

# Towards a Distributed, Cognitive Robotic Architecture for Autonomous Heterogeneous Robotic Platforms

Scott D. Lathrop and Christopher M. Korpela  
Department of Electrical Engineering and Computer Science  
United States Military Academy, West Point, New York 10996  
{scott.lathrop@usma.edu, christopher.korpela@usma.edu}

**Abstract**—*This paper describes an early phase towards a general, holistic, distributed robotic architecture capable of supporting different domain specific tasks autonomously or semi-autonomously on multiple platforms. Rather than focusing on individual components and algorithms, we take a systems level approach striving for a task-independent architecture. Our ultimate goal is to employ autonomous scouts that augment a military unit with additional reconnaissance and security assets. We envision one soldier controlling a mix of three to five ground and aerial robots in contrast to one soldier controlling a single robot as realized in currently deployed systems. Such a general robotic architecture may be useful in other domains to include law enforcement, search, and rescue, hospital logistics, and domestic use. To demonstrate interoperability this paper discusses the architectural design and current set of heterogeneous prototypes built with common hardware, software, and a distributed, web-based operator control unit (OCU). We explore future extensions that incorporate a cognitive architecture and work towards the Joint Architecture for Unmanned Systems (JAUS) compliance.*

## 1. INTRODUCTION

Current state-of-the-art military robotics solutions used in Iraq and Afghanistan are composed of platform and task specific hardware, protocols, interface drivers, and high-level control software. Usage requires one or more persons to tele-operate a single robot through an operator control unit (OCU). Examples of such systems are the Foster-Miller TALON™, iRobot® Packbot®, and the Mesa Robotics Matilda™. The U.S. military uses these systems to perform jobs in hazardous situations such as Improvised Explosive Device (IED) disposal and the result is extraordinary in the terms of saved soldiers' lives.

Currently fielded approaches have hardware and software that is exclusive to the respective platform. These stove-piped systems are non-interoperable and resource intensive. Units must maintain separate stockpiles of repair parts, contractor support is required to troubleshoot different configurations, and soldiers must learn how to operate different platforms and OCUs.

These systems impose a constraint of at least one tele-operator per robot where what we desire is one soldier controlling multiple (i.e. 3-5) robots.

Some platform specific hardware devices, low-level protocols, and interface drivers are unavoidable as the physical hardware constrains low-level software interactions. Nonetheless, achieving a general, high-level task-independent architecture controlling a robot's overall goals, planning, reasoning, decision-making, learning, and execution is necessary if we are to realize autonomy. Our goal is to employ multiple (i.e. 3-5) autonomous, robotic scouts that can augment a unit with additional reconnaissance and security assets without requiring a one-to-one soldier to robot mix.

## 2. ARCHITECTURAL DESIGN GOALS AND CONSTRAINTS

To support that goal our architectural design takes a holistic approach where we consider how one functional component of the system, whether it be hardware, software, or a communication protocol constrains another. Table 1 describes our design goals and constraints. The constraints are in agreement with the Joint Architecture for Unmanned Systems (JAUS) [1] that desires platform independence, mission isolation, and computer hardware independence.

The first and second goals require the development of a distributed command and control architecture and homogenous, high-level, robotic control software where new knowledge and behavior is relatively easy to incorporate. The first three constraints tie directly into these first two goals, as supporting autonomous behavior, especially when there is disrupted communication between operator and robot or robot and robot, requires on-board, robust intelligence necessary to handle these situations. From a hardware perspective, these constraints imply that sufficient power, processing, memory, and storage must be on the platform. From a software perspective, the high-level architecture should be able to support task-independent, intelligent behavior. Communication protocols must support interoperability and dynamic, ad-hoc, mesh networks.

Design goal three requires a consistent interface between low-level processing and a high-level controller so that when developers add new or improved sensors or actuators to the platform, they only have to construct or modify low-level drivers. Changes to the individual, high-level control software or the system's command and control software

should be minimized. Such a robotics device interface is where software such as Player [2] and Microsoft Robotics Studio [3] focus much of their efforts.

**Table 1: Design Goals and Constraints**

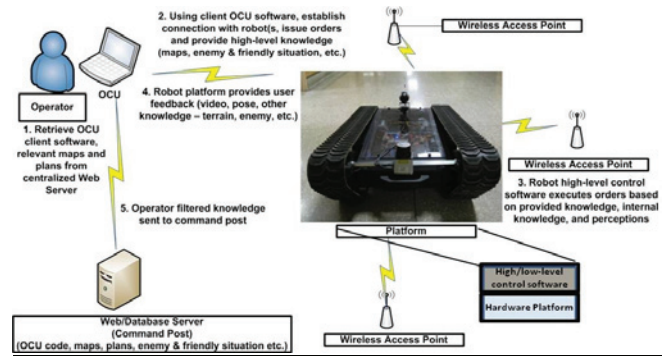
Goals	Constraints
1) Homogenous high-level framework—able to apply distributed, high-level control software to multiple, heterogeneous platforms. 2) Extensible--relative ease of incorporating new knowledge and behavior whether programmed or learned. 3) Heterogeneous adaptability—ability to incorporate new hardware (sensors, actuators, platforms) where changes are localized to byte-level processing units. 4) Robust communication—ability to communicate with humans and other robots in austere environments.	1) Decentralization: Intelligence must reside on platform. 2) Local decision-making: Robotic decision-making does not require access to the entire global state. 3) Uniformity: Must support uniform high-level architecture across platforms. 4) Distributed: Must support distributed and hierarchical command and control across a wireless communications network. 5) One-to-many: Must support one operator/OCU for many platforms rather than one-to-one.

Figure 1 shows the initial architecture supporting some of these goals and constraints. In the center of the figure is the particular platform. The platform contains the sensors, actuators, low-level software to interact with these devices, intermediate or robotics device interface software to support basic sensing and movement, and the high-level software to support decision-making, problem solving, planning, reasoning, and communication.

The operator, shown in the upper left of Figure 1, interacts with the robotics platform with an OCU. To control robots, the user initially logs into a distributed, web-based application that provides the client software and other information such as maps and plans (step 1). The application server hosting this information may be at a centralized location (such as a military command post), but the operators are distributed throughout the operating environment. The advantage is that system administrators and functional managers can make changes to the client software and initial information from a central location.

After the operator's OCU receives the client software, the operator can connect to one or more robots within operational range of the underlying wireless network and provide initial knowledge to the robots such as maps, plans, high-level commands, etc. (step 2). In a military operational environment, this wireless network would be an ad-hoc,

mesh network. In domains with fixed structures such as a hospital or home, such a network could be permanent.



**Figure 1: Distributed Robotic Architecture**

The robotics platform receives high-level commands from the operator and begins executing the commands either autonomously or semi-autonomously (step 3). During execution, the robot may provide feedback to the user, such as visual video feeds or symbolic information (step 4). The operator may further filter and forward such information to the centralized server.

### 3. PROTOTYPE PLATFORMS

We have developed two primary prototypes. The first, described in [4], incorporates a three-tiered, distributed architecture (platform, client OCU, and web server). This prototype enabled the operator to tele-operate and receive visual feedback from a iRobot® Create, Lynxmotion 4WD Chassis, and a Mesa Robotics Matilda™, focusing on the overall infrastructure and tele-operation. It demonstrated the ability to control a set of robots using a web-based interface.

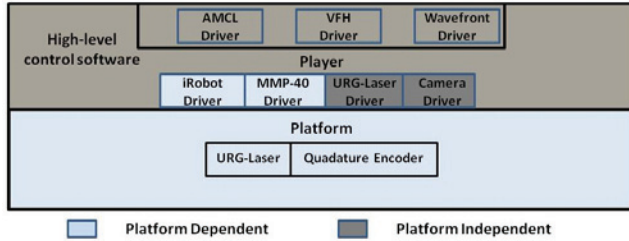
The second set of prototypes focuses on high-level control software, a standard low-level device driver interface, and autonomous movement in a controlled, indoor environment. For the second scenario, we developed two prototype platforms to demonstrate a single operator controlling more than one robot and a high-level software framework re-used across multiple platforms.

In this prototype, the high-level intelligence resides on the robot rather than on a central processor. Figure 2 illustrates this architectural design. Leveraging Player [2], we separate the low-level control drivers for sensors and actuators from the high-level control algorithms. The platforms used in the prototype include the iRobot® Create [5] (Figure 3) and the MMP-40X from The Machine Labs [6] (Figure 4).

#### 3.1. Hardware Platforms

The iRobot Create served as our platform for initial development and testing. Figure 3 shows the Create with a mounted Pico-ITX processor for onboard processing and a mounted Hokuyo URG-04LX Laser to support navigation. To support testing we used the Stage [2] simulation environment and the hallways within an academic building that provides an ideal, controlled environment because of

the existing IEEE 802.11 network, wide corridors, and a level ground floor plan. For now, we choose to leverage the IEEE 802.11 standard for communication due to its widespread use, ease of implementation, and the existing infrastructure at our organization. Future work may address alternative wireless protocols such as IEEE 802.15 (Zigbee).



**Figure 2: Implemented Platform Architecture**



**Figure 3: iRobot Create with mounted components**



**Figure 4: MMP40X with mounted components**

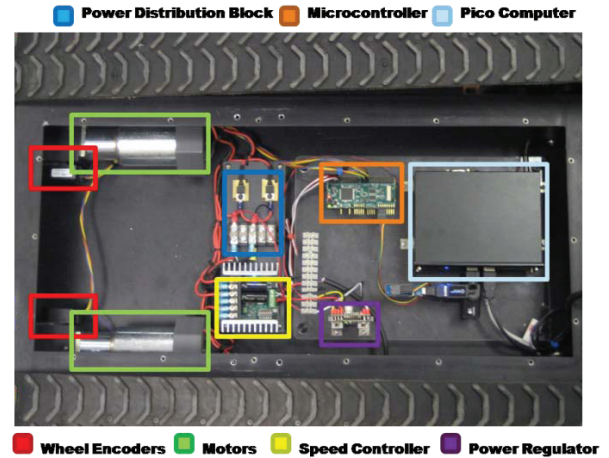
To demonstrate the architecture working across heterogeneous platforms, we chose a MMP-40X robotic chassis as it gave us a potential future benefit of being able to operate in a tactical environment (Figure 4). The MMP-40X has two DC motors. Each motor is configured with a multi-input speed controller and 500 counts per revolution quadrature wheel encoders to drive the chassis. For power, the chassis includes two, 25.6 V Li-PeFO rechargeable batteries with a charge capacity of 7.2 amp-hours.

While the stock MMP-40X chassis provided the necessary power, motor, and wheel encoder subsystems, we had to augment the chassis with additional components to support autonomous navigation. Additional components included an URG-04LX laser to support localization; a video camera to support operator feedback; a microcontroller for processing the encoders and driving the tracks; a USB 802.11 wireless adapter for communication with the OCU; and the Pico-ITX embedded computer running the Ubuntu Linux operating system (Table 2 and Figure 5).

**Table 2: MMP40X Component Augmentation**

Component	Model
<b>Power Regulator</b>	80W picoPSU
<b>Wheel Encoder</b>	U.S. Digital Optical Encoder
<b>LIDAR</b>	Hokuyo URG Laser-04LX
<b>Camera</b>	Logitech Communicate STX Webcam
<b>Motor</b>	Dimension Engineering Motor Driver
<b>Controller</b>	
<b>Wireless NIC</b>	Alfa 802.11g Wireless USB Adapter
<b>Microcontroller</b>	Robostix with an Atmel ATmega128
<b>Microprocessor</b>	1GHz, 1 GB RAM VIA EPIA Pico-ITX mainboard

We mounted the Pico with a polycarbonate mounting plate to the bottom of the chassis making it trivial to remove the processor during development. To provide the proper power voltages to the sensors, we connected a wide-input ATX power regulator to the chassis batteries. To mount the URG-04LX laser and camera sensors we made minor external modifications to the chassis (e.g. installed a crossbar across the front of the robot so that the laser had better visibility). Finally, we fabricated and placed a clear polycarbonate lid over the chassis for protection of the system internals enabling a better view for troubleshooting and presentation.



**Figure 5: MMP40X Internal Components**

### 3.2. Platform dependent software

For processing low-level sensory input, we leveraged Player [2] as it provides off-the-shelf software drivers for the URG laser and video camera (Figure 2). Player also has an iRobot Create driver that provides odometer readings and enables higher-level control software to set the speed and direction. For the MMP-40X, we wrote two, low-level software components to provide odometer sensing and transmit speed and directional control to robot's tracks.

First, software for the Robostix microcontroller read the tracks' quadrature encodings adjusting the velocity of each track based on throttle and steering information. The quadrature wheel encoders provide tick counts for each



track on the robot. Our microcontroller’s implementation used a 32-bit microsecond counter as a timer between encoder counts and transmits this data approximately every 20 ms to a second software component, a MMP40-X driver running on an embedded Pico computer.

The MMP40-X software driver converts the encoder counts and time into a pose (i.e. location and orientation) as input to a localization algorithm (discussed below). Likewise, high-level control software passes the desired speed and direction to the driver. We implemented (P)roportional and (I)ntegral portion of PID control to determine desired throttle and steering information. The microcontroller software induces the motors by sending pulse width modulation (PWM) signals to the motor controller.

### 3.3. High-level control software

The two robotics platforms were different with variations in the sensor and motor suite (specifically for odometer readings), but the high-level control software and the distributed, client software on the OCU were the same on both. We used the Adaptive Monte Carlo Localization (AMCL) driver for localization, the Vector Field Histogram (VFH) driver for local obstacle avoidance, and the Wavefront algorithm for global path planning. AMCL uses particle filters and Bayesian methods as described in [7]. It assumes the robot has access to a bitmap of the environment relying on hypotheses formed from the laser and odometer readings to determine the pose probabilistically.

Using the robot’s pose and the operator determined goal the Wavefront algorithm determines a set of waypoints for the robot. The VFH driver uses these waypoints to provide local path planning and dynamic obstacle avoidance from waypoint to waypoint. VFH transmits the speed and direction for the robot to the robot’s platform dependent drivers to command the platform’s servo motors.

### 3.4. Distributed command and control software

The OCU consists of a basic laptop running Ubuntu Linux and our customized client software written in Java and C++ (Figure 6). The graphical user interface enables the operator to select a robot and set its next checkpoint. The robot then autonomously moves to that checkpoint in real time. A central server (lower left of Figure 1) maintains the OCU code, maps, and, in the future, any other initial knowledge that should be provided to the robot prior to its mission. We use the Java Network Launching Protocol (JNLP) enabling the OCU machine to download the client application and any of its dependencies through a web-based interface. The protocol caches those software components and resources and does not download them again, unless they have been updated on the server.



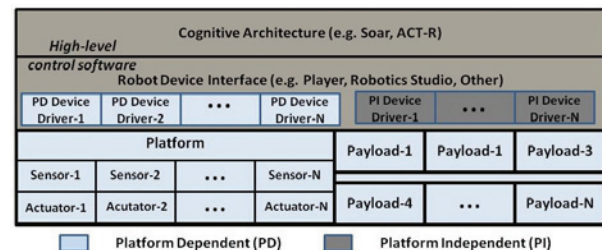
**Figure 6: Prototype OCU**

## 4. FUTURE EXTENSIONS

Although the first two iterations of the architecture partially met our design goals to include a homogeneous high-level framework and support for distributed operations, the prototypes had numerous shortfalls. The high-level control algorithms are *platform-independent* but not *task-independent* as the robot can only perform a single task (navigation). An operator is unable to provide the robot with a mission’s task and purpose and allow it to carry out that mission. The high-level software controller cannot incorporate new knowledge (e.g. task knowledge such as the current enemy situation or tactical doctrine). The implemented prototypes do not have any capability for reasoning, decision-making, learning, or communicating. In short, the system is inflexible and brittle.

Robotics device frameworks, such as Player [2], provide an abstraction layer that enables the interaction with low-level sensors and actuators. The support for high-level control typically takes algorithmic approaches providing short-term gains to a specific problem (e.g. navigation, grasping, etc). Typical uses have the robot perform one or two specific tasks. These approaches often lead to long-term issues when trying to integrate different types of tasks, as the design does not account for the constraints of each component.

The challenge is how to achieve such capability in robotic systems. We believe part of the answer is by integrating a cognitive architecture with the robotic platform to serve as a high-level controller (Figure 7). A cognitive architecture specifies the underlying infrastructure for an intelligent system that is constant across task domains. This infrastructure includes short- and long-term memories; representation of knowledge in those memories to include procedural (i.e. skill knowledge), semantic (i.e. facts, beliefs), and episodic (i.e. memories of past events); and learning mechanisms [8].



**Figure 7: Platform Architecture Proposal**

To integrate a cognitive architecture with a robotic platform, there are several issues that have to be addressed—three of which we will highlight here. These issues are not unique to cognitive architectures, but rather issues that any robotics system must address if the goal is to achieve general behavior. These issues include the symbol grounding problem, unification of the behavior-based versus deliberate approaches, and robot-robot and human-robot communication. Our hypothesis is that the current state of cognitive architectures is prepared to address these issues.

First, there is the anchoring or symbol grounding problem. Anchoring is a process that creates and maintains a mapping between symbols and the raw sensor data referring to the same physical entity [9]. The problem is how to implement anchoring in a robotic system where the implementation is general and not specific to any one task.

One possible approach to address the anchoring problem is augmenting cognitive architectures with perceptual-based memories and processing units. For example, architectures that support mental imagery processing may be useful as the reasoning combines raw sensory input with symbolic knowledge [10]. Roy et al. [11] have already shown how using mental imagery in robots facilitates human-robot communication while Kuipers [12] has demonstrated how robots can learn and use general spatial representations as they explore their environment.

The second issue is a central point made by Arkin [13] in the development of hybrid reactive (i.e. behavior-based) and deliberative robotics architectures. This issue is how to develop a unifying methodology taking into account an understanding of the user's intent and the world beyond sensor input while maintaining real-time and robust reactivity to sensory inputs.

Orthogonal to this issue is the use of closed loop versus open loop control systems. In most robotic architectures, whether they are behavioral- or deliberative-based, the designer has to decide whether to use closed loop or open-loop control. In closed-loop controllers, the robot senses the environment and responds every cycle, which gives the robot reactive behavior in dynamic environments. Such a controller may overwhelm the robot's perceptual resources without an attention mechanism to focus its sensors. On the other hand, in an open loop controller, the robot may attempt to execute an extended sequence over multiple cycles while ignoring its senses. This provides efficient execution but is only appropriate for more complex skills (e.g. path planning) where constant interaction with the environment is not necessary. Typical robotic architectures commit to either a closed or open-loop controller.

An advantage of using a cognitive architecture is that the architecture's decision procedure can use task knowledge in deciding where on the closed/open-loop spectrum its behavior should fall rather than committing to one or the other [14]. For example, in the cognitive architecture Soar,

the decision cycle is closed loop but it may implement open loop behavior using its hierarchical subgoal mechanism that breaks a complex, abstract task (e.g. planning) into more refined subtasks. If during that reasoning process a more urgent sensory input arrives, the subgoal mechanism supports reaction to the more urgent task.

For example, in [10], a simulated robotic scout may be imagining the current situation to insure its team has good observation over an approaching enemy force. If during that thought process the scout's teammate communicates that it has sighted an approaching enemy, the analysis processing can pause, react to the immediate communication, and then continue with its deliberative thought processing if the situation warrants. The context switch between deliberate and reactive behavior is based on task knowledge and directly supported by the architectural mechanisms.

The last issue is that current robotic architectures do not commit to a robot-robot or robot-human communication protocol. JAUS [1] proposes a communication standard between subsystems and components that communicate across a network. The problem with JAUS-like protocols are that they address communication at the byte transfer level rather than the semantic level, which is required if we desire flexible robotic behavior. Cognitive architectures have been used in natural language processing [15], and again may provide some relief, but an interface between a JAUS-like communication protocol and a natural language's semantics will have to be considered. Additionally, we have to commit to a more robust and scalable underlying communication protocol, where the assumption that there is an existing infrastructure can be relaxed. Currently we are evaluating IEEE 802.15 (Zigbee) protocols.

## 5. RELATED WORK

Many researchers are investigating general autonomy. Much of the focus is on a common controller for multiple, heterogeneous robotic platforms. For example, Space and Naval Warfare Systems Center [16] has developed an unmanned vehicle and operator control interface capable of controlling and monitoring multiple sets of heterogeneous systems. Grocholsky et. al. [17] have developed algorithms and control frameworks to enhance cooperation between ground and aerial robots.

There have been others who are either planning [14] or have demonstrated [18, 19] the use of cognitive architectures as high-level controllers for robots. For example, Trafton et al. [18] have interfaced a cognitive architecture with a Nomad 200 to mimic a child's hiding behavior in a game of hide-and-go-seek. While we are focusing more on the end-to-end distributed system, these researchers are focusing on the interface between the cognitive architecture and the robot device interface—work which we plan on leveraging.

Albus [20] argues that the 4D/Real-time Control System (4D/RCS) architecture has achieved this integration.

4D/RCS relies on an internal world model of the external environment that includes both symbolic and iconic memories. It uses this world model for perception (e.g. focusing attention and grouping) and behavior (e.g. decision-making and planning). 4D/RCS addresses many issues that one has to consider when building an autonomous system, but it does not incorporate some mechanisms necessary for intelligence (e.g. learning) [19].

## 6. CONCLUSIONS

We are working towards a distributed, cognitive robotic system where one operator controls 3-5 heterogeneous robotic platforms with the goal of using these platforms as autonomous scouts. This goal requires a system that is general in nature. We have demonstrated a high-level architecture for such a distributed robotic system on two different prototype systems and propose augmenting each robotic platform with a cognitive architecture to enable behavior across many task domains. The cognitive architecture serves as the high-level controller and the primary communication interface between a human operator and the robotic platform. We are also evaluating the use of alternative, low-level communication protocols to provide a robust infrastructure in austere environments.

## 7. ACKNOWLEDGEMENTS

We would like to thank the students involved in this project for their outstanding work: Alan Adame, Alex Diaz-Martinez, Neil Milchak, Richard Miles, David Nelson, Sarah Noreen, and Andrew Thompson. The fabrication and construction of the robots was largely due to the work of Robert McKay and Frank Blackmon. Network and systems support provided by Gaylen Wong<sup>1</sup>.

- [1] "The Joint Architecture for Unmanned Systems (JAUS)," [Online]. Available: <http://www.jauswg.org/>.
- [2] B. P. Gerkey, R. T. Vaughan, and A. Howard, "The Player/Stage project: Tools for multi-robot and distributed sensor systems," *Proc. International Conf. on Advanced Robotics (ICAR)*, Coimbra, Portugal, 2003, pp. 317-323.
- [3] "Microsoft Robotics Studio," [Online]. Available: <http://msdn.microsoft.com/en-us/robotics/default.aspx>.
- [4] C. Korpela and B. Ring, "A networked control system for heterogeneous robots," presented at the IEEE International Conf. on Technologies for Practical Robot Applications, Woburn, Massachusetts, 2008.
- [5] "iRobot<sup>TM</sup>," [Online]. Available: <http://www.irobot.com>.
- [6] "MMP-40 Mobile Robot Platforms," [Online]. Available: [http://www.robotmarketplace.com/store\\_machinelab\\_mmp40.html](http://www.robotmarketplace.com/store_machinelab_mmp40.html).
- [7] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, "Robust Monte Carlo localization for mobile robots," *Artificial Intelligence*, vol. 128, pp. 99-141, 2001.
- [8] P. Langley, J. E. Laird, and S. Rogers, "Cognitive architectures: Research issues and challenges," *Cognitive Systems Research*, vol. 10, pp. 141-160, 2009.
- [9] S. Coradeschi and A. Saffiotti, "An introduction to the anchoring problem," *Robotics and Autonomous Systems*, vol. 43, pp. 85-96, 2003.
- [10] S. D. Lathrop and J. E. Laird, "Extending cognitive architectures with mental imagery," *Proc. Second Conf. on Artificial General Intelligence*, Arlington, VA, 2009, pp. 97-102.
- [11] D. Roy, K.-Y. Hsiao, and N. Mavridis, "Mental imagery for a conversational robot," *IEEE Transactions on Systems, Man, and Cybernetics--Part B: Cybernetics*, vol. 34, pp. June 2004, 2004.
- [12] B. Kuipers, "The spatial semantic hierarchy," *Artificial Intelligence*, vol. 119, pp. 191-233, 2000.
- [13] R. C. Arkin, *Behavior-Based Robotics*. Cambridge, MA: MIT Press, 1998.
- [14] D. P. Benjamin, D. Lyons, and D. Lonsdale, "ADAPT: A cognitive architecture for robotics," presented at the 2004 International Conf. on Cognitive Modeling, Pittsburgh, PA, 2004.
- [15] J. Ball, A. Heiberg, and R. Silber, "Toward a large-scale model of language comprehension in ACT-R 6," *Proc. 8th International Conf. on Cognitive Modeling*, Ann Arbor, MI, 2007, pp. 163-168.
- [16] D. N. Powell, G. Gilbreath, and M. H. Bruch, "Multi-robot operator control unit," presented at the SPIE 6203: Unmanned Systems Technology VIII, Defense & Security Symposium, Orlando, FL, 2006.
- [17] B. Grocholsky, J. Keller, V. Kumar, and G. Pappas, "Cooperative air and ground surveillance," *IEEE Robot Automation Magazine*, vol. 13, pp. 16-26, 2006.
- [18] J. G. Trafton, A. C. Schultz, D. Perzanowski, M. D. Bugajska, W. Adams, N. L. Cassimatis, and D. P. Brock, "Children and robots learning to play hide and seek," presented at the 1st Annual Conf. on Human Robot Interaction, Salt Lake City, UT, 2006.
- [19] J. E. Laird "Toward Cognitive Robotics," *Proc. SPIE Defense and Sensing Conf.*, Orlando, FL, 2009, pp.
- [20] J. Albus, "4D/RCS: A reference model architecture for unmanned ground vehicle systems," presented at the SPIE 16th Annual International Symposium on Aerospace/Defense Sensing, Simulation and Controls, Orlando, FL, 2002.

<sup>1</sup> Views expressed are those of the authors and do not reflect the official position of the US Military Academy, the US Department of the Army, the US Department of Defense, or the US government.