BY KIMON P. VALAVANIS,
LEFTERIS DOITSIDIS,
MATT LONG, AND
ROBIN ROBERSON MURPHY

A Case Study of Fuzzy-Logic-Based Robot Navigation

Validation of a Distributed Field Robot Architecture Integrated with a MATLAB-Based Control Theoretic Environment

This article presents fundamental aspects of a multilayer, hybrid, distributed field robot architecture (DFRA), implemented in Java using Jini to manage distributed objects, services, and modules between robots and other system components [39]. It is designed for heterogeneous teams of unmanned robots operating in uncertain/hostile environments. Emphasis is given to the control theoretic lower-level MATLAB-based environment (supported and integrated into the Java-based framework using the JMatLink Java class library [30]), experimentally validated by implementing simple prototype support modules for outdoor mobile robot navigation.

Derivation, implementation, and testing of such a distributed architecture bring together the diverse fields of distributed artificial intelligence, human robot interaction, and multiagent systems, combined with control theoretic approaches. This is where the main contribution and novelty of this article lies: although the DFRA is implemented in Java/Jini, the MATLAB environment allows for mathematical control theoretic research and experimentation and for rapid prototyping of behavioral and control modules and services. Wrapping the MATLAB workspace environment with JMatLink, in conjunction with the Jini distributed object platform, allows modules and services implemented as native interpreted MATLAB code to be accessed as remote and distributed objects and to be directly incorporated into behavioral architectures, resulting in acceleration of development and added flexibility of implementation. This combination is not well represented in the literature and no complete approach has been published.

Experimental validation is demonstrated by implementing simple prototype support modules, like 1) a time-history laser filter module; 2) a heuristic geographic positioning system (GPS)-based pose detection module; and 3) fuzzy logic controllers utilizing laser, GPS, and odometers as inputs.

The laser and GPS modules are not meant to challenge the state of the art of robot positioning or sensor filtering; their purpose is to show rapid development and testing of these modules on real robots operating in the field, justifying and demonstrating the abilities of the overall architecture to integrate disparate

1070-9932/06/$20.00©2006 IEEE

components of varying levels of sophistication and development into a unified functional whole. For example, the pose detection system that provides adequate positioning data in the outdoor experimental environment demonstrates how the architecture is used to generate useful test-support modules rapidly and with a minimum of effort. Even though this module is almost minimally rudimentary, it is shown to be effective in supporting the derived fuzzy logic controllers. These simple modules in no way compare to or challenge state-of-the-art positioning systems and sound mathematical approaches reported in the literature (for example, see [41] and [42]).

Related research is presented next, followed by a discussion of the DFRA and MATLAB integration. The support module development is presented next, along with details of the fuzzy logic controllers for outdoor navigation. Experimental results, discussion, and conclusions complete the article.

Related Research

A recent review of autonomous robot control architectures is presented in [20]. Behavioral robotics architectures are presented in [2]–[4], [31], [34], [38], [39]. The DFRA is an example of an architecture falling within the behavioral robotics paradigm.

Utilization of MATLAB as the primary computing environment for mobile robot control experimentation has been reported in [32] and [33], in which utilities in the form of standard MATLAB function calls have been developed to allow access to all sensors and actuators, and these could be used in any standard MATLAB script. These systems also exemplify architectures explicitly intended to provide as few restrictions on robot control programming as possible while allowing platform transparent implementation.

In the general robotics and engineering cases, integration of MATLAB into Java-based systems has been addressed in [35] and [40]; however, MATLAB does not run on remote self-powered agents, such as autonomous mobile robots, in its full interpreted form .

Methods employing fuzzy-logic-based mobile robot navigation are reported in [5], [7], [10]–[16], and [17]. Most of these address navigation in indoor structured laboratory environments, while [18] and [19] discuss outdoor navigation using fuzzy systems. Waypoint navigation in which vehicles move in predefined environments is reported in [26] and [27]. Outdoor environment navigation using odometer data has been proven inadequate due to significant cumulative odometer errors [21], [22]. The use of absolute position sensing such as GPS is generally considered to be essential for successful outdoor navigation [23]–[25], at least within the confines of current autonomous field robot localization.

The design of the fuzzy logic controllers reported here differs from related work in [18] and [19]. In [18], dead reckoning is used requiring a priori knowledge of initial position. It is not robust against cumulative odometer errors, and at times the robot remains stationary during execution. The approach followed in [19] for outdoor navigation used small lab-based robots to prototype controllers for large outdoor vehicles in agricultural environments. The indoor robots used hierarchical fuzzy controllers having a set of different behaviors, including obstacle avoidance, goal seeking, and edge following, which were then transferred to outdoor vehicles. However, that work was applied mainly to corridor and edge following and used an infrared (IR) beacon for homing on goal positions, thus avoiding the need for absolute position knowledge. This, of course rules out the general case of navigation to a novel unvisited location because it requires someone or something to place the IR beacon in the first place. Finally, compared with reported research in [5]–[7], the fuzzy controllers introduced here use GPS and laser data, and they are applied in outdoor environments, as opposed to using only sonar sensor data for indoor navigation.

Distributed Field Robot Architecture and Integration with MATLAB

The DFRA, presented in detail in [39] and [45], is a distributed multiagent Java-based generalization of the sensor fusion effects (SFX) architecture [2]. It provides distributed capabilities by adding a distributed layer to SFX using the Jini package for distributed Java-based system implementation. The formulation of the distributed layer is inspired from the concept of a persona from psychology in that distributed services, up to and including individual robots, are represented by their functional characteristics to the broader distributed system as a whole. Services can be searched for based on needed functionality, rather than by name or physical location. The DFRA is the backbone of the overall heterogeneous multirobot system.

Jini is utilized for the underlying middleware layer; the Java programming language and runtime environment are utilized for implementation and execution. Seven key constraints have influenced the design of the DFRA:

- **Behavior-based and deliberative support:** Behavior-based control has historically worked well for low-level, time-sensitive control, while the deliberative approach is geared toward learning, artificial intelligence, and processes with weaker time constraints. This requirement is met by the inclusion of the SFX hybrid deliberative reactive architecture as a base.
- **Open standards:** It is important to build on a base that is open, flexible, and extensible. An important working requirement is that the software be built on open standards and on open source if possible. Java, Jini, XML, and other core technologies are common, with large user bases and active development.
- **Fault tolerant:** Both the overall system and individual modules should be reliable in the face of hardware faults, software errors, and network problems. The use of Jini as a foundation contributes to system-level fault tolerance, while the use of SFX incorporates prior work on robot fault recovery.
- **Adaptable:** The overall system is implemented in Java, thus, software portability is not an issue as long as all services correctly implement specified interfaces. However, modules need to adapt and be good "net-

work citizens" to allow the network environment as a whole to function efficiently. This may involve limiting communication and message passing to maintain sufficient bandwidth for critical services (such as live video) to function correctly.

- **Longevity:** A robot should not be taken out of service for installation of updates and other modifications. To support this, components need to be modified, administered, logged, and maintained during runtime. This is accomplished using dynamic class loading, a feature of the Java language.
- **Consistent programming model:** Implementation should abstract object locality. The same method should be able to access local or remote services without sacrificing error handling or performance. While this constraint is of primary concern for implementation, it does impact the approach taken and the conceptual model of how services are located, acquired, and used.
- **Dynamic system:** The system should be dynamic and should be able to flexibly accommodate new sensors, effectors, or other components. This implies that clients are able to discover needed services at runtime and adapt to the addition and removal of services over time. For a client in a general distributed computing environment, the salient characteristics of a service are the capabilities and attributes of the service. This is also true for robotics as clearly presented in [39].

Three key technologies address the above constraints: the SFX architecture, Java, and Jini, as briefly discussed below.

SFX Base Architecture

SFX is a managerial architecture [2] with deliberative and reactive components incorporating sensor fusion. The primary component of the reactive layer is the *behavior* that maps some sensing percept generated by a perceptual schema to a template for motor output, known as a *motor schema*. A perceptual schema processes sensory data and other percepts to generate a representation of what is sensed.

Once a percept is generated, the behavior passes the information to a motor schema that incorporates the control necessary to act on the percept.

While reactive components operate rapidly, they do not have the ability to plan or even to maintain past state. Deliberative components executing at a slower pace do have this ability, however, and many are incorporated in the SFX architecture.

Java Implementation Language

The Java programming language and runtime environment serve as base and foundation for the distributed system controlling multiple robot platforms. It has been chosen for five reasons:

- **Platform independence:** Java is an interpreted language that can be executed on any platform that runs the Java Virtual Machine (JVM).
- **Strong typing:** Java is a strongly typed language, so it is possible to specify via interfaces certain actions that a class must implement. It is possible to interact with objects of the class in a known manner. Strong typing aids in system development and with error handling during system execution.
- **Library support:** There are many available software libraries for Java providing various functionalities. Some of the most important utilized in this research are: JDOM, an XML parser; Java3D, a vector math and 3-D visualization package; and a Java-MATLAB bridge and a Java-CORBA library to communicate with robot control software.
- **Dynamic class loading:** Dynamic class loading is a critical benefit of the Java platform, especially in a distributed scenario. Dynamic class loading allows a Java program to load classes at runtime. This enables the use of classes that may not have even been written when the Java program was started. In a distributed environment, programs or services may run for extended periods of time. Robots may move around their environment and may wish to share information or code with programs on other robots. The ability to do this dynamically is vital.
- **Performance:** Since Java is a byte-compiled language, it traditionally has been considered slow. Java runs through a virtual machine that interprets the byte code stream and generates the native machine instructions to perform each operation. This interpretation step reduces performance. However, modern virtual machines include a just-in-time (JIT) compiler that compiles basic blocks of Java code to machine code the first time the block is executed. Subsequent executions will use the newly compiled code rather than reinterpret the byte code.

Jini Distributed Layer

Middleware frameworks [43] are abstractions of a distributed computing environment allowing software developers to more easily extend system infrastructures into a distributed environment. Jini is an example of a middleware framework [44], which has a goal of providing for the spontaneous networking of services—connecting any device to any other device in the network. Jini consists of a set of specifications that describe an operational model for a Jini network and how components interact within the system. The use of a standard middleware layer such as Jini has a benefit: systems built on top of the middleware layer automatically inherit the attributes of a distributed system.

There are four primary benefits to Jini that are heavily used in this approach: Jini provides protocols that enable services to dynamically adapt to the network and computing environment; Jini provides a form of naming service, called the *lookup service*, which enables advertisement of services and availability to potential clients; Jini provides a distributed event system in which a client can register interest in a service and can be notified when that service becomes available; to handle partial

failures, Jini uses a leasing mechanism that enables a client to obtain a lease on a service, and when the lease expires, the client can either stop using the service or attempt to renew the lease.

The DFRA uses modular services to implement all robot capabilities, including sensors, effectors, and behaviors. Modules are exported to a distributed runtime system as services with certain attributes and types. Services can then be searched for (using a distributed-object lookup service) based on functional attributes rather than details of actual implementation or physical location. This architecture allows a decoupling of client and server, providing an interface (proxy) to the requesting process in a modular fashion, regardless of where the requested service physically resides or how it is implemented at the local level.

Figure 1 shows a pictorial representation of the DFRA, emphasizing the distributed layers and their relationship to the base SFX architecture. The diagram is divided into three main layers. The lowest layer represents the base SFX behavior-based hybrid deliberative-reactive robot control architecture as seen on any individual robot. This layer implements all functionalities of a single robot. The middle layer (distributed resource protection) provides access guards and security protections for any services that are distributed and available for other agents in the larger multirobot system to access. The highest level (persona) provides representations and actual access to distributed services, and this is where components may be accessed based on functionality. The entire system is implemented in Java, including the SFX base as reported in [34].

Distributed services and modules are exported to a distributed runtime system as services with certain attributes and types. Services may be searched for using a distributed

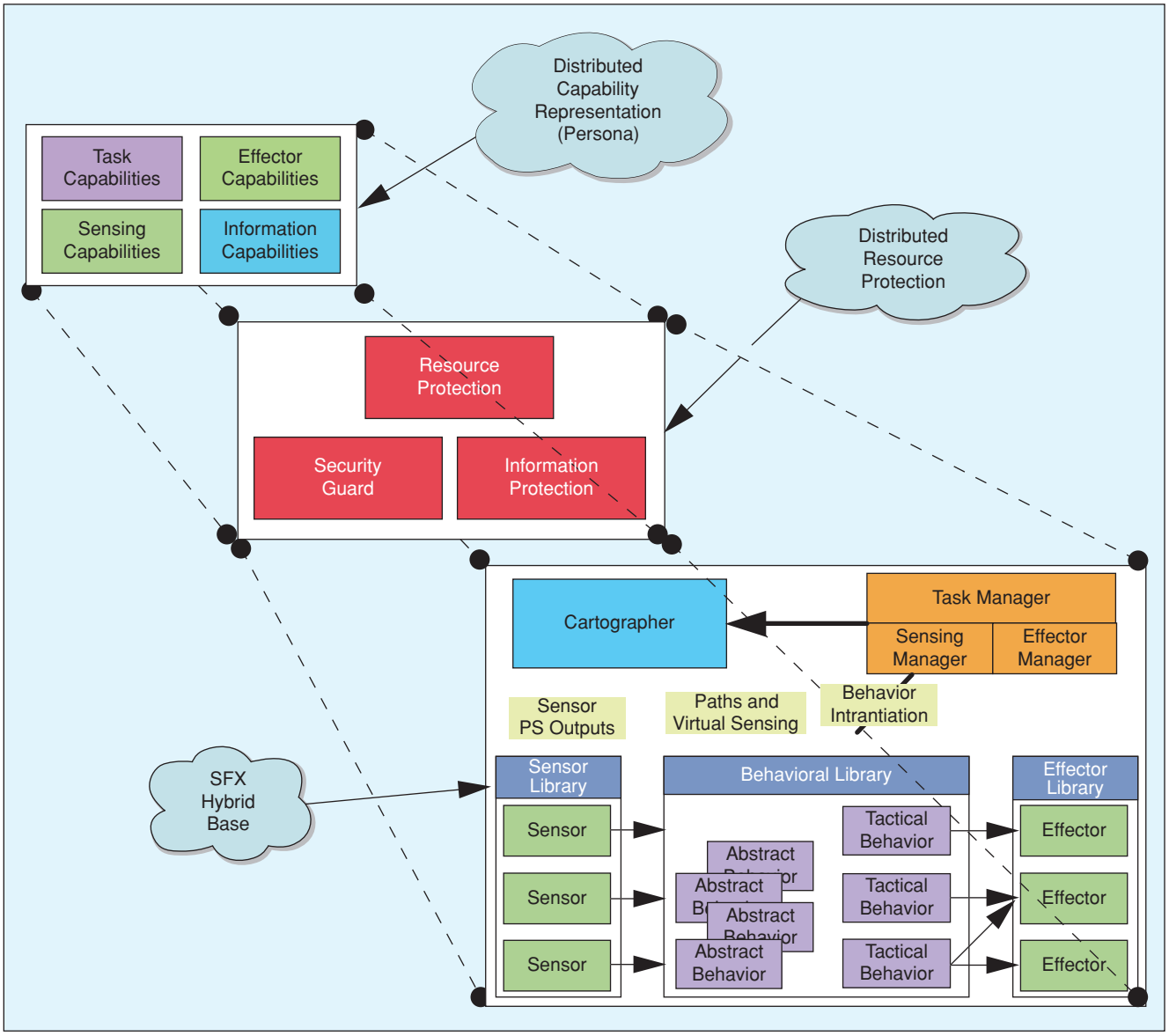

Figure 1. Distributed field robot architecture showing the relationship between the distributed system components and the base SFX architecture.

object lookup service based on functional attributes, rather than details of actual implementation or physical location. Each module is roughly divided into three components, namely, the *proxy*, *server*, and *driver*. The proxy is the representation of the service that is transported around the network, providing the ability to move code and data (it is not merely a local representation of a remote object). The server is the representation of the service that deals with the distributed system, mediating between the implementation of the service (the driver) and the remote clients. The driver is the actual implementation of the service.

The JMatLink Java class library [30] is used to integrate MATLAB into the Java-based system. JMatLink includes methods and objects that allow Java to initialize a workspace, write data members of any format to the workspace, read from the work space, and execute command line functions. The MATLAB workspace engine is accessed by delivering a formatted string to MATLAB, and its behavior is identical to that seen by a user entering command via the MATLAB workspace command line. MATLAB scripts and functions may run locally on the robots as interpreted code without the need to be compiled into stand-alone executables. Figure 2 shows the forms of support for MATLAB within the larger distributed SFX architecture. The block on the right of Figure 2 (Development Phase) represents several MATLAB-based modules in development and testing. On the left of the figure (Production Phase), a completed MATLAB-based module is shown. Note that MATLAB modules do not need to be compiled, even in the production phase.

Although the DFRA is implemented in Java/Jini, the MATLAB environment allows for mathematical control theoretic research and experimentation and for rapid prototyping of behavioral and control modules and services.

MATLAB is supported at the driver module implementation level, and it may be used as the native server implementation of a service as shown in Figure 3. The associated server and proxy handle the remote overhead and interaction with other services. Details are provided in [29].

Support Module Development

MATLAB module prototyping is demonstrated by two very simple modules: 1) a laser range data filter reducing noise and ghost readings caused by laser bouncing, variations in grass, vegetation, and other unforeseen outdoor environment conditions; this is an example of a heuristic filter relying on MATLAB's matrix and data processing power for ease of implementation and 2) an extremely simple GPS-based position detection module designed to show how the overall system integrates modules of varying sophistication and quality into a functional whole.

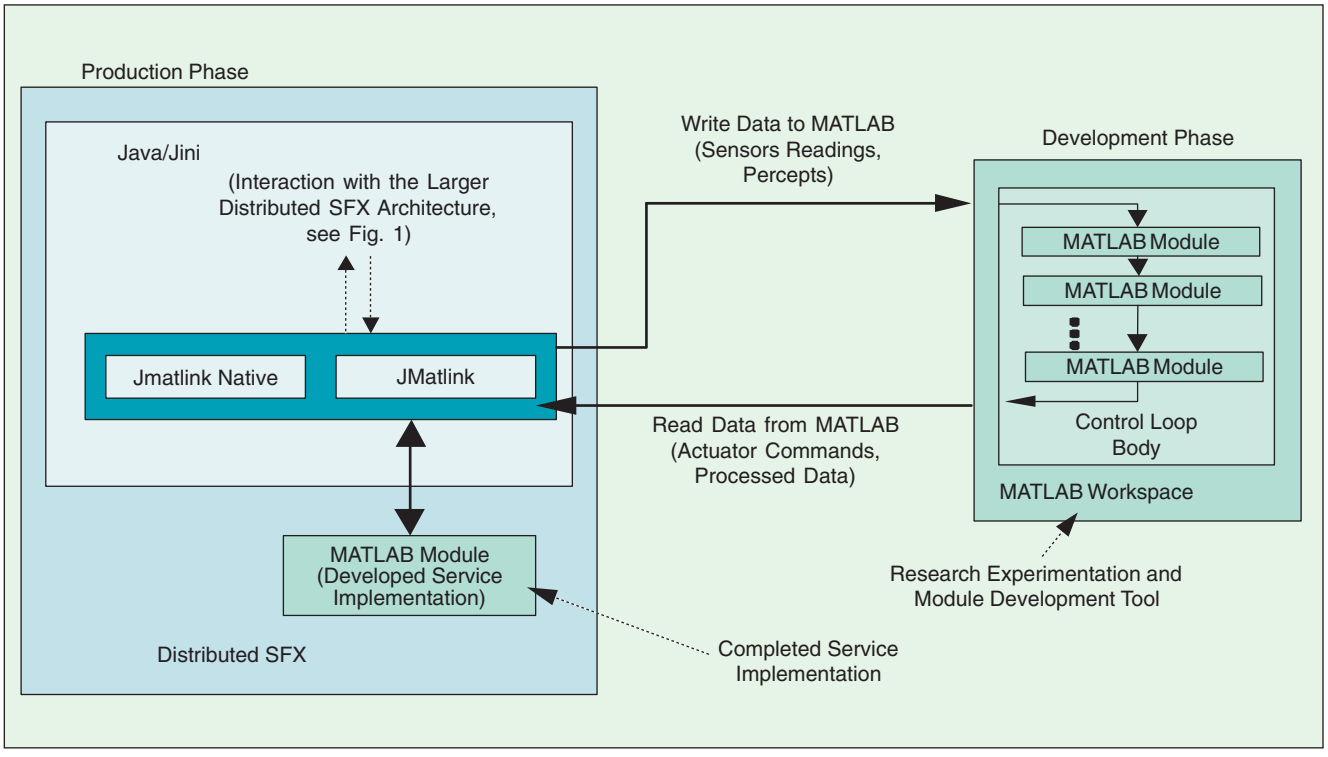

Figure 2. The relationship between MATLAB and overall DFRA.

Laser Scan Filtering

Range data needed for object avoidance is obtained from scanning planar laser units mounted on the robots. Information from the recent time history of sensor inputs is integrated to eliminate noise. The field of view of the laser scanner is 180°, centered on the robot body attached reference frame. Each consecutive point in a single scan is offset by 1° (181 total points per scan).

The laser scan filtering process integrates information from the most recent scan at time k, up to n previous scans, $k-1, k-2, \ldots, k-n$. Figure 4(a) shows several consecutive laser scans, transformed into the robot's current frame of reference. Inconsistent scan data (top right) are removed by the filter. The lower part of the scan shows a consistent object that will not be removed after filtering. Figure 4(b) shows two separate laser scans taken at time k (the most recent scan) and at time $k-n$ (a previous scan), with a and l the relative angular and linear offsets of the scans at time k and time k–n, respectively. For experimental purposes, n has been set to 3, representing a tradeoff between accuracy and response time for rotation measurements.

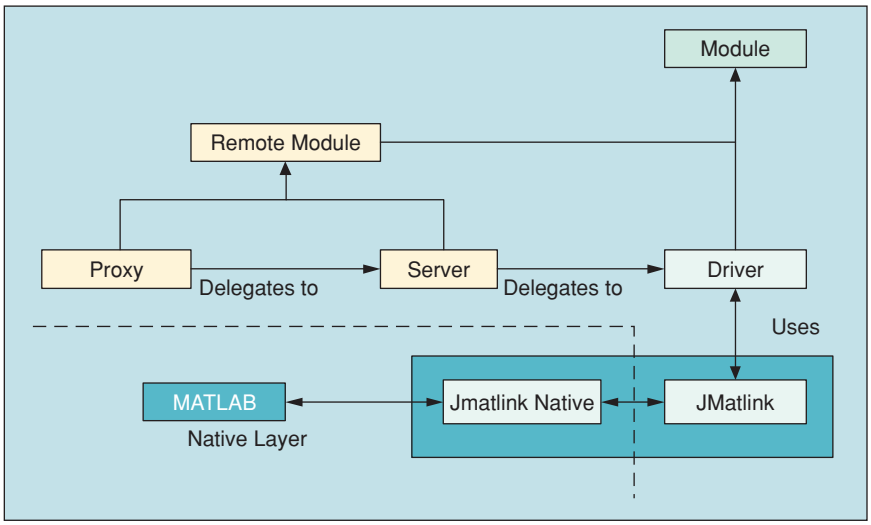

Figure 3. A horizontal hierarchy with MATLAB as the driver implementation module.

Each laser scan is represented by

$$\mathbf{L} = \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix}, \tag{1}$$

where $\mathrm{x} = [x_1, x_2, \ldots, x_I]$, $\mathrm{y} = [y_1, y_2, \ldots, y_I]$, and $I = 181$ is the number of elements per scan. Each column of L represents a consecutive (x, y) coordinate pair moving from left to right across the 180° sweep of the laser scan.

Each of the (previous) past $n \in \{1, N\}$ scans is transformed into the reference frame of the most recent scan $\mathbf{L}(k)$. In order to account for the fixed orientation of the laser to the robot frame of reference in previous scans, coordinate pairs are shifted (left or right) by a number of elements equal to the number of degrees of rotation between the robot's current position and its position associated with that previous scan [i.e., by a from Figure 4(b)] Then, using a standard rotation and translation transformation $\mathbf{T}$, passed scans are transformed into the current scan's frame of reference

$$\mathbf{L_T}(k-n) = \mathbf{L}(k-n)\mathbf{T}_{\mathbf{r}(k),\mathbf{r}(k-n)}. \tag{2}$$

$\mathbf{T}$ is given in terms of linear and rotational offsets, and it is recalculated for each previous scan as follows

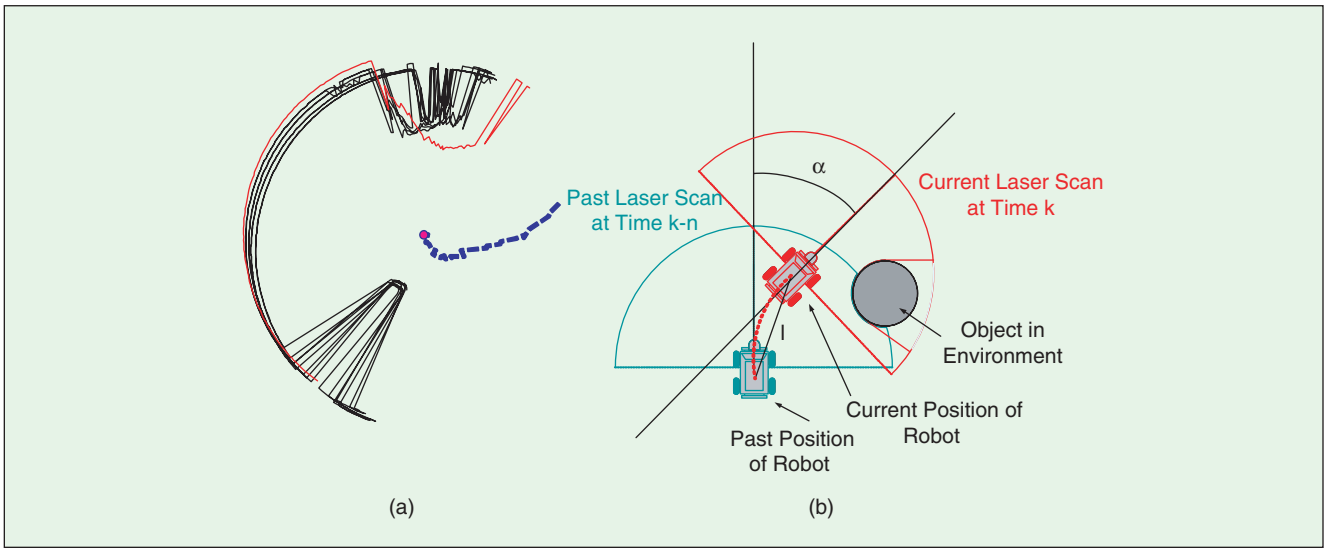

Figure 4. (a) Consecutive laser scans collected during a robot run in the field. (b) Diagram depicting laser scans taken at time k and at time k–n.

$$\mathbf{T}_{\mathbf{r}(k),\mathbf{r}(k-n)} = \begin{bmatrix} c(\alpha/2) & -s(\alpha/2) & 0 \\ s(\alpha/2) & c(\alpha/2) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & l \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} c(\alpha/2) & -s(\alpha/2) & 0 \\ s(\alpha/2) & c(\alpha/2) & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (3)$$

Each shifted and transformed $\mathbf{L}$ is then converted into a vector $\mathbf{r}$ in polar form with elements ri, $i \in \{1, 181\}$ representing the Euclidian distance from the origin of the robot body attached reference frame, with angles implicitly defined:

$$\mathbf{r} = [\, r_1 \quad r_2 \quad \cdots \quad r_{181} \,], \quad \text{where} \quad r_i = \sqrt{x_i^2 + y_i^2}. \quad (4)$$

The matrix $\mathbf{R}$ shifted, transformed, and converted to polar scans represents the last n laser scans with all range readings transformed into the current robot reference frame

$$\mathbf{R} = \begin{bmatrix} \mathbf{r}(k) \\ \mathbf{r}_\mathbf{T}(k-1) \\ \vdots \\ \mathbf{r}_\mathbf{T}(k-n) \end{bmatrix}. \quad (5)$$

Hence, objects detected in multiple current and past scans appear as range readings of similar values in multiple rows of $\mathbf{R}$.

A simple heuristic and statistical filter is employed using the variance, the mean value, and the maximum range readings of each of the columns of $\mathbf{R}$. These are calculated respectively as

$$\begin{aligned} \mathbf{r}_{\text{var}} &= \text{var}(\mathbf{R}) \\ &= [\text{var}(r_1(k), r_1(k-1), \cdots, r_1(k-n)) \cdots \\ &\qquad \text{var}(r_{181}(k), r_{181}(k-1), \cdots, r_{181}(k-n))] \quad (6) \\ \mathbf{r}_\mu &= [\mu(r_1(k), r_1(k-1), \cdots, r_1(k-n)) \cdots \\ &\qquad \mu(r_{181}(k), r_{181}(k-1), \cdots, r_{181}(k-n))] \quad (7) \\ \mathbf{r}_{\max} &= [\max(r_1(k), r_1(k-1), \cdots, r_1(k-n)) \cdots \\ &\qquad \max(r_{181}(k), r_{181}(k-1), \cdots, r_{181}(k-n))]. \end{aligned} \quad (8)$$

Angles of all range readings are implicit in the order in which the readings appear in the rows of $\mathbf{R}$, being consistent for all rows of $\mathbf{R}$ due to initial shifting. The final filtered laser vector is then given by

$$\mathbf{r}_\Phi = [\, r_1 \quad r_2 \quad \cdots \quad r_{181} \,] \text{ where } r_i = \begin{cases} r_{i,\mu} & \text{if } r_{i,\text{var}} < \nu \\ r_{i,\max} & \text{otherwise,} \end{cases} \quad (9)$$

where $\mathbf{r}_\Phi$ is the final filtered set of range readings spanning the forward field of the robot in its current position, and $r_{i,\mu}$, $r_{i,\text{var}}$, and $r_{i,\max}$ are the ith elements of $\mathbf{r}_\Phi$, $\mathbf{r}_{\text{var}}$, and $\mathbf{r}_{\max}$, respectively, and ν is an appropriately defined threshold value.

The filter essentially takes the average of the ranges (in a particular direction) if they agree over n previous scans. If there is significant disagreement, then the most optimistic (or furthest) range is taken. The reason for this optimistic default is that the laser range finders (unlike sonar and even IR) very rarely report a real object to be further than it actually is

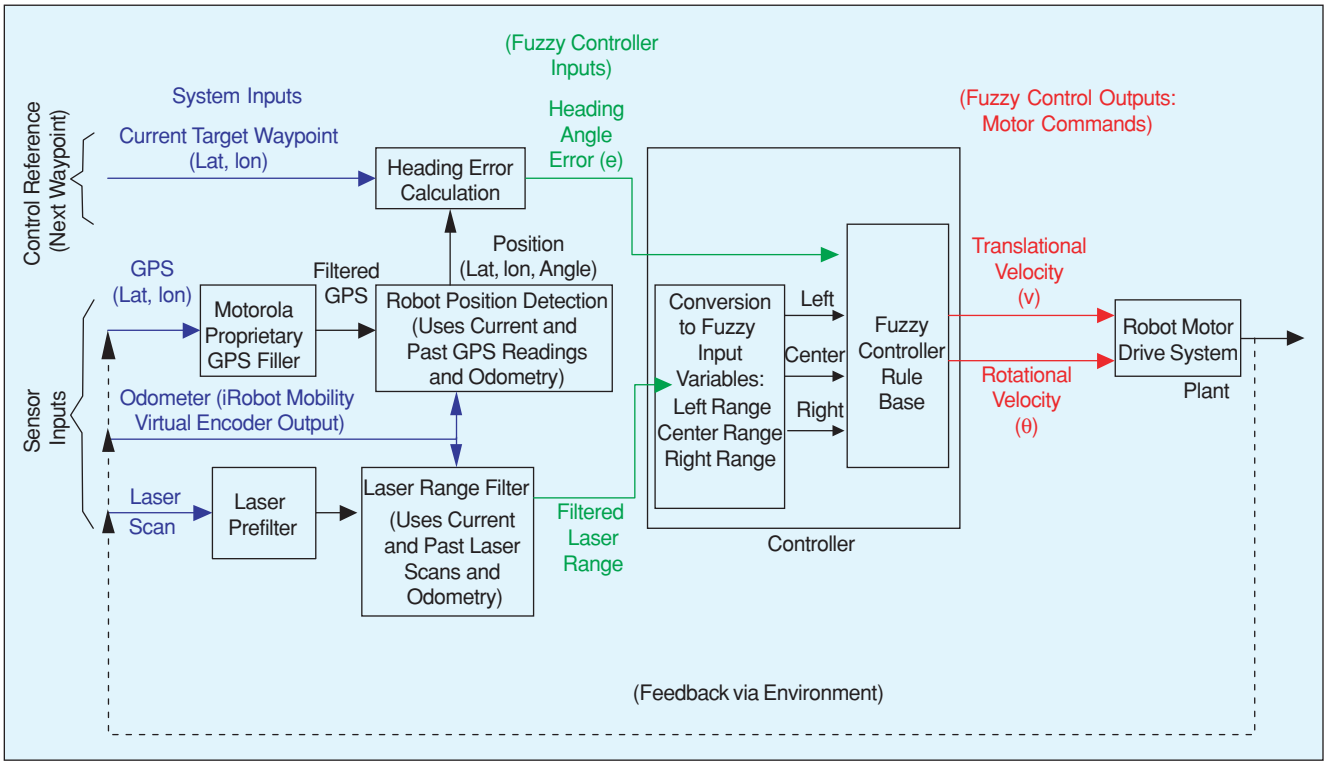

Figure 5. Control system shown as a collection of interrelated modules.

(exceptions are glass and certain reflective surfaces); this has been experimentally observed. Data from scans prior to current time k outside the robot's current forward facing field of view (after shifting and transformation) are discarded in the above calculations.

Position Detection

Pose (position) detection has been based on current and passed GPS readings. Pose is represented by a triplet, (x, y, θ).

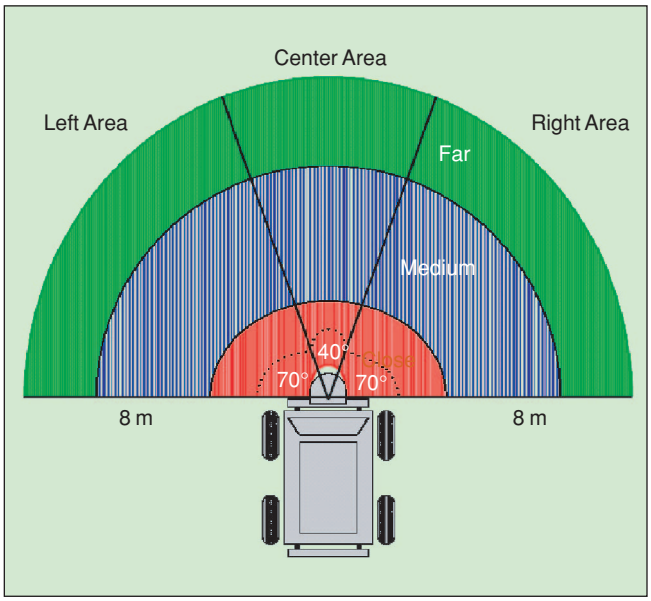

Figure 6. The laser scan area's radial sectors divided into close, medium, and far areas.

The location of the robot (x, y) is taken directly from the current GPS latitude and longitude readings (filtered using a proprietary filter supplied with the Motorola unit). Heading θ is calculated by determining the angle made between a line passing through the current robot position and a previous position and due east. In this work, the offset between points for heading calculation are set to 10 points, corresponding to a distance of approximately 2 m when the robot was traveling at an average speed of 0.5 m/s. This simple method has provided adequate positioning data, and the average GPS error measured during all experiments has been found to be less than 1 m when the robots traveled over a 10-m path with known absolute ground position.

This coarse pose detection method is not state of the art; it is used to demonstrate how the overall DFRA-MATLAB

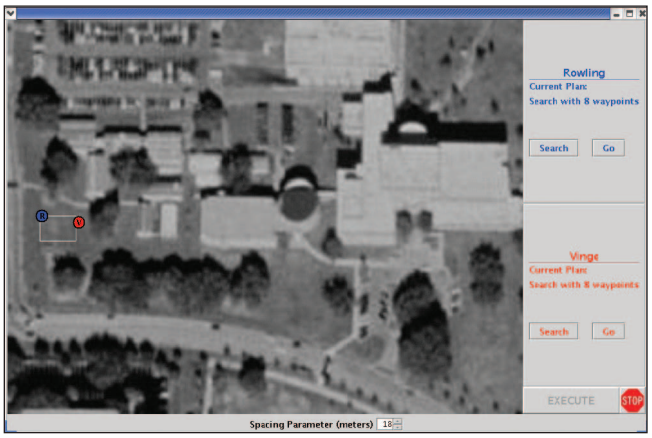

Figure 7. Graphical user interface.

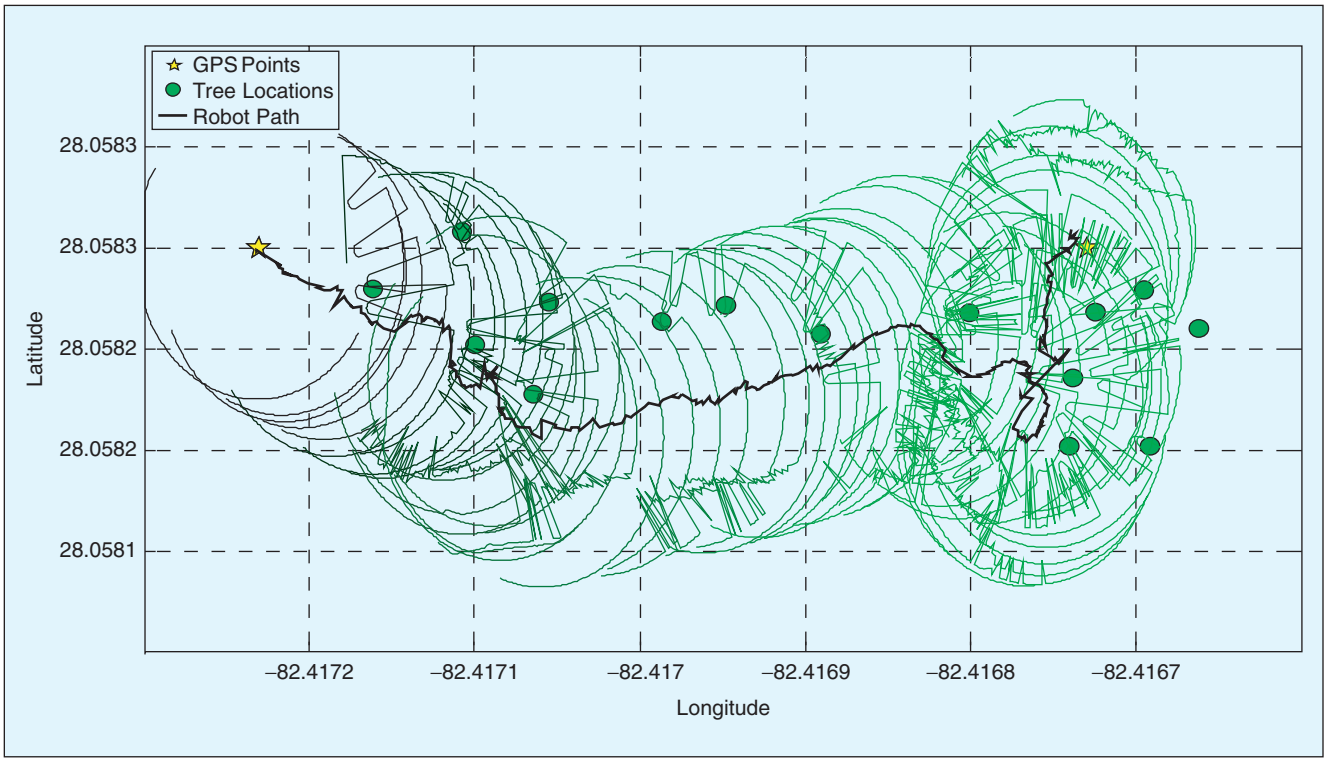

Figure 8. Robot path: laser scans are shown.

framework may be used with components of varying degrees of refinement.

Fuzzy Logic Controller

The multisensor control system is shown in Figure 5. It consists of four modules: the laser range filter, position detection, heading error calculation, and the actual fuzzy logic robot controller. An additional module logs all sensor data, controller outputs, control loop delay, and various other system data time stamped and synchronized so that experiments performed can be reconstructed and analyzed at a fine level of detail.

The control system receives as inputs laser, odometer, and GPS data as well as a control reference input (next waypoint or goal point). It outputs actuator commands in terms of robot rotational and translational velocities.

The fuzzy logic controller is implemented as a Mamdani-type controller similar to previous work [5], [7]. The fuzzy logic controller rule base includes the fuzzy rules responsible for vehicle control. The inference engine activates and applies relevant rules to control the vehicle. The fuzzification module converts controller inputs into information used by the inference engine. The defuzzification module converts the output of the inference engine into actual outputs for the vehicle drive system.

For formulation of the filtered laser data into fuzzy linguistic variables to be used as input into the fuzzy controllers, the laser scan area is divided in three radial sectors labeled as *Left Area*, *Center Area*, *Right Area*, denoted by W_i $i = 1, 2, 3$, each one including further division in *Close, Medium*, and *Far* regions as shown in Figure 6. Laser effective range is experimentally verified to be about 8 m (25 ft). The left and right areas have a width of 70° each and the center area of 40°.

The fuzzy controller input from the filtered laser range block consists of a three-value vector with components related to the distance of the closest object in the left sector of the scan, in the center sector, and in the right sector, respectively. This information is used to calculate three collision possibilities *left*, *center*, and *right*, reflecting potential static/dynamic obstacles in the robot field of view, similar to the approach followed in [5], [7] but for outdoor environments. The fourth input to the fuzzy logic controller is the robot's heading error calculated from the robot's current heading and the desired heading.

Implementation-wise, each of the three aggregate range inputs includes three trapezoidal membership functions, namely, *close*, *medium,* and *far*. The input linguistic variables are denoted as *left distance*, *right distance,* and *center distance*, corresponding to the left area, right area, and center area sectors. The *heading error* input uses four trapezoidal membership functions and one triangular membership function. Chosen membership functions for the input variables are empirically derived based on extensive tests and experiments as reported in [29].

The value of each distance input variable d_i (corresponding to left area, center area, right area) is fuzzified and expressed by the fuzzy sets C_i, MD_i, A_i referring to *close*, *medium*, and *far* as shown in Figure 6. The range of the membership functions for each d_i is between 0–8 m. The value of the input variable *heading error*, *he*, is fuzzified and expressed by the fuzzy sets *FL*, *L*, *AH*, *R*, *FR*, referring to *far left*, *left*, *ahead*, *right*, and *far right*, respectively. The range of the membership functions for the *heading error* is between −180° and 180°.

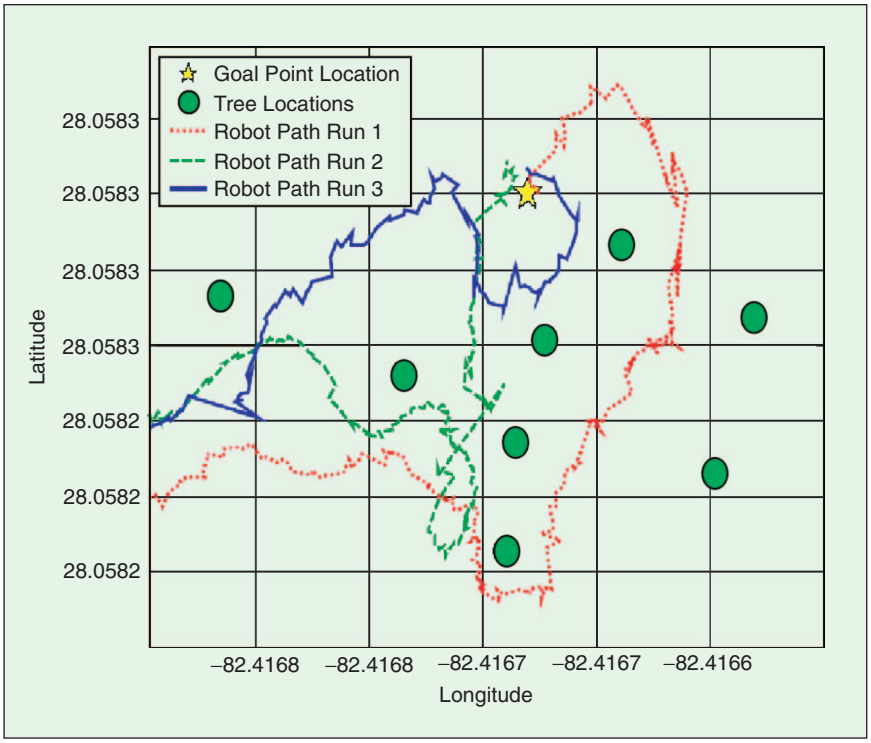

Figure 9. *Three different paths for the same scenario.*

Figure 10. *A sequence of images showing the robot consecutively avoiding two trees.*

The fuzzy logic controller has two output variables 1) *translational velocity* (*tr*) implemented with two trapezoidal membership functions and one triangular membership function and 2) *rotational velocity* (*rv*) implemented with four trapezoidal membership functions and one triangular membership function.

The value of the output variable *tr* is expressed by the fuzzy sets *ST*, *SL*, *F*, referring to *stop*, *slow*, and *fast*. The value of the output variable *rv* is expressed by the fuzzy sets *HRR*, *RR*, *AHR*, *LR*, and *HLR*, referring to *hard right*, *right*, *ahead*, *left*, and *hard left*.

The output commands are normalized in a scale from 0–1 for the translational velocity, where 0 corresponds to complete stop and 1 to maximum speed. Rotational velocity output commands are normalized from −1 to 1, where −1 corresponds to a right turn with maximum angular velocity and 1 to a left turn with maximum angular velocity. Each fuzzy rule j is expressed as: IF d_1 is D_{j1} AND d_2 is D_{j2} AND d_3 is D_{j3} AND he is HE_j THEN tr is TR_j AND rv is RV_j; for $j = 1, \ldots,$ number of rules.

D_{ji}, is the fuzzy set for d_i in the jth rule, which takes the linguistic value of C_i, MD_i, A_i. HE_j is the fuzzy set for the *he,* which takes the linguistic values *FL*, *L*, *AH*, *R*, *FR*. TR_j and RV_j are the fuzzy sets for *tr* and *rv*, respectively. The generic mathematical expression of the jth navigation rule is given by

$$\mu_{R^{(j)}}(d_i, he, tr, r) = \min[\mu_{D_i^j}(d_i), \mu_{HE^{(j)}}(he), \mu_{TR^{(j)}}(tr), \mu_{RV^{(j)}}(rv)]. \quad (10)$$

The overall navigation output is given by the max-min composition and in particular

$$\mu_N^*(tr, vr) = \max_{d_i, he} \min \left[\mu_{AND}^*(d_i, he), \mu_R(d_i, he, tr, rv)\right], \quad (11)$$

where

$$\mu_R(d_i, he, tr, rv) = \bigcup_{j=1}^{J} \mu_{R^{(j)}}(d_i, he, tr, rv).$$

The navigation action dictates change in robot speed and/or steering correction, and it results from the defuzzification formula, which calculates the center of the area covered by the membership function computed from (11).

Experimental Results

Experiments were performed in outdoor environments using two ATRV-Jr mobile robot platforms. There was access to all Mobility functions (which may be called if needed), but Mobility itself was not used as a support software environment. Reported results concentrate in fuzzy-logic-based navigation and collision avoidance; more complicated case studies

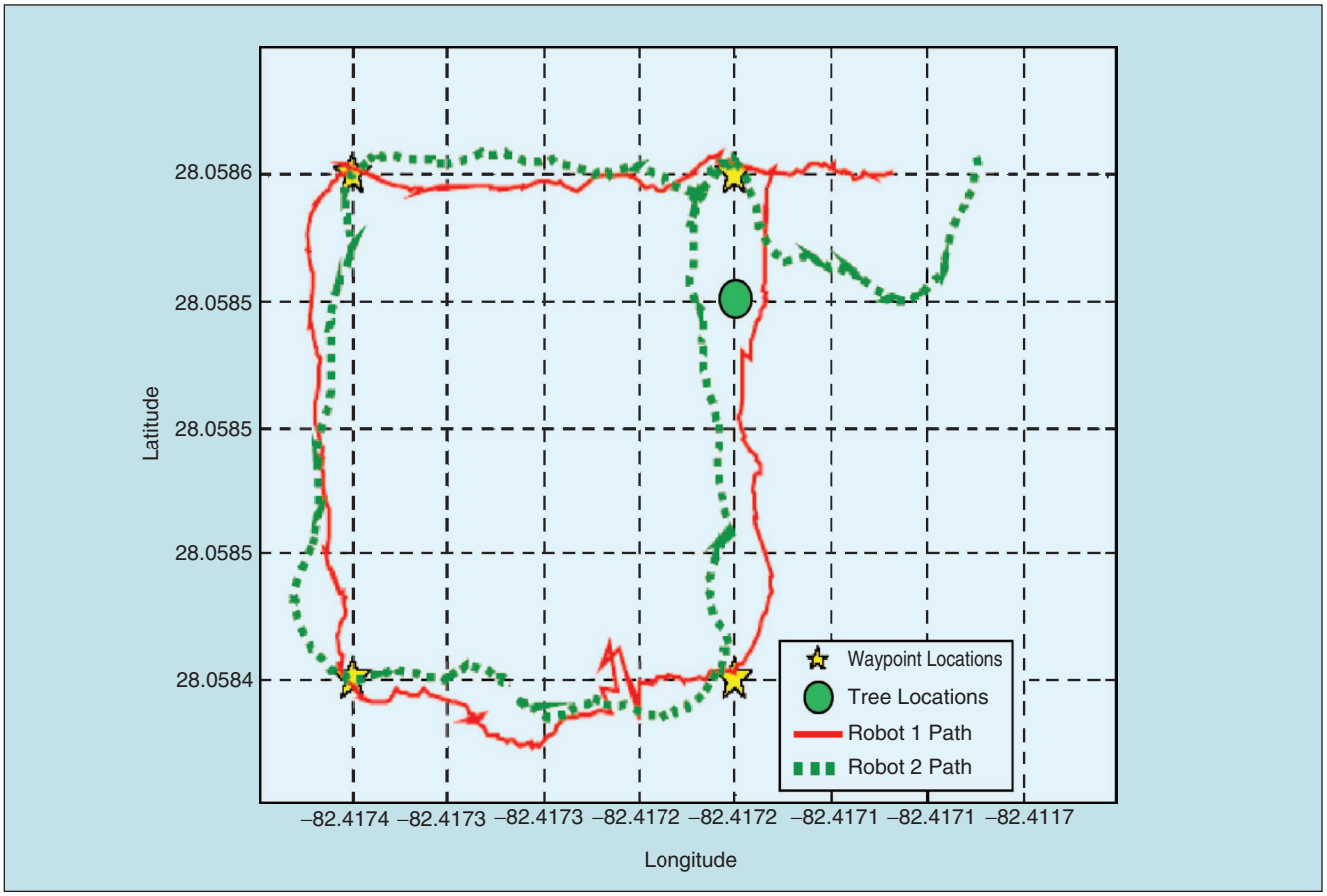

Figure 11. *The paths followed by the two robots in the second experiment.*

have been reported in [45]. A brief discussion is also offered for both odometer and GPS error quantification.

Discussion of Odometer and GPS Error Quantification

To quantify, somehow, odometer and GPS errors, experiments were conducted where ATRVs followed three predetermined test patterns: forward and backward motion along a 15-m straight line; tracing a 10-m square; and tracing a circle with a radius of 25 m. A total of six tests per pattern over a three-day period were conducted to quantify raw GPS, filtered GPS (using the Motorola-supplied filter within the GPS unit), and odometer errors. A full control loop cycle required 0.2 s, thus, data were collected at a rate of 5 Hz.

It was observed that in the rectangular and circular tests, odometer and GPS positions deviated by approximately 2 m and 6 m on average, respectively.

Considering all tests for the forward and backward robot movement, the cumulative odometer error was found to be 0.4% per meter traveled, while the average GPS position error was found to be 0.91 m with an error standard deviation of 0.52 m.

The path generation for the square pattern was generated by a timed sequence of forward commands followed by a rotation command calibrated to produce a 90° turn. Slight variations in turning times and loop delays resulted in progressively incrementing position error, even as measured by odometry. The circle tests, on the other hand, were generated by applying the same actuator arc command repeatedly. More details are offered in [29].

Dynamic Fuzzy-Logic-Based Control in Outdoor Environments

Experiments were performed in an outdoor (somewhat uneven terrain) environment with dirt, grass, tree, and some vegetation. The first set of experiments required that the robots travel through predefined waypoints while avoiding static and dynamic obstacles. The second set required that robots follow sets of waypoints that can be changed dynamically by a human operator while the robots are moving. Additional experiments included raster scans with two robots starting from different initial positions, avoiding each other as well as other obstacles found in their paths.

To facilitate further robot deployment, a graphical user interface (GUI) was designed, shown in Figure 7, allowing human operators to monitor robot movement, to modify dynamically their waypoints or current goal positions, or to define areas to perform a raster search (select a desired area to scan specifying a lane spacing parameter). Although robots receive final goal positions and waypoint sets from the GUI, all control processing is performed locally on the robots. Robot controllers may revert to locally stored goal locations or waypoint lists if the GUI is not in operation, if it is not required for a particular experiment, or if communication between the remote GUI and the robots is cut off.

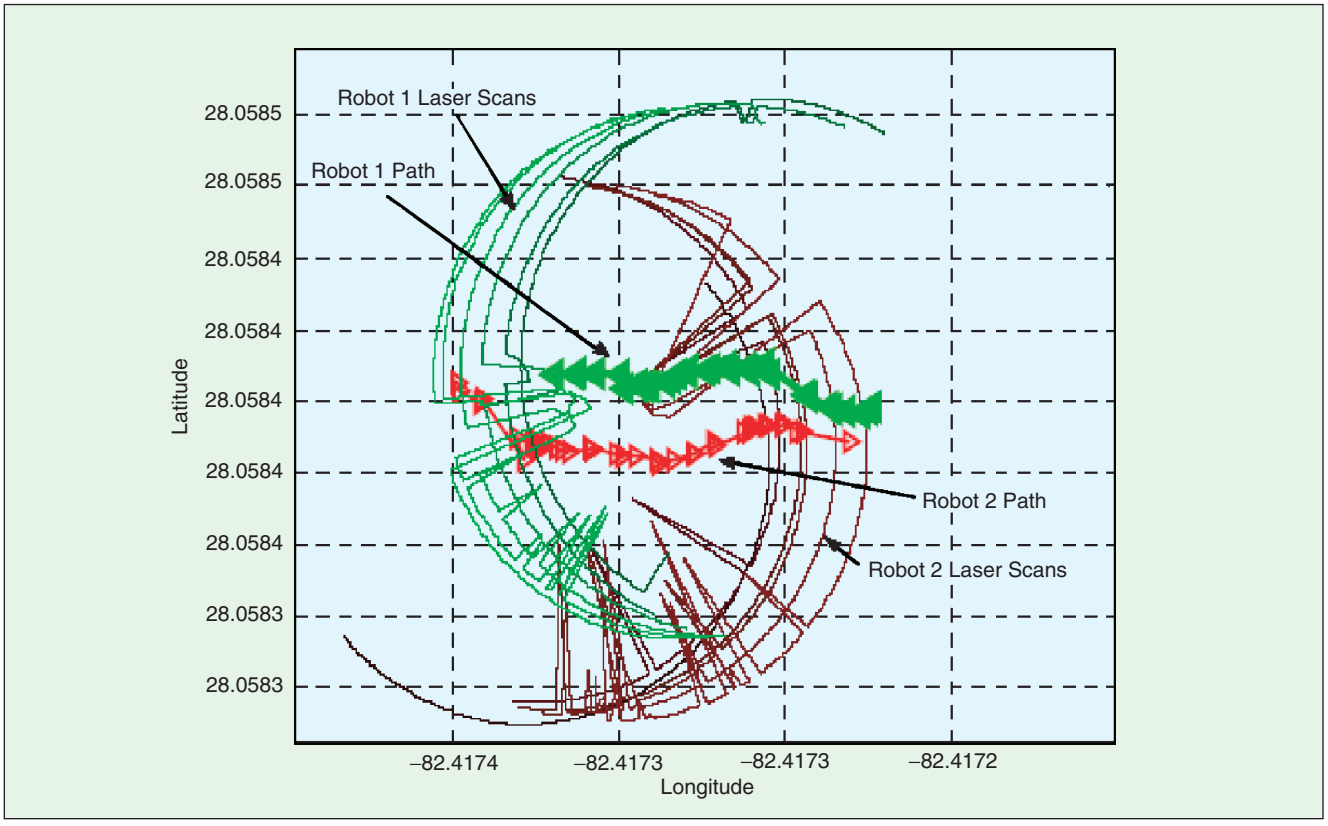

Figure 12. *Robots avoiding each other.*

Experiment 1

The first experiment demonstrates goal point following in an environment with many unknown obstacles (trees). The robot is given an initial position on one side of a large group of trees and a final goal point location on the other side of the group of trees. Figure 8 shows one full path traveled through the tree-covered area from the initial point to the final point; periodic laser scans are shown over the course of the robot's path. This experiment was repeated several times, and Figure 9 shows the different paths followed by the robot for three repetitions of the same experiment. Path differences were caused by variations in the input readings produced by differences in GPS reception and by subtle and cumulative differences in laser reading and odometer readings. However, in all cases, the robot moves toward its final goal GPS point. In each of the experiments, robots were able to ultimately find the distant goal positions without colliding with any of the trees. Figure 10 presents a sequence of photos of the robot navigating among the trees (sequence from upper left to lower right).

Experiment 2

Experiments have been performed with two robots operating simultaneously. Robots travel through a set of goal points delineating a box shape. The robots start from different initial positions and travel to each of the goal points defining the box. One robot moves clockwise, the other counterclockwise. The paths followed are shown in Figure 11; similar results have been obtained repeating the same experiment. The robots negotiate a static object (a tree) and a dynamic object (the other robot) as they traverse the set of goal GPS points. While the robots were moving, they crossed each other's path and avoided one another as shown in Figure 12 and visualized with a sequence of images in Figure 13.

Experiment 3

A raster search was also performed using two robots. Robots start from different positions in the field and move in opposite directions while performing the raster search. The trajectories followed by the two robots are presented in Figure 14. The experiment has been repeated several times with similar results. For archive videos, see http://www.csee.usf.edu/~aanelson/Robot_Movies.html.

Discussion and Conclusions

Discussion

It is essential to clarify that while individual modules described in this article appear to be relatively simple, *they*

Figure 13. *Images of two robots moving towards each other and avoiding one another.*

form the basic building blocks needed to execute more complex tasks. Such a task, presented in [45], is a complex outdoor-simulated demining task where sensors, effectors, schemas, and behaviors implemented in each robot were managed through the DFRA.

The task involved two unmanned ground vehicles (UGVs) and an unmanned vertical take-off and landing (VTOL) vehicle. Each UGV was paired to an operator control unit; a mission control unit coordinated the entire operation. The three robots and all control interfaces communicated through a wireless LAN that used the dynamic discovery capabilities of the DFRA to locate and interact with the services on other machines. The scenario was as follows:

- The mission controller directed one robot to perform a raster search of a part of the demonstration field, looking for simulated mines;
- The VTOL vehicle, under remote teleoperation but using onboard vision processing, began searching the field, too.
- When a potential mine was detected by the VTOL, a recruitment agent, implemented within DFRA, sent a request for help. Following the recruitment protocol described in [46], the second, idle robot answered the call for help and began a local search of the area containing the potential mine. This process operated independent of operator interaction, solely at the direction of the software agents operating on behalf of each robot.

The DFRA successfully integrated operator graphical interfaces, high-level deliberative and low-level reactive robot services. Each ground robot operated under the direction of both the DFRA and the MATLAB-based fuzzy logic controller. High-level agents interacted to provide task-level commands. The control theoretic lower level was then responsible for the execution and successful completion of the navigation goals.

This demonstration validated four key aspects of the system: the architecture can be extended and implemented on

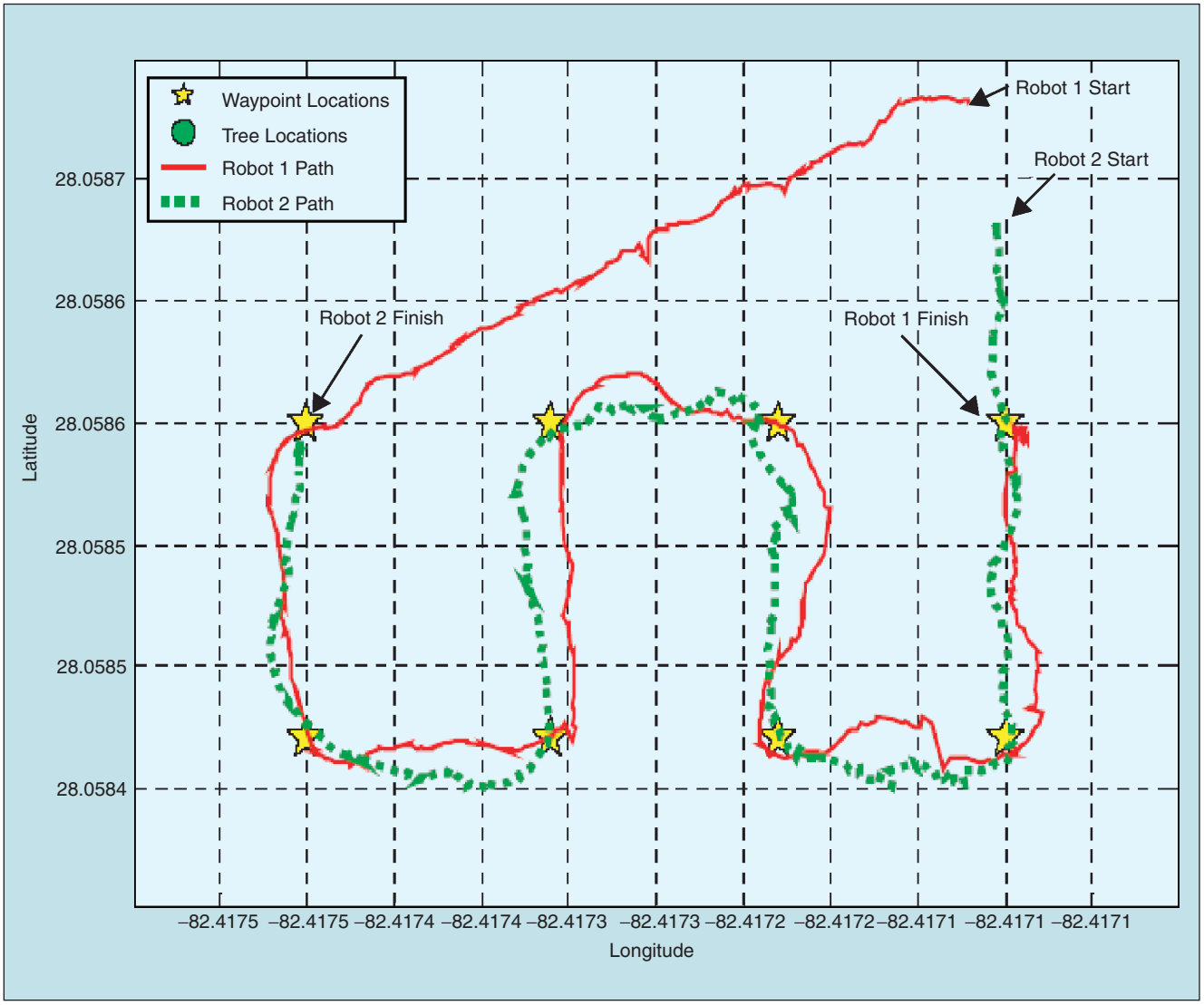

Figure 14. *A raster search performed by the two robots.*

real hardware; reactive components in general and the MATLAB-based control theoretic components in particular can operate in a real-world situation; deliberative agents can interact with and influence the operation of multiple robots; and the distributed system can be used effectively for a complex task.

It is also true that due to "timing issues," controllers must be designed to accommodate an acceptable variation in control delay. For the performed experiments, the system controller loop delay was about 0.2 s (5 Hz), thus, the fuzzy rule set was designed to function in a control loop window of between 10–1 Hz.

Conclusions

This article presented a DFRA and its integration with MATLAB that is capable of supporting simple and complex functionality of heterogeneous teams of robot systems. This architecture was used to demonstrate multisensor mobile robot fuzzy-logic-based navigation in outdoor environments.

The main contribution of the article is the overall architecture that serves as the backbone for any module design and implementation. A second contribution is the fuzzy logic controllers that are extensions of previously reported ones in [5]– [7]. The deviation from the previous design is in using a totally different sensor suite (lasers, odometers, and GPS) for outdoor navigation (versus sonar sensor-based indoor navigation), different area division for scanning and simplicity of implementation. Finally, the MATLAB control theoretic level as integrated with the rest of the system architecture may serve as a "standard platform" for distributed heterogeneous systems.

Acknowledgments

The authors wish to thank wholeheartedly Dr. Andrew Nelson for his contribution to this project in general and this article in particular. This work has been partially supported by a grant from the Office of Naval Research, N00014-03-01-786 (with USF 2132-033-LO). L. Doitsidis is also supported partially by "IRAKLITOS" fellowships for research from the Technical University of Crete, EPEAEK II—88727.

Keywords

Mobile robots, fuzzy logic, autonomous navigation, collision avoidance.

References

[1] A. Saffiotti, "Handling uncertainty in control of autonomous robots," in *Uncertainty in Information Systems*, A. Hunter and S. Parsons, Eds. New York: Springer, 1998, LNAI 1455, pp. 198–224.

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

[3] R.C. Arkin, *Behavior-Based Robotics*. Cambridge, MA: MIT Press, 1998.

[4] T. Balch and L.E. Parker, Eds. *Robot Teams, from Diversity to Polymorphism*, A.K. Press, 2002.

[5] L. Doitsidis, K.P. Valavanis, and N.C. Tsourveloudis, "Fuzzy logic based autonomous skid steering vehicle navigation," in *Proc. IEEE Int. Conf. Robotics Automat.*, Washington, DC, 2002, pp. 2171–2177.

[6] K.P. Valavanis, T. Hebert, R. Kolluru, N.C. Tsourveloudis, "Mobile robot navigation in 2-dynamic environments using electrostatic potential fields," *IEEE Trans. Syst., Man Cybern.*, vol. 30, no. 2, pt. A, pp. 187–196, 2000.

[7] N.C. Tsourveloudis, K.P. Valavanis, and T. Hebert, "Autonomous vehicle navigation utilizing electrostatic potential fields and fuzzy logic," *IEEE Trans. Robot. Automat.*, vol. 17, no. 4, pp. 490–497, 2001.

[8] F. Abdessemed, K. Benmahammed, and E. Monacelli, "A fuzzy-based reactive controller for a non-holonomic mobile robot," *Robot. Auton. Syst.*, vol. 47, no. 1, pp. 31–46, 2004.

[9] T.L. Lee and C.J. Wu, "Fuzzy motion planning of mobile robots in unknown environments," *J. Intell. Robot. Syst.*, vol. 37, no. 2, pp. 177–191, 2003.

[10] E. Aguire and A. Gonzalez, "Fuzzy behaviors for mobile robot navigation: Design, coordination and fusion," *Int. J. Approx. Reas.*, vol. 25, no. 3, pp. 225–289, 2000.

[11] S.C. Goodridge and R.C. Luo, "Fuzzy behavior fusion for reactive control of an autonomous mobile robot: MARGE," in *Proc. IEEE Int. Conf. Robotics Automation*, San Diego, CA, 1997, pp. 1622–1627.

[12] S. Ishikawa, "A method of indoor mobile robot navigation by using fuzzy control," in *Proc. IEEE/RSJ Int. Workshop Intell. Robots Syst. IROS,* Osaka, Japan, 1991, pp. 1013–1018.

[13] W. Li, "Fuzzy logic-based 'perception-action' behavior control of a mobile robot in uncertain environment," in *Proc. 3rd IEEE Conf. Fuzzy Systems*, 1994, vol. 3, pp. 1626–1631.

[14] F.G. Pin and Y. Watanabe, "Navigation of mobile robots using fuzzy logic behaviorist approach and custom design fuzzy inference boards," *Robotica*, vol. 12, pp. 491–503, 1994.

[15] A. Saffiotti, "Fuzzy logic in autonomous robotics: Behavior coordination," in *Proc. IEEE Int. Conf. Fuzzy Systems*, 1997, pp. 573–578.

[16] A. Saffiotti, K. Konolige, and H.E. Ruspini, "A multivariable logic approach to integrating and planning and control," *Artif. Intell.*, vol. 76, no. 1–2, pp. 481–526, 1995.

[17] E. Tunstel, "Mobile robots autonomy via hierarchical fuzzy behavior," in *Proc. 6th Int. Sym. Robotics Manufacturing*, 1996, pp. 837–842.

[18] H. Seraji and A. Howard, "Behavior-based robot navigation on challenging terrain: A fuzzy logic approach," *IEEE Trans. Robot. Automat.*, vol. 18, no. 3, pp. 308–321, 2002.

[19] H. Hagras, V. Callaghan, and M. Colley, "Outdoor mobile robot learning and adaptation," *IEEE Robot. Automat. Mag.*, vol. 8, no. 3, pp. 53–69, 2001.

[20] A.D. Mali, "On the behavior-based architectures of autonomous agency," *IEEE Trans. Syst., Man Cybern.*, vol. 32, no. 3, pp. 231–242, Aug. 2002.

[21] J. Borenstein and L. Feng, "Measurement and correction of systematic odometry errors in mobile robots," *IEEE Trans. Robot. Automat.*, vol. 12, no. 6, , pp. 869–880, Dec. 1996.

[22] L. Ojeda and J. Borenstein, "Reduction of odometry errors in over-constrained mobile robots," in *Proc. UGV Technology Conf. 2003 SPIE AeroSense Symp.*, Orlando, FL, Apr. 21–25, 2003.

[23] S. Panzieri, F. Pascucci, and G. Ulivi, "An outdoor navigation system using GPS and inertial platform," *IEEE/ASME Trans. Mechatron.*, vol. 7, no. 2, pp. 134–142, June 2002.

[24] K. Ohno, T. Tsubouchi, and B. Shigematsu, "Outdoor navigation of a mobile robot between buildings based on DGPS and odometry data fusion," in *Proc. IEEE Int. Conf. Robotics Automat.*, Taipei, 2003, pp. 1978–1984.

[25] R. Thrapp, C. Westbrook, and D. Subramanian, "Robust localization algorithms for an autonomous campus tour guide," in *Proc. IEEE Int. Conf. Robotics Automat.*, 2001, vol. 2, pp. 2065–2071.

[26] T.W. Vaneck, "Fuzzy guidance controller for an autonomous boat," *IEEE Control Syst. Mag.*, vol. 17, no. 2, pp. 43–51, Apr. 1997.

[27] M.H. Bruch, G.A. Gilbreath, J.W. Muelhauser, and J.Q. Lum, "Accurate waypoint navigation using non-differential GPS," in *Proc. AUVSI Unmanned Syst. 2002*, Lake Buena Vista, FL, July 9–11, 2002.

[28] M.T. Long, R.R. Murphy, and L.E. Parker, "Distributed multi-agent diagnosis and recovery from sensor failures," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2003, vol. 3, pp. 2506–2513.

[29] A.L. Nelson, L. Doitsidis, M.T. Long, K.P. Valavanis, and R.R. Murphy, "Incorporation of MATLAB into a distributed behavioral robotics architecture," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2004, pp. 2028–2035.

[30] S. Müller and H. Waller, "Efficient integration of real-time hardware and Web based services into MATLAB," in *Proc. ESS'99 11th European Simulation Symp. Exhibition,* ESS'99, Erlangen-Nuremberg, Oct. 26-28, 1999.

[31] L.E. Parker, "ALLIANCE: An architecture for fault tolerant multirobot cooperation," *IEEE Trans. Robot. Automat.*, vol. 14, no. 2, pp. 220–240, 1998.

[32] A.L. Nelson, E. Grant, and T.C. Henderson, "Evolution of neural controllers for competitive game playing with teams of mobile robots," *J. Robot. Auton. Syst.*, vol. 46, no. 3, pp. 135–150, Mar. 2004.

[33] G.J. Barlow, T.C. Henderson, A.L. Nelson, and E. Grant, "Dynamic leadership protocol for S-nets," in *Proc. IEEE Int. Conf. Robotics Automat.,* 2004, vol. 2, pp. 4987–4994.

[34] R.R. Murphy and R.C. Arkin, "Sfx: An architecture for action-oriented sensor fusion," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 1992, vol. 2, pp. 1079–1086.

[35] N.N. Okello, D. Tang, and D.W. McMichael, "TRACKER: A sensor fusion simulator for generalized tracking," in *Proc. Inform. Decision. Control*, IDC, Feb. 1999, pp. 359–364.

[36] R. Willgoss, V. Rosenfeld, and J. Billingsley, "High precision GPS guidance of mobile robots," in *Proc. Australasian Conf. Robotics Automat.*, 2003.

[37] K. Ohno, T. Tsubouchi, B. Shigematsu, S. Maeyama, and S. Yuta, "Outdoor navigation of a mobile robot between buildings based on DGPS and odometry data fusion," in *Proc. IEEE Int. Conf. Robotics Automat.*, 2003, vol. 2, pp. 1978–1984.

[38] K.P. Valavanis and G.N. Saridis, *Intelligent Robotic Systems: Theory, Design and Applications*. Norwell, MA: Kluwer, 1992.

[39] M.T. Long, "Creating a distributed field robot architecture for multiple robots," M.Sc. thesis, USF, Summer 2004.

[40] J. Contreras, A. Losi, and M. Russo, "JAVA/MATLAB simulator for power exchange markets," in *Proc. 22nd Int. Conf. IEEE Power Eng. Soc.*, May 20–24, 2001, pp. 106–111.

[41] S. Sukkarieh, E.M. Nebot, and H.F. Durrant-Whyte, "A high integrity IMU/GPS navigation loop for autonomous land vehicle applications," *IEEE Trans. Robot. Automat.*, vol. 15, no. 3, pp. 572–578, June 1999.

[42] O. Brock and O. Khatib, "High-speed navigation using the global dynamic window approach," in *Proc. IEEE Int. Conf. Robotics Automat.*, 10–15 May 1999, vol. 1, pp. 341–346.

[43] P.A. Bernstein, "A model for distributed system services," *Communicat. Assoc. Computing Mach.*, vol. 39, no. 2, Feb. 1996.

[44] K. Arnold, B. O'Sullivan, R.W. Scheifler, J. Waldo, and A. Wollrath, *The Jini Specification*. Reading, MA: Addison-Wesley, 1999.

[45] M. Long, A. Gage, R. Murphy, and K. Valavanis, "Application of the distributed field robot architecture to a simulated demining task," in *Proc. IEEE Int. Conf. Robotics Automat.*, Barcelona, Spain, Apr. 19–22, 2005.

[46] A. Gage, "Multi-robot task allocation using affect," Ph.D. dissertation, Univ. South Florida, 2004.

Kimon P. Valavanis received his Ph.D. in computer and systems engineering from Rensselaer Polytechnic Institute in 1986. He is currently a professor at the Computer Science and Engineering Department, University of South Florida. His research interests are focused in the area of distributed intelligence systems, unmanned systems, and manufacturing. He has published over 250 book chapters, technical journal/transactions, and conference papers. He has organized several IEEE conferences, and he is the general chair (along with P. Antsaklis) of the 15th Mediterranean Conference on Control and Automation and of the IEEE International Conference on Distributed Human-Machine Systems (with W. Gruver). He is vice president-administration of the IEEE Mediterranean Control Association (MCA) and a former Fulbright Scholar.

Lefteris Doitsidis received the Diploma and M.Sc. in production engineering and management from the Technical University of Crete, Chania, Greece, in 2000 and 2002, respectively. He is currently completing his Ph.D. thesis. From August 2003–June 2004 he was a visiting scholar at the Center for Robot Assisted Search & Rescue, University of South Florida. His research interests are in the areas of multirobot teams, autonomous operation/navigation of unmanned vehicles, and evolutionary computation.

Matt Long is a Ph.D. candidate in Robotics at the University of South Florida (USF). He earned his B.S. in mathematics and computer science in 1998 from the Colorado School of Mines and his M.S. in computer science from USF in 2004 for creating a distributed architecture for teams of heterogeneous robots. This architecture is in use on a number of platforms at USF. He has done prior work in sensor fault tolerance for unmanned systems and is a volunteer with the Center for Robot-Assisted Search and Rescue. His research interests include distributed robotics and computing, software agents, and computational intelligence.

Robin Roberson Murphy received her Ph.D. in computer science in 1992 from Georgia Tech, where she was a Rockwell International Doctoral Fellow. She is professor in the Computer Science and Engineering Department at the University of South Florida with a joint appointment in cognitive and neural sciences in the Department of Psychology. She has published over 100 publications and she has authored the textbook, *Introduction to AI Robotics*. She is director of the Center for Robot-Assisted Search and Rescue at the University of South Florida and recipient of an NIUSR Eagle Award for her participation at the World Trade Center. From 2000-2002, she served as the secretary of the IEEE Robotics and Automation Society and is the North American cochair of the Safety, Security and Rescue Robotics technical committee.

Address for Correspondence: Kimon P. Valavanis, Center for Robot Assisted Search and Rescue, Department of Computer Science and Engineering, University of South Florida, Tampa, FL 33620 USA. E-mail: kvalavan@csee.usf.edu.