when there is a force on the sensor and vice versa. In the first second, the voltages were constant because the measurement was started while the robot was standing in the parking stance. Note that the voltage values of both sensors are not identical to each other. When the hexapod started walking, each sensor value changes periodically due to the cycle of leg movement.

The walking forward sequence of the hexapod uses the tripod gait. The tripod includes the front and back legs of one side, and the middle leg of the other side. Two sets of tripods are alternatively supporting the weight of the hexapod and moving to the new position. Therefore, the voltage values from the front left leg and the front right leg should have the opposite values e.g. while one is high, the other one is low. From Figure 4, the results are in the predicted manner. The right sensor has high voltage during the third to sixth second while the left sensor has relatively low voltage. On the other hand, the left sensor has high voltage from the sixth to ninth second when the right sensor has comparatively low value.

During the period when the sensor is touching the ground, the voltage profiles have interesting characteristics. It reaches a high value and abruptly decreases to the lowest value in the next second. After the dramatic decrease, the voltage gradually increases again until the foot lifts over the ground. This unusual behavior corresponds to the walking forward gait of the hexapod. Since each leg of the robot has two degree of freedom, the hexapod moves by sweeping its feet backward in order to move itself forward. Its movement is generated by the friction made by all three tripod legs striding on the ground. The dynamic friction causes the absence and the increasing of the load on each supporting leg. The inconstant value of the voltage causes difficulty in measuring the force while the robot is walking. Furthermore, some short intervals in Figure 4 show the low voltage of both sensors at the same time; for example, between the fourth and fifth second, and between the ninth and tenth second. If the software mistakenly measures force during these intervals, it may interpret that both legs are not touching the ground. In fact, one of its front legs is touching the ground but it is sweeping on the floor. This issue emphasizes that the timing of the force measurement is crucial.

The difficulties of force measurement are not only caused by the hexapod locomotion (which generates the unusual voltage profile, as previously discussed), but also the complexity of the software system. Two issues arose in striving to get an accurate reading from the force sensors. First, the method to read a value from the analog-to-digital channel in the BrainStem package provided by Acroname is unstable and takes time to obtain the value. Generally, the reading of two sonar sensors and the electronic compass uses less than one second while the read analog method takes more than one second. Second, the standard Java language is not well suited for real-time applications and the available hexapod walking control class is a thread which runs independently and continuously. In order to implement the force sensors in the existing software system, the data acquisition method and the walking control class must be modified.

The TEA code for acquiring values from the force sensors was developed and pre-loaded into the EEPROM slot. When the TEA program was started, it continuously read the values from the analog-to-digital channels and wrote them to the scratchpad which is the RAM of the BrainStem. In the BrainStem package, the method to read data from the scratchpad was written. By acquiring data in this way, the force sensor value can be obtained within less than one second. However, the obtained readings from Java are not as accurate as the readings from the analog-to-digital converter shown in Figure 5. Therefore, the specific time to read the value from the force sensors must be identified.

The hexapod walking sequence is separated into two steps. The first one is moving the tripod using the front left leg forward and the other one is moving the tripod using the front right leg forward. Each sequence ends when the front leg stretches to reach the floor. At this point, the force sensor should provide the high value due to the weight loaded on the front-supporting leg. By measuring the force at the end of each step, the robot can sense whether its leg is touching the ground or not. However, the hexapod was not aware of its stepping on any obstacle due to the changing of weight distribution.

The Java code, which alternately reads values from the force sensors and sends walking command to the hexapod, was developed. This code stops the robot as soon as either one of its front feet does not touch the ground. The Soar productions for controlling the leg locomotion by using the force sensor were developed. The interaction between the Java code and Soar architecture is as follows: (1) all relevant Java objects will be created and all of the communication ports will be initialized and (2) the hexapod will walk one step and the value from the force sensor will be sent to the Soar input links. The Soar architecture will then execute the input and send the output to Java code. The output can be either step forward when the hexapod's feet are touching the ground, or backward and then turn to the other way to avoid a hole on the ground. Java reads the output from the Soar output link and sends the command to the servos via the PSC Board. Once the output is executed, the loop will start over again by reading from another force sensor. These processes will continue until the mission is timed out. In addition to the Soar productions, the Java

walking control class was revised in order to perform the mission by using the force sensors.

walking sequence was developed. The Soar production for controlling the hexapod leg locomotion on the unlevel ground was developed. By integrating the force sensors on

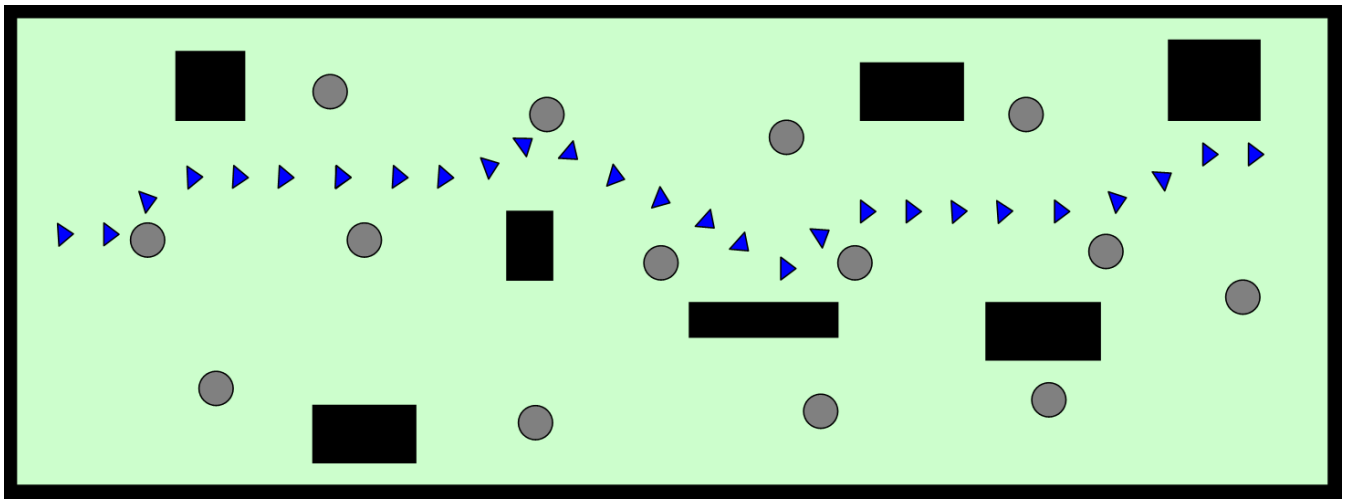

Figure 6 – The Hexapod's Path on Unlevel Terrain Controlled by Soar Architecture.
The grey circles represent the holes and the square black rectangles represent the small obstacles.

The Soar productions, which contain two subgoals, were developed: obstacle avoidance and hole avoidance. The obstacle-avoidance subgoal acquires data from two sonar sensors while the hole-avoidance subgoal utilizes two force sensors to navigate along the robot's path. One of the experimental paths is shown in Figure 6. The experiment was performed on a ten-feet-in-length anti-fatigue mattress with random holes and small obstacles. From the figure, the hexapod can walk along the terrain by switching between two Soar subgoals. The Soar learning mechanism was tested and the walking durations were recorded; however, there was no significant difference between the walking durations without chunking and with chunking turned-on. One possible reason is the long duration between each walking step of the hexapod. Between each step, the hexapod requires the amount of time to measure the value from the force sensor while its feet are contacting the ground. The total In addition to these two subgoals, Soar can be utilized for managing multi-tasks while the hexapod is avoiding the obstacle, going to the specific GPS location, and walking on the unlevel terrain mission.

6. Conclusion and Future Work

This paper described a hexapod robot and the software and hardware used to control it. Using the Soar cognitive architecture, the robot was able to avoid obstacles and walk to a pre-specified GPS location. In addition, two force sensors were attached to the front two feet of the hexapod. The force profile of a walking hexapod was examined. The data acquisition method for force sensors was implemented and the Java code to interface between the sensors and the robotic platform, the hexapod has the ability to sense the terrain and avoid holes and hills.

One limitation in this research is the limited readings from force sensors. The limited readings and the unequal weight distribution among tripod gait could not provide the sensing of the feet on any obstacle. This limitation might be solved by attaching force sensors to all of hexapod's feet and observing the pattern of the changing value on different kinds of terrains. In this case, the pattern recognition technique might be implemented to detect the alternation of the value from all force sensors. The next limitation is the insufficient number of force sensors. Two force sensors are insufficient to sense all the information from the terrain. Another limitation is the long duration between each step of walking command due to the waiting time for reading values from force sensors.

The Soar learning mechanism should be very useful for complex terrain. The combination of force sensors combined with other sensors will also be interesting to study. And finally, the cooperation with other UGVs and UAVs is possible.

References

[1] F. Naderi, D. J. McCleese, and J. F. Jordan, Jr., "Mars exploration," *IEEE Robotics & Automation Magazine,* vol. 13, pp. 72-82, 2006.

[2] S. Pinker, *How the Mind Works*. New York, NY: Norton & Company, 1997.

[3] D. J. Todd, *Walking Machines: An Introduction to Legged Robots*. New York: Chapman and Hall, 1985.
[4] A. G. H. Barett Mitchell, Brian C. Williams, "Search-based foot placement for quadrupedal traversal of challenging terrain," in *2007 IEEE International Conference on Robotics and Automation* Roma, Italy, 2007.
[5] K. V. Karl C. Kulling, Cristina M. Wilcox, "A foot placement planning algorithm for a walking quadruped rover," in *AIAA Infotech@Aerospace 2007 Conference and Exhibit* Rohnert Park, CA, 2007.
[6] P. D. N. John R. Rebula, Brian V Bonnlander, Matthew J. Johnson, Jerry E. Pratt, "A controller for the LittleDog quadruped walking on rough terrain," in *2007 IEEE International Conference on Robotics and Automation* Roma, Italy, 2007.
[7] D. Pongas, M. Mistry, and S. Schaal, "A robust quadruped walking gait for traversing rough terrain," in *2007 IEEE International Conference on Robotics and Automation*, Roma, Italy, 2007, pp. 1474-1479.
[8] Y. Go, Y. Xiaolei, and A. Bowling, "Navigability of multi-legged robots," *IEEE/ASME Transactions on Mechatronics*, vol. 11, pp. 1-8, 2006.
[9] Youcef Zennir and P. Couturier, "Multiactor approach and hexapod robot learning," in *2005 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, Espoo, Finland, 2005.
[10] L. N. Long, S. D. Hanford, O. Janrathitikarn, G. L. Sinsley, and J. A. Miller, "A review of intelligent systems software for autonomous vehicles," in *Proceedings of the IEEE Symposium on Computational Intelligence in Security and Defense Applications*, Honolulu, HI, 2007.
[11] A. Newell, *Unified Theories of Cognition*. Cambridge: Harvard University Press, 1990.
[12] R. M. Jones, J. E. Laird, P. E. Nielsen, K. J. Coulter, P. Kenny, and F. V. Koss, "Automated intelligent pilots for combat flight simulation," in *AI Magazine*, 1999, pp. 27-41.
[13] J. E. Liard and P. S. Rosenbloom, "Integrating execution, planning, and learning in Soar for external environments," in *Proceedings of the 8th National Conference on Artificial Intelligence*, 1990, pp. 1022-1029.
[14] O. Janrathitikarn, "A Use of a Cognitive Architecture to Control a Six-legged Robot," in *Aerospace Engineering*. Master of Science University Park, PA: The Pennsylvania State University, 2007, p. 128.
[15] J. F. Lehman, J. Laird, and P. Rosenbloom, "A Gentle Introduction to Soar, An Architecture for Human Cognition," 2006.
[16] J. E. Laird, C. B. Congdon, and K. J. Coulter, "The Soar User's Manual," Electrical Engineering and Computer Science Department, University of Michigan 2006.
[17] T. Software, "SML Quick Start Guide," 2006.
[18] http://www.parallax.com, 2006.
[19] http://www.crustcrawler.com, 2006.
[20] http://www.hitecrcd.com, 2007.
[21] http://www.via.com.tw/en, 2007.
[22] http://www.acroname.com, 2006.
[23] http://www.robot-electronics.co.uk, 2007.
[24] http://www.garmin.com, 2006.
[25] http://www.phidgetsusa.com, 2006.
[26] http://www.logitech.com, 2006.
[27] http://java.sun.com, 2006.
[28] http://jusb.sourceforge.net, 2006.
[29] D. M. Auslander, J. R. Ridgely, and J. D. Ringgenburg, *Control Software for Mechanical Systems: Object-Oriented Design in a Real-Time World*. Upper Saddle River, NJ: Prentice Hall, 2002.
[30] http://www.rtsj.org, 2006.

BIOGRAPHY

Oranuj Janrathitikarn *is a Ph.D. student in the Department of Aerospace Engineering, the Pennsylvania State University. She has a Bachelor of Engineering degree from Chulalongkorn University, Bangkok, Thailand, and a Master of Science in Aerospace Engineering from Penn State. Her master's thesis focused on the implementation of the Soar architecture on the six-legged robot. Her research interests are intelligent systems, cognitive architecture, and unmanned ground vehicles.*

Lyle N. Long *is a Distinguished Professor of Aerospace Engineering, Bioengineering, and Mathematics at The Pennsylvania State University. He is the Founder and Director of their Graduate Minor Program in Computational Science (www.csci.psu.edu). He was also the founding Editor-in-Chief of the Journal of Aerospace Computing, Information, and Communication (www.aiaa.org/jacic). He has worked at Lockheed Aircraft, Thinking Machines, and NASA. He has degrees from George Washington (D.Sc.), Stanford (M.S.) and Minnesota (B.M.E.). He is an IEEE Certified Software Development Professional (CSDP). He won the IEEE Gordon Bell Prize in 1993, and in 2007-2008 he was a Moore Distinguished Scholar at The California Institute of Technology. He is also a Fellow of the AIAA.*