Validation of a Distributed Field Robot Architecture Integrated with a MATLAB Based Control Theoretic Environment: A Case Study of Fuzzy Logic Based Robot Navigation

K. P. Valavanis[1], A. L. Nelson, L. Doitsidis[2], M. Long, R. R. Murphy
Center for Robot Assisted Search and Rescue
Department of Computer Science and Engineering
University of South Florida
Tampa, FL 33620

Abstract

The paper presents fundamental aspects of a multi layer, hybrid, deliberative and reactive *Distributed Field Robot Architecture (DFRA)* that has been designed to support functionality of heterogeneous teams of unmanned (ground and aerial) robot vehicles. The *DFRA* is implemented in *Java* using *Jini* to manage distributed objects, services and modules between robots and other system components. It is interfaced with a control theoretic *MATLAB* environment, which is supported and integrated into the *Java* based framework using the *JMatLink Java* class library. This allows modules and services implemented as native interpreted *MATLAB* code to be accessed as remote and distributed objects. The combination of the *Java* based distributed architecture and the use of *MATLAB* in its interpreted form for autonomous robot navigation and control is a unique aspect of the reported research.

Experimental validation of the *DFRA* and its *MATLAB* integration is demonstrated by implementing simple prototype support modules for robot navigation. These modules include: i) a time-history laser filter module; ii) a heuristic GPS-based pose detection module; iii) fuzzy logic controllers that utilize laser, GPS and odometer data as inputs. Navigation experiments in the field utilize single and multiple robots and included scenarios in which a single robot navigated through an environment with many unknown obstacles to reach a distant goal location, and scenarios in which robots executed search routines by traveling through sets of way points. Robots negotiated both static obstacles and dynamic obstacles including other robots.

Acknowledgement: This work has been partially supported by a grant form ONR, 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.

[1] Corresponding author, E-mail:kvalavan@csee.usf.edu

[2] Technical University of Crete, DPEM, Chania, Crete, Greece.

1. INTRODUCTION

Multirobot field research and development systems require a diverse set of system competencies, ranging from delivery of low level services to coordination of complex high level multi-agent interactions. Such systems must support multiple operation modes, including teleoperation, guarded teleoperation, and semi-autonomous operation; they must be capable of mission planning and replanning, task generation, navigation, control, coordination, fault tolerance, and fault detection and isolation (FDI).

The paper presents fundamental aspects of a multi layer *Distributed Field Robot Architecture (DFRA)* designed for heterogeneous teams of unmanned (ground and aerial) robots operating in uncertain and possibly hostile environments. The *DFRA* is a hybrid deliberative/reactive architecture implemented in *Java* using *Jini* to manage distributed objects, services and modules between robots and other system components [39]. Emphasis is given to the control theoretic *MATLAB* based environment of the lower level of the architecture, which is supported and integrated into the *Java* based framework using the *JMatLink Java* class library [30]. The *MATLAB* based environment is then experimentally validated by implementing simple prototype support modules for outdoor mobile robot navigation.

Derivation of such a distributed architecture, during its design as well as during its implementation and testing phases, brings together concepts and ideas from the diverse fields of Distributed Artificial Intelligence, Human Robot Interaction and Multi Agent Systems at the higher levels, combined with Control Theoretic approaches at the lower levels. This is where the main contribution and novelty of the paper lies: although the *DFRA* is implemented in *Java*/*Jini*, the *MATLAB* environment (supported using the *JMatLink* library) 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. Although it is the case that many behavioral architectures and languages support implementation of modules in different languages, this approach allows *MATLAB* to be directly incorporated into behavioral architectures resulting in speed up of development and added flexibility of implementation. The combination of *Java* based distributed architectures and the use of *MATLAB* in its interpreted form for autonomous robot navigation and control is not well represented in the literature, and up to now, no complete approach has been published.

Experimental validation of the *DFRA* and its *MATLAB* integration is demonstrated by implementing simple prototype support modules for robot navigation. These modules include: i) a time-history laser filter module; ii) a heuristic GPS-based pose detection module; iii) fuzzy logic controllers that utilize laser, GPS and odometer data 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 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 the large body of complex state of the art positioning systems and sound mathematical approaches reported in the literature (for example see [41] and [42]).

The paper is organized as follows: Related research is presented below, while Section 2 discusses the *DFRA* and *MATLAB* integration. Section 3 presents the support module development while Section 4 describes details of the fuzzy logic controllers for outdoor navigation, Experimental results are the topic of Section 5 while Section 6 concludes the paper.

A. Related Research

A recent review of autonomous robot control architectures is presented in [20]. Behavioral robotics architectures [3], [4], [31], [34], [38], [39] support robot control within the methodological and conceptual constraints of the behavior based robotics [2]. A defining feature of behavior based architectures is that they provide methods for parallel integration of simple behaviors to generate more complex emergent robot behaviors. 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], [33]. In those examples, utilities in the form of standard *MATLAB* function calls have been developed to allow for 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], [40]. In those cases, however, *MATLAB* does not run in its full interpreted form on remote self powered agents such as autonomous mobile robots.

Although it is the case that many behavioral architectures and languages support implementation of modules in different languages, the approach reported here allows *MATLAB* to be directly incorporated into behavioral architectures resulting in speed up of development and added flexibility of implementation. Modules and services implemented as native interpreted *MATLAB* code can be accessed as remote and distributed objects. This is particularly useful because all *MATLAB* Toolboxes can be fully access, and *MATLAB* scripts need not be compiled, even in their final production-phase form. The combination of *Java* based distributed architectures using *MATLAB* in its interpreted form for autonomous robot navigation and control is not well represented in the literature, and up to now, no complete approach has been published.

Methods employing fuzzy logic based mobile robot navigation are reported in [5], [7], [10], [11], [12], [13], [14], [15], [16] and [17]. Most of these address navigation in indoor structured laboratory environments. In [18] and [19] outdoor navigation using fuzzy systems is discussed. Waypoint navigation in which vehicles move in predefined environments is reported in [26] and [27]. Outdoor environment navigation using

odometer data only 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], [24], [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]. The reported research in [18] uses dead reckoning 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 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], [6], [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.

2. DISTRIBUTED FIELD ROBOT ARCHITECTURE AND INTEGRATION WITH MATLAB

The *Distributed Field Robot Architecture (DFRA)*, presented in detail in [39], is a distributed multi agent *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. Hence, 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**: The architecture must support common robotic paradigms. 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**: Robot hardware and software platforms lave a limited life cycle; hence, it is important to build on a base that is open, flexible and extensible. While specific robot hardware is beyond the scope of this work, 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 foundation contributes to system level fault tolerance, while the use of *SFX* incorporates prior work on robot fault recovery.
- **Adaptable**: The system should be able to adapt to its operating environment. Because the overall system is implemented in *Java*, software portability is not an issue as long as all services correctly implement specified interfaces. However, modules need to adapt and be good ``network 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 at 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 rather than static 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. For example, if a robot has two identical cameras providing color images (the capabilities of the sensors), then a client will not have a preference between which camera provides an image, all else being equal. However, if the two cameras are mounted in different locations on the robot (the attributes of the sensor) then there may be differences in the client's preference for service. Thus, a service should provide a listing of its various capabilities and attributes to clients in the distributed system, allowing a client to make intelligent choices related to available services. Since the design constraints require adaptation to a changing environment, these capabilities and attributes must be changeable as the system evolves or services fail.

The above design constraints are addressed by the application of three key technologies: the *SFX* architecture, *Java*, and *Jini*, as briefly discussed below.

A. SFX Base Architecture

SFX is a managerial architecture [2] with deliberative and reactive components designed to incorporate sensor fusion. The primary component of the reactive layer is the *behavior*. A behavior maps from 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. For example, a perceptual schema may process a camera image to generate a mine percept representing the location of the closest mine in the image.

Once a percept is generated, the behavior passes the information to a motor schema. The motor schema incorporates the control necessary to act on the percept. This may involve actions ranging moving the robot to orienting a pan-tilt unit to achieve a better view. Behaviors can act and react rapidly, allowing the robot to operate in real time.

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.

B. 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 3D visualization package; 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 even have 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 has traditionally 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 re-interpret the byte code.

C. Jini Distributed Layer

Middleware frameworks [43] are abstractions of a distributed computing environment. These frameworks allow software developers to more easily extend system infrastructures into a distributed environment. This is because the middleware is ``in the middle'', between the operating system or network services and the application layer, abstracting the details of the system specific networking and other low level code. *Jini* is an example of a middleware framework [44], which has a goal of providing for 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; *Jini* uses a leasing mechanism to handle partial failures 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 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.

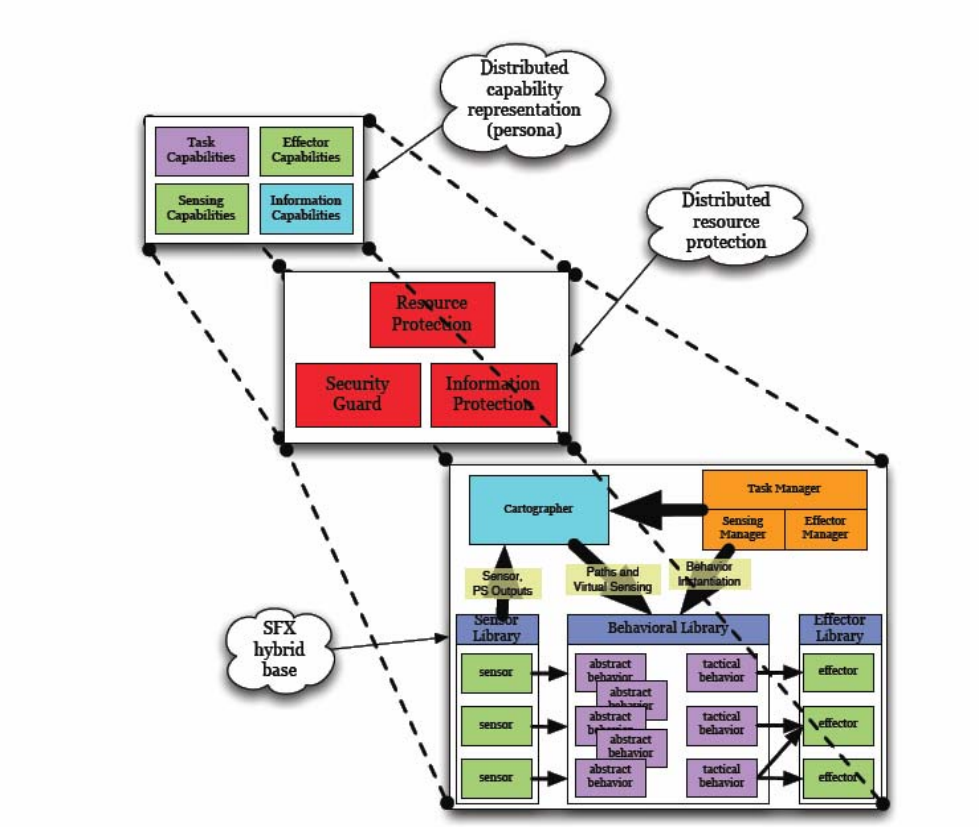


Figure 1: *Distributed Field Robot Architecture* showing the relationship between the distributed system components and the base *SFX* architecture.

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.

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].

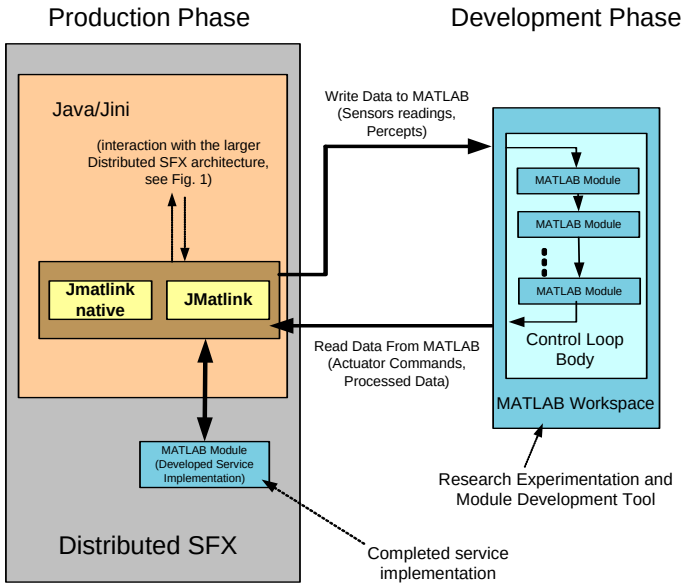


Figure 2: Relationship between *MATLAB* and overall *DFRA*

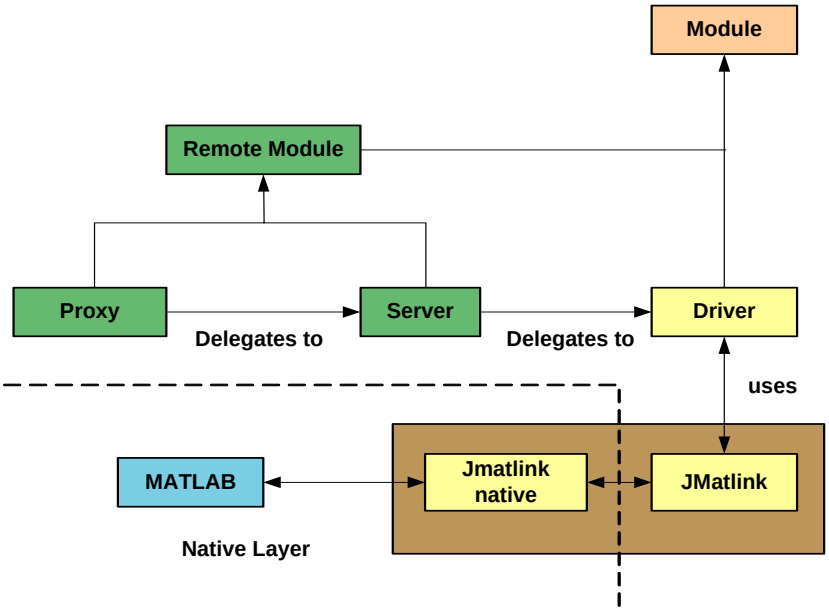


Figure 3: Horizontal hierarchy with *MATLAB* as driver implementation module

3. SUPPORT MODULE DEVELOPMENT

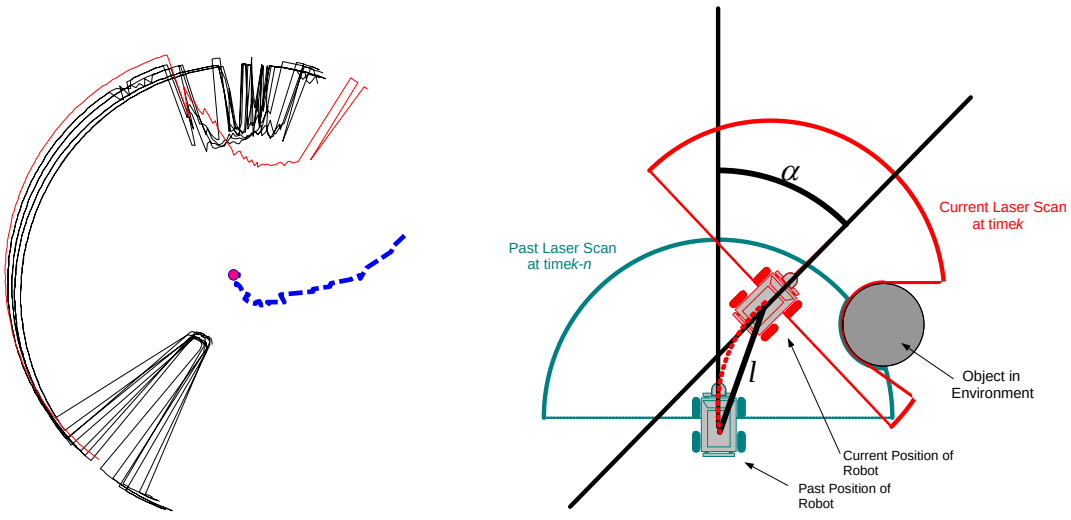
Module prototyping using *MATLAB* is demonstrated to support multi sensor fuzzy logic based navigation. Two very simple modules are discussed:

- A laser range data filter designed to reduce noise and ghost readings caused by laser bouncing, variations in grass and vegetation, as well as other unforeseen outdoor environment conditions. This module is an example of a heuristic filter that relies on *MATLAB's* matrix and data processing power for ease of implementation.
- 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.

A. 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* degrees, centered on the robot body attached reference frame. Each consecutive point in a single scan is offset by *1* degree (*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*, …, *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 α 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 trade off between accuracy and response time for rotation measurements.



(a) (b)

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*.

Each laser scan is represented by:

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

where $\mathbf{x} = [x_1, x_2, ..., x_I]$, $\mathbf{y} = [y_1, y_2, ..., y_I]$, $I = 181$ 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* degree 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 **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 α from Figure 4b). Then, using a standard rotation and translation transformation **T**, passed scans are transformed into the current scan's frame of reference:

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

T is given in terms of linear and rotational offsets and it is recalculated for each previous scan as follows:

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

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

$$\mathbf{r} = \begin{bmatrix} r_1 & r_2 & \cdots & r_{181} \end{bmatrix}, \text{ where } r_i = \sqrt{x_i^2 + y_i^2} \tag{4}$$

The matrix **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:

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

Hence, objects detected in multiple current and past scans appear as range readings of similar values in multiple rows of **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 **R**. These are calculated respectively as:

$$\begin{aligned} \mathbf{r}_{\text{var}} &= \text{var}(\mathbf{R}) \\ &= \begin{bmatrix} \text{var}(r_1(k), r_1(k-1), \cdots, r_1(k-n)) & \cdots & \text{var}(r_{181}(k), r_{181}(k-1), \cdots, r_{181}(k-n)) \end{bmatrix} \end{aligned} \tag{6}$$

$$\mathbf{r}_{\mu} = \begin{bmatrix} \mu(r_1(k), r_1(k-1), \cdots, r_1(k-n)) & \cdots & \mu(r_{181}(k), r_{181}(k-1), \cdots, r_{181}(k-n)) \end{bmatrix} \tag{7}$$

$$\mathbf{r}_{\max} = \begin{bmatrix} \max(r_1(k), r_1(k-1), \cdots, r_1(k-n)) & \cdots & \max(r_{181}(k), r_{181}(k-1), \cdots, r_{181}(k-n)) \end{bmatrix} \tag{8}$$

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

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

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 i^{th} elements of $\mathbf{r}_\Phi$, $\mathbf{r}_{\text{var}}$ and $\mathbf{r}_{\max}$, respectively, and ν 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 (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.

B. Position Detection

Pose (position) detection has been based on current and passed GPS readings. Pose is represented by a triplet, (x, y, θ). 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* meters 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* meter when the robots traveled over a *10* meter path with known absolute ground position.

This coarse pose detection method is in no way state of the art; it is used to demonstrate how the overall *DFRA-MATLAB* framework may be used with components of varying degrees of refinement. Although this is the case, as shown in Section 5, results have been satisfactory in terms of waypoint, pattern following and goal point navigation.

4. FUZZY LOGIC CONTROLLER

The overall structure of the multi sensor 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 with the robots 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.

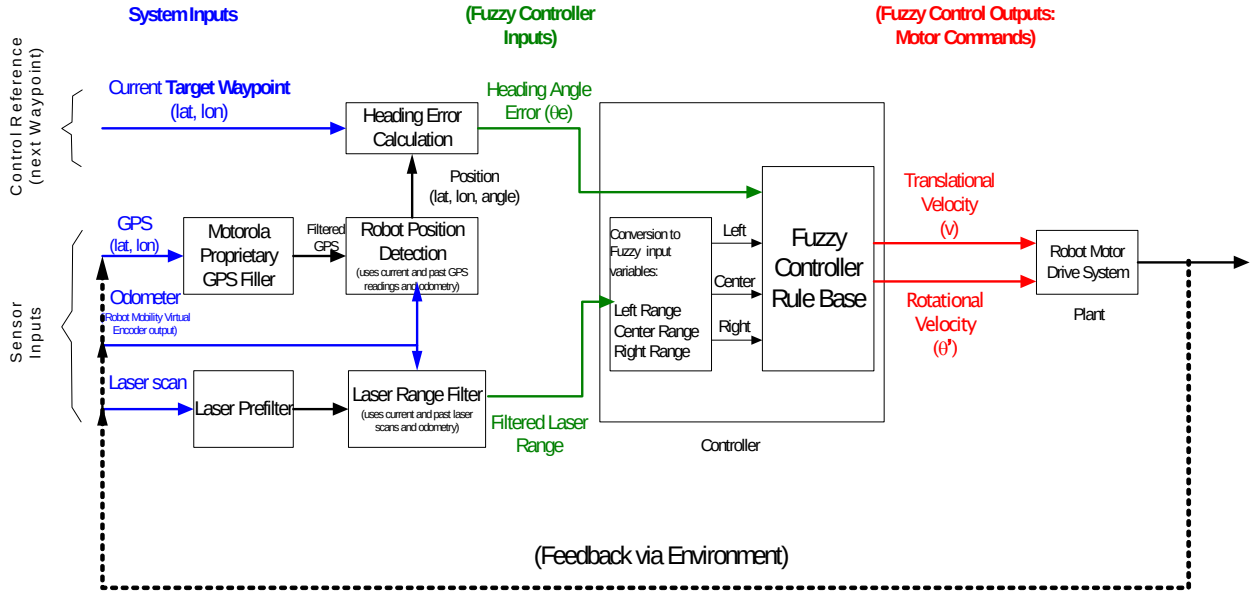


Figure 5: Control system shown as a collection of interrelated modules

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* meters (*25* feet). The left and right areas have a width of 70^0 each and the center area of 40^0.

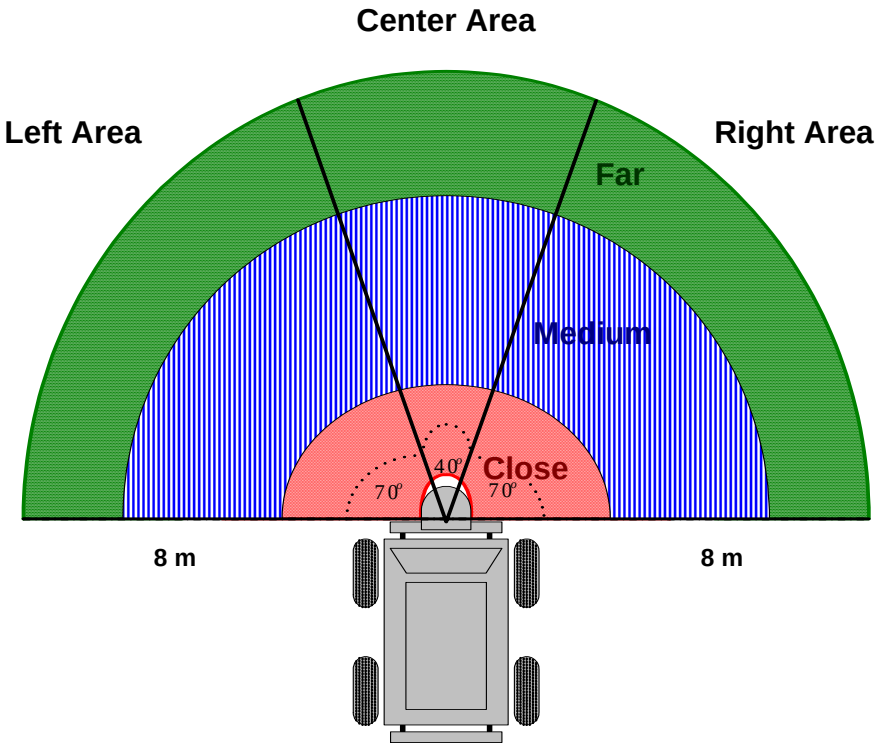
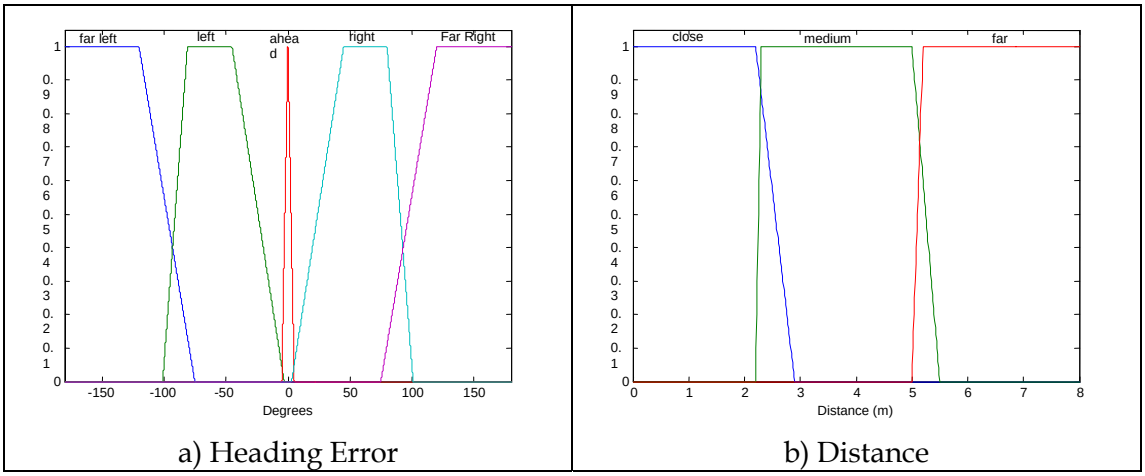


Figure 6: Laser scan area radial sectors divided into close, medium and far areas

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*, *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 shown in Figure 7. They are empirically derived based on extensive tests and experiments.



a) Heading Error b) Distance

Figure 7: Membership functions for the input variables

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* meters. 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* degrees.

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

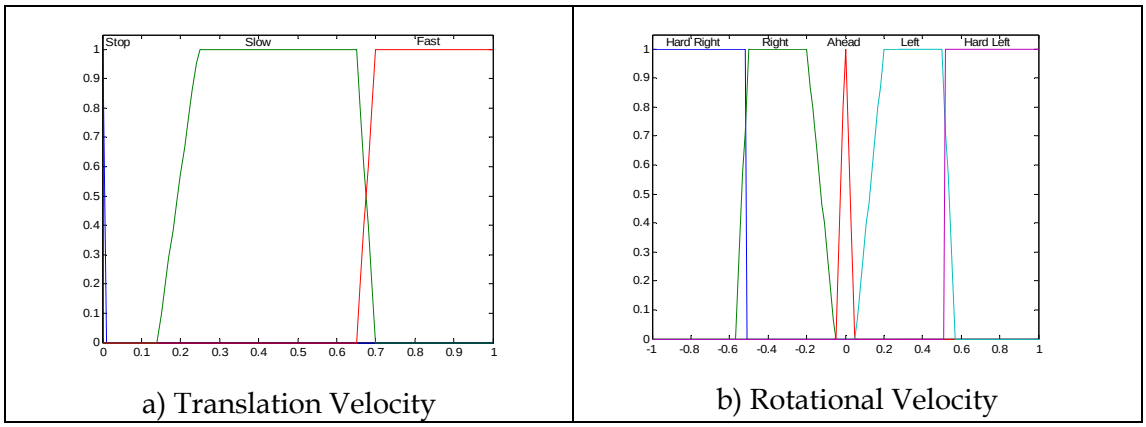
The value of the output variable *tr* is expressed by the fuzzy sets *ST*, *SL*, *F* referring to *stop*, *slow*, and *fast*. The value of the output variable *rv* is expressed by the fuzzy sets *HRR*, *RR*, *AHR*, *LR*, *HLR* referring to *hard right*, *right*, *ahead*, *left*, *hard left*.

The output commands are normalized in a scale from 0 to 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. The output variables membership functions are presented in Figure 8.

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,…, 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. A sample of the rule base is presented in Table I.



a) Translation Velocity b) Rotational Velocity

Figure 8: Output variables membership functions

Left distance from obstacle	Center distance from obstacle	Right distance from obstacle	Heading Error	Translational Velocity	Rotational Velocity
far	*far*	*far*	*far left*	*stop*	*hard left*
far	*medium*	*far*	*Left*	*slow*	*hard left*
far	*far*	*far*	*far right*	*stop*	*hard right*
close	*close*	*close*	*far right*	*stop*	*hard right*

Table I: Sample rule base

The generic mathematical expression of the *jth* navigation rule is given by:

$$\mu_{R^{(j)}}(d_i, he, tr, r) = \min[\mu_{D_i^i}(d_i), \mu_{HE^{(j)}}(he), \mu_{TR^{(j)}}(tr), \mu_{RV^{(j)}}(rv)] \qquad (15)$$

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

$$\mu_N^*(tr, vr) = \max_{d_i, he} \min[\mu_{AND}^*(d_i, he), \mu_R(d_i, he, tr, rv)] \qquad (16)$$

where $\mu_R(d_i, he, tr, rv) = \bigcup_{j=1}^{J} \mu_{R^{(i)}}(d_i, he, tr, rv)$. The navigation action dictates change in robot speed and/or steering correction and it results from the deffuzification formula, which calculates the center of the area covered by the membership function computed from (16).

Timing issues play a role in real time systems and controllers must be designed to accommodate an acceptable variation in control delay. The system used in this research produces a variable delay depending on application and system load. For the results presented in this paper, the system controller loop delay averaged about *0.2*

seconds or *5* Hz. To accommodate this, the fuzzy rule set has been designed to function in a control loop window of between *10* Hz to *1*Hz.

5. EXPERIMENTAL RESULTS

Experiments have been performed in outdoor environments using two *ATRV-Jr* mobile robot platforms that have been modified as stated in Section 1. There is access to all *Mobility* functions (which may be called if needed), but *Mobility* itself is not used as a support software environment. Reported results include both odometer and GPS error quantification, as well as fuzzy logic based navigation and collision avoidance.

A. Odometer and GPS Error Quantification

The robot vehicles followed three predetermined test patterns: forward and backward motion along a *15* meter straight line; tracing a *10* meter square; tracing a circle with a radius of *25* meters. Paths traveled by the robots (as judged by GPS and odometer position measurement methods) are shown as sequences of points. The origin was the starting point of the robot. A total of *6* tests per pattern over a *3* day period have been conducted to quantify raw GPS, filtered GPS (using the Motorola supplied filter within the GPS unit) and odometer errors.

Recorded data included GPS readings, actuator commands, odometer generated position, time stamps, laser range values. Since a full control loop cycle required about *0.2* seconds, data were collected at a rate of approximately *5* Hz.

During the tests, positions calculated from odometer and positions measured from GPS deviate from one another, as expected. In the rectangular and circular tests, odometer and GPS positions deviate by approximately *2.0* meters and *6.0* meters on average, respectively. This is to be expected because odometer errors are cumulative and influenced by the size and duration of the test patterns. Figure 9 shows collected GPS and wheel odometer position data; each panel shows data collected over a single example run of each test pattern.

Considering all tests for the forward and backward robot movement, the cumulative odometer error has been found to be *0.4%* per meter traveled, while the average GPS position error has been found to be *0.91* meters with an error standard deviation of *0.52* meters. This has been feasible because intermediate points along the path have been specified, verifying the robot's true real world position (that may be compared to the GPS position if needed).

However, no real conclusion can be made for the other two test patterns because of incremental odometer errors and uncertainty in the robot's true real world position.

The path generation for the square pattern has been generated by a timed sequence of forward commands followed by a rotation command calibrated to produce a *90* degree 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 have been generated by applying the same actuator arc command repeatedly. This produces the same curvature, regardless of duration, so the pattern is invariant with respect to controller loop delay.

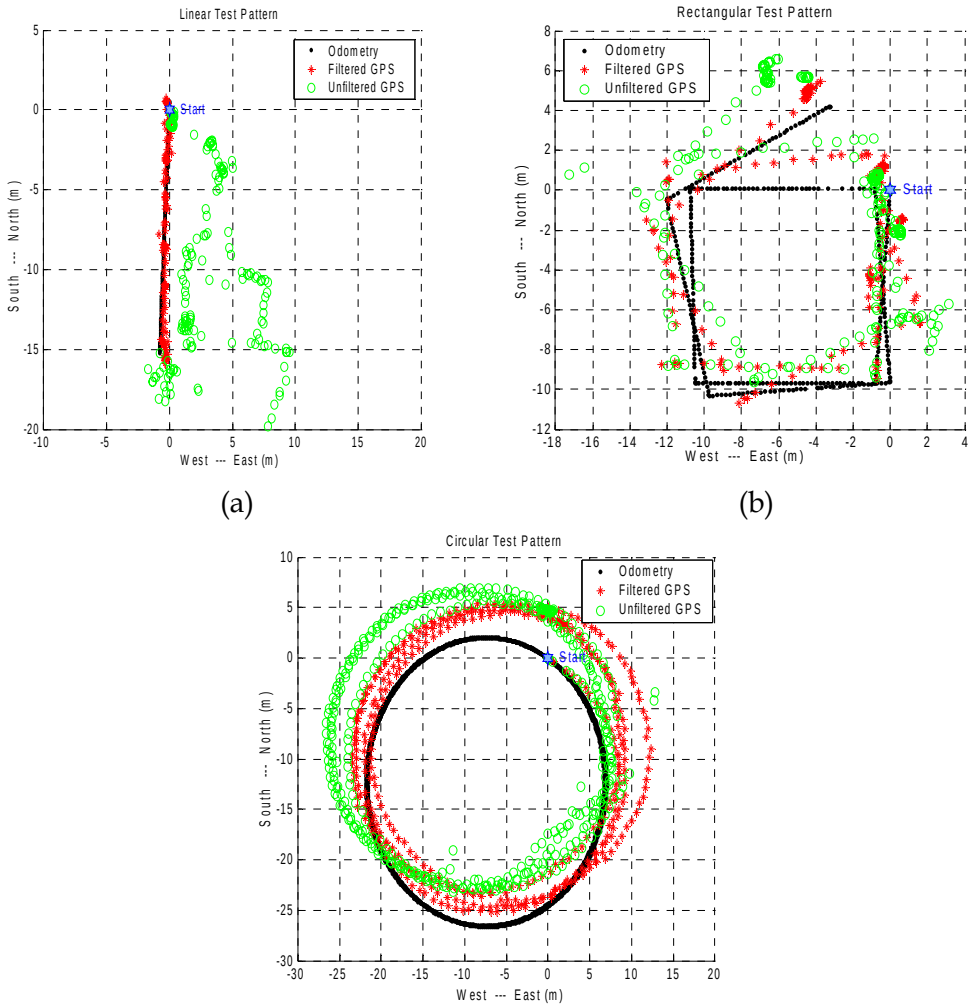


Figure 9: Odometer, unfiltered and filtered GPS data comparison

B. *Dynamic Fuzzy Logic Based Control in Outdoor Environments*

Testing and validation of the designed fuzzy logic controller within the overall *DFRA* as presented in Section 2, has been performed using single and multiple robots. Experiments have been performed in an outdoor (somewhat uneven terrain) environment with dirt, grass, trees 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 include raster scans with two robots starting from different initial positions, avoiding each other as well as other obstacles found in their path.

To facilitate further robot deployment, a GUI has been designed that allows human operators to monitor robot movement, modify dynamically their waypoints or current goal positions, or define areas to perform a raster search (select a desired area to scan specifying a lane spacing parameter). A screen capture of the GUI is presented in

Figure 10. Although robots receive final goal positions and waypoint sets from the GUI, all control processing is done 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 10: Graphical User Interface

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 11 shows photos of a robot in the tree covered area. Archived videos may be viewed at http://www.csee.usf.edu/~aanelson/Robot_Movies.html.

Figure 11: Robot navigating in an area with trees

Figure 12 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 has been repeated several times, and Figure 13 shows the different paths followed by the robot for three repetitions of the same experiment. Path differences are 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. The outdoor environment is effectively continuous; hence a small change in movement or initial conditions may result in a different path followed upon a repetition of the same goal following experiment. 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 14 presents a sequence of photos while the robot navigates among the trees (sequence from upper left to lower right).

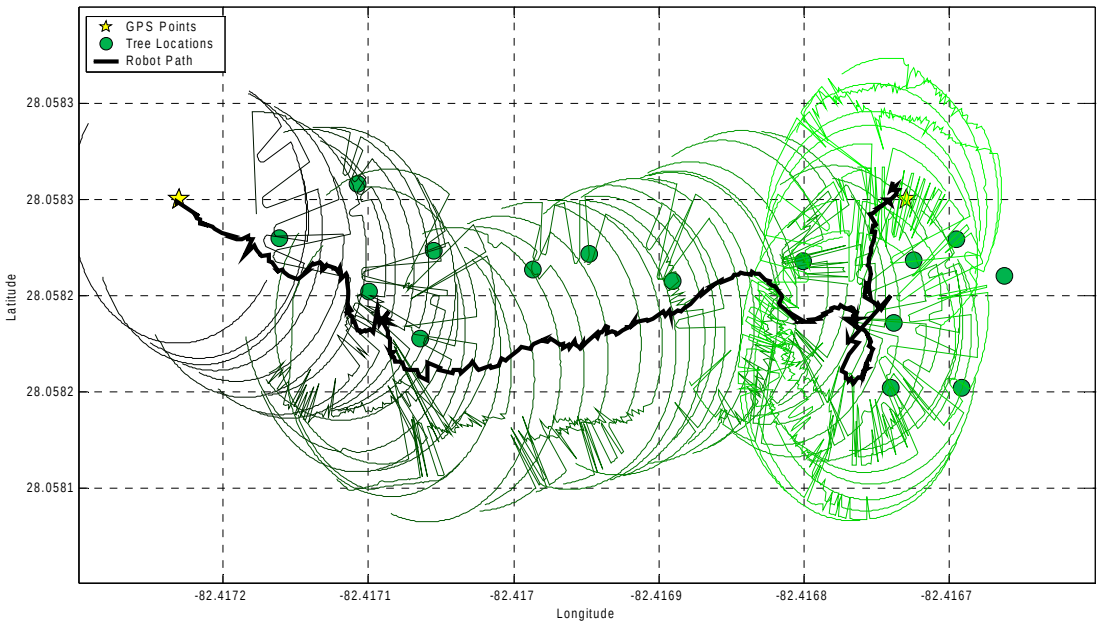


Figure 12: Robot path – laser scans are shown

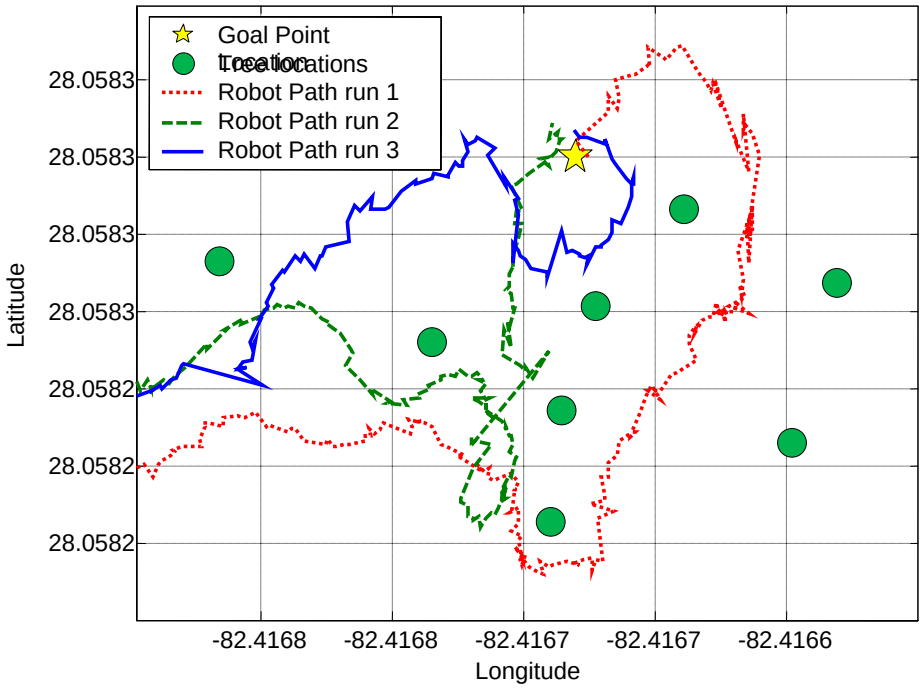


Figure 13: Three different paths for the same scenario.

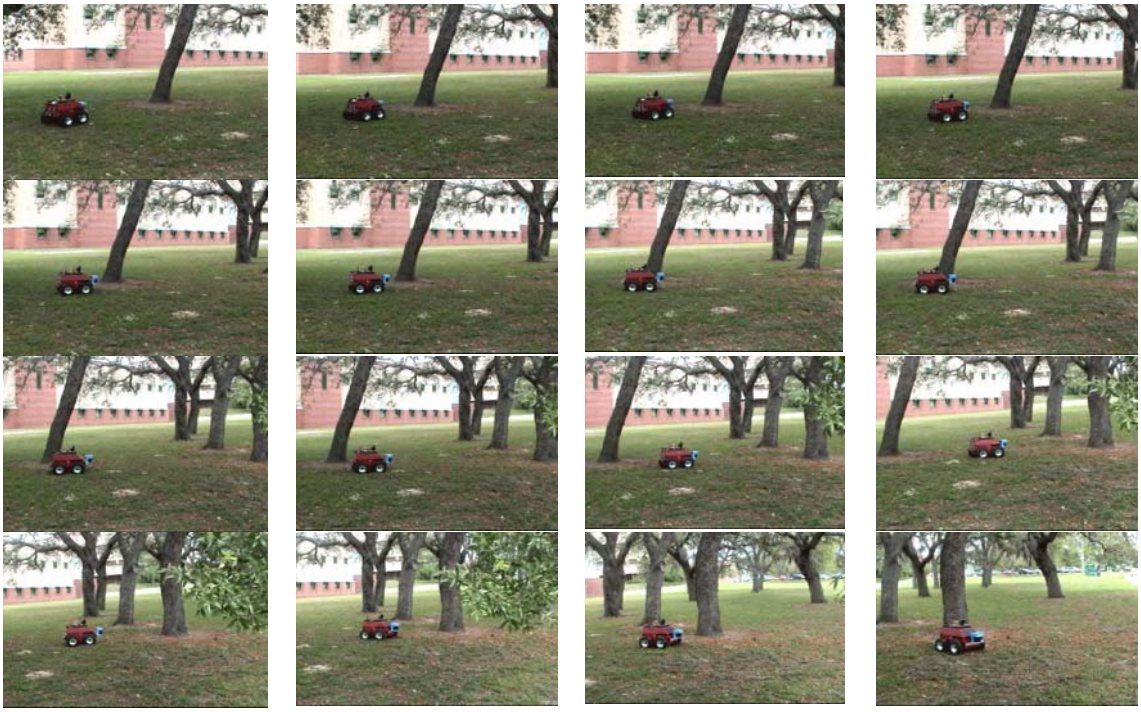


Figure 14: A sequence of images showing the robot consecutively avoiding two trees

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 15; 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 16 and visualized with a sequence of images in Figure 17.

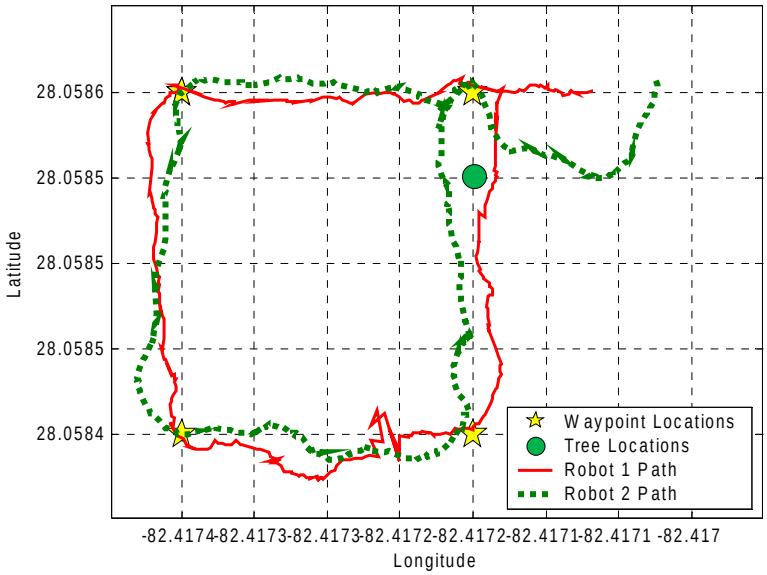


Figure 15: Path followed by the two robots in the second experiment

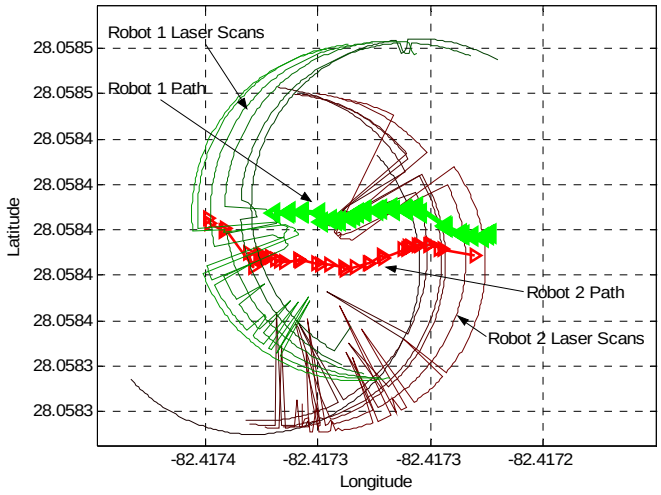


Figure 16: Robots avoiding each other

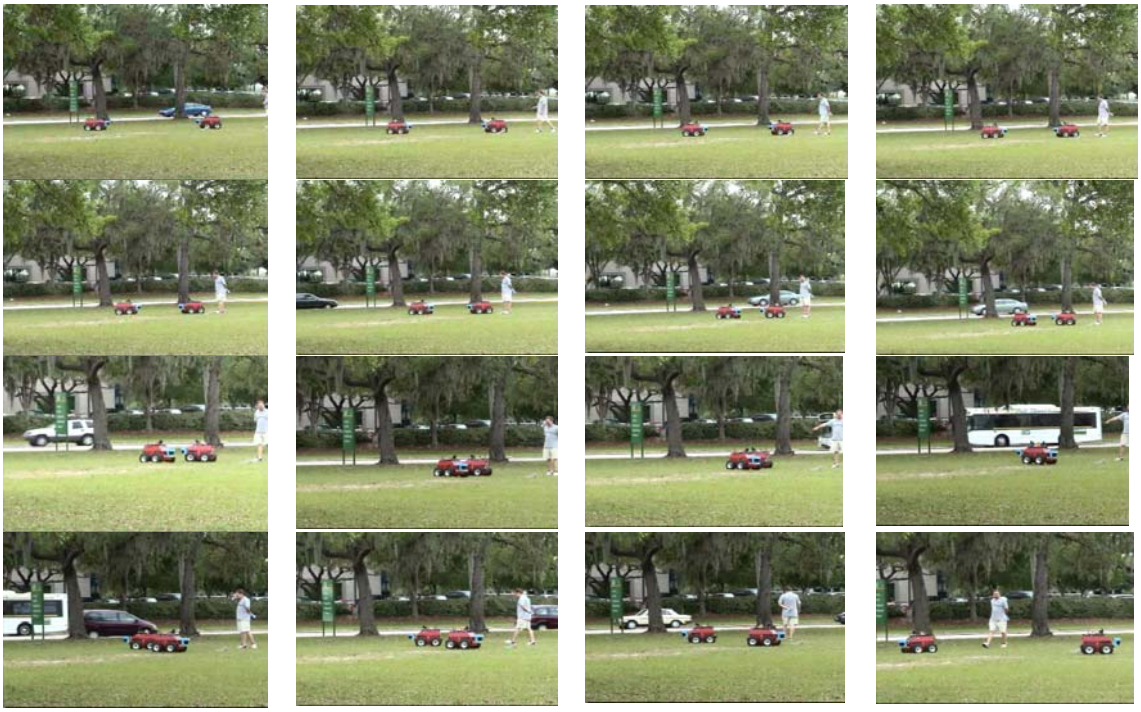


Figure 17: Images of two robots moving towards each other and avoiding one another

Experiment 3

A raster search is 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 18. The experiment has been repeated several times with similar results.

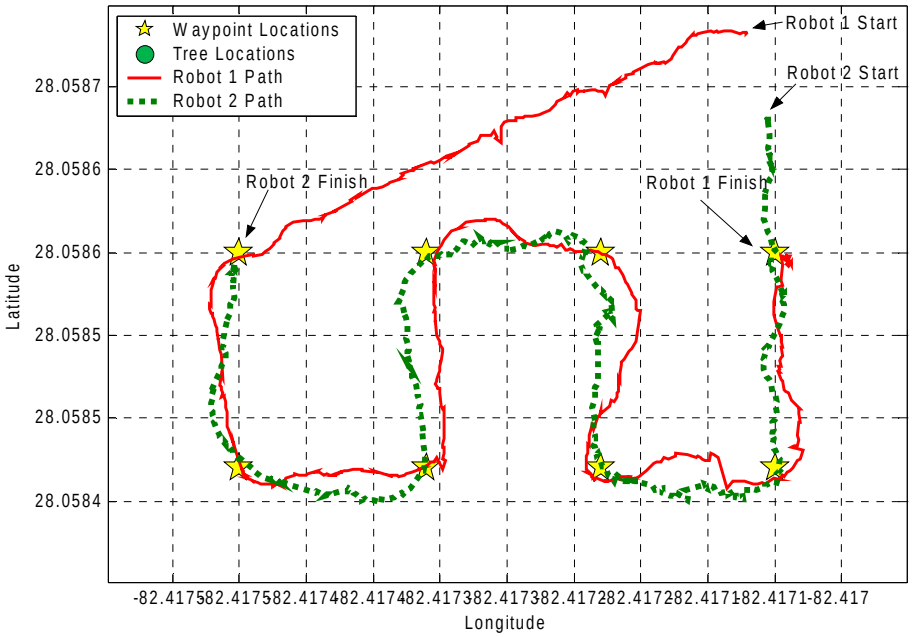


Figure 18: Raster search performed by the two robots

6. CONCLUSIONS AND DISCUSSION

This paper has presented a *DFRA* and its integration with *MATLAB* that is capable to support simple and complex functionality of heterogeneous teams of robot systems. This architecture has been used to demonstrate multi sensor mobile robot fuzzy logic based navigation in outdoor environments.

The main contribution of the paper 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] to [[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.

On going research includes controller enhancement involving additional sensors like IMU, FLIR and standard video; derivation of remote supervisory controllers for coordinated movement and variable robot autonomy; inter robot communication and multi robot distributed control.

Further, research is conducted on coordinated control of aerial unmanned VTOL vehicles and ground robots (like the *RAPTOR 90* and the *Rotomotion Bergen Observer* and *Twin*, and the *ATRVs)* where goal points for the ground vehicles, specific locations to be reached and areas to be searched will be dictated by the helicopter through its on board vision system, or with the aid of a human operator interpreting such information and commanding robots accordingly (see Figure 19).

Figure 19: Robots autonomously executing a pattern search while being observed by a helicopter with camera (remotely controlled).

REFERENCES

1. Saffiotti, A., "Handling uncertainty in control of autonomous robots," in Uncertainty in Information Systems, edited by A. Hunter and S. Parsons, Springer, LNAI 1455, pp: 198-224, 1998.
2. Murphy, R. R., Introduction to AI Robotics, MIT Press, 2000.
3. Arkin, R. C., Behavior-Based Robotics, MIT Press, 1998.
4. Balch, T., Parker L. E., (Editors), Robot Teams, from Diversity to Polymorphism, A. K. Press, 2002.
5. Doitsidis, L., Valavanis, K. P., Tsourveloudis, N. C., "Fuzzy Logic Based Autonomous Skid Steering Vehicle Navigation", In *Proceedings of the IEEE International Conference on Robotics and Automation,* pp: 2171-2177, Washington DC, 2002.
6. Valavanis, K. P., Hebert, T., Kolluru, R., Tsourveloudis, N. C., "Mobile Robot Navigation in 2-Dynamic Environments Using Electrostatic Potential Fields," *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 30, No. 2, Part A, pp: 187-196, 2000.
7. Tsourveloudis, N. C., Valavanis, K. P., Hebert, T., "Autonomous Vehicle Navigation Utilizing Electrostatic Potential Fields and Fuzzy Logic," *IEEE Transactions on Robotics and Automation*, Vol. 17, No. 4, pp: 490-497, 2001.
8. Abdessemed, F., Benmahammed, K., Monacelli, E., "A fuzzy-based reactive controller for a non-holonomic mobile robot," In *Robotics and Autonomous Systems,* article in press.
9. Lee, T. L., Wu, C. J., "Fuzzy Motion Planning of Mobile Robots in Unknown Environments," *Journal of Intelligent Robotic Systems,* Vol. 37, pp: 177-191, 2003.
10. Aguire, E., Gonzalez, A., "Fuzzy Behaviors for Mobile Robot Navigation: Design, Coordination and Fusion," *International Journal of Approximate reasoning,* 25: 225-289, 2000.
11. Goodridge, S. C., Luo, R. C., "Fuzzy Behavior Fusion for Reactive Control of an Autonomous Mobile Robot: MARGE," In *Proceedings of the IEEE International Conference on Robotics and Automation,* pp: 1622-1627, San Diego, CA, 1997.
12. Ishikawa, S., "A Method of Indoor Mobile Robot Navigation by Using Fuzzy Control," In *Proceeding of the IEEE/RSJ International Workshop on Intelligent Robots and Systems IROS,* pp: 1013-1018, Osaka, Japan, 1991.
13. Li, W., "Fuzzy Logic-based 'Perception-Action' Behavior Control of a Mobile Robot in Uncertain Environment," In *Proceedings of the 3rd IEEE Conference on Fuzzy Systems,* Vol. 3, pp: 1626-1631, 1994.
14. Pin, F. G., Watanabe, Y., "Navigation of Mobile Robots Using Fuzzy Logic Behaviorist Approach and Custom Design Fuzzy Inference Boards," *Robotica,* Vol. 12, 1994.
15. Saffiotti, A., "Fuzzy Logic in Autonomous Robotics: Behavior Coordination," In *Proceedings of the IEEE International Conference on Fuzzy Systems,* pp: 573-578, 1997.
16. Saffiotti, A., Konolige, K., Ruspini, H. E., "A Multivariable Logic Approach to Integrating and Planning and Control," *Artificial Intelligence,* 76: 481-526, 1995.
17. Tunstel, E., "Mobile Robots Autonomy Via Hierarchical Fuzzy Behavior," In *Proceedings of the 6th International Symposium on Robotics and Manufacturing,* pp: 837-842, 1996.

18. Seraji, H., Howard, A., "Behavior-Based Robot Navigation on Challenging Terrain: A Fuzzy Logic Approach," *IEEE Transactions on Robotics and Automation*, Vol. 18, No. 3, pp: 308-321, 2002.
19. Hagras, H., Callaghan, V., Colley, M., "Outdoor Mobile Robot Learning and Adaptation," *IEEE Robotics and Automation Magazine*, pp: 53-69, 2001.
20. Mali, A. D., "On the Behavior-Based Architectures of Autonomous Agency," *IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews*, Vol. 32, No. 3, pp: 231-242, August 2002.
21. Borenstein, J. and Feng. L., "Measurement and Correction of Systematic Odometry Errors in Mobile Robots," *IEEE Journal of Robotics and Automation*, Vol. 12, No. 6, , pp: 869-880, December 1996
22. Ojeda, L. and Borenstein, J., 2003, "Reduction of Odometry Errors in Over-constrained Mobile Robots," In *Proceedings of the UGV Technology Conference at the 2003 SPIE AeroSense Symposium*, Orlando, FL, April 21-25, 2003.
23. Panzieri, S., Pascucci, F., Ulivi, G.,"An Outdoor Navigation System Using GPS and Inertial Platform," *IEEE Transactions on Mechatronics*, Vol. 7, No. 2, pp:134-142, June 2002.
24. Ohno, K., Tsubouchi, T., Shigematsu, B., "Outdoor Navigation of a Mobile Robot between Buildings based on DGPS and Odometry Data Fusion", In *Proceedings of International Conference in Robotics and Automation*, pp: 1978-1984, Taipei, 2003.
25. Thrapp, R., Westbrook, C., Subramanian, D., "Robust Localization algorithms for an autonomous campus tour guide," In *Proceedings of the IEEE International Conference on Robotics and Automation,* Vol. 2, pp: 2065-2071, 2001.
26. Vaneck, T.W., "Fuzzy Guidance Controller for an Autonomous Boat," *IEEE Control Systems Magazine*, pp: 43-51, April, 1997
27. Bruch, M.H., Gilbreath, G.A., Muelhauser, J.W.,and J.Q. Lum, "Accurate Waypoint Navigation Using Non-differential GPS," *AUVSI Unmanned Systems 2002*, Lake Buena Vista, FL, July 9-11, 2002.
28. Long, M. T., Murphy, R. R., Parker, L. E., "Distributed multi-agent diagnosis and recovery from sensor failures," in *Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems,* Vol. 3, pp: 2506-2513, 2003.
29. Nelson, A. L., Doitsidis, L., Long, M. T., Valavanis, K. P., Murphy, R. R., "Incorporation of Matlab into a Distributed Behavioral Robotics Architecture", to be presented in IROS 2004.
30. S. Müller, H. Waller, "Efficient Integration of Real-Time Hardware and Web Based Services Into MATLAB," *ESS'99 11th European Simulation Symposium and Exhibition*, Erlangen-Nuremberg, 1999, ESS'99, Oct. 26-28, 1999. JMatLink download site: http://www.held-mueller.de/JMatLink/
31. L. E. Parker, "ALLIANCE: An Architecture for Fault Tolerant Multi-Robot Cooperation," *IEEE Transactions on Robotics and Automation*, Vol. 14, No. 2, pp: 220-240, 1998.
32. A. L. Nelson, E. Grant, T.C. Henderson, "Evolution of neural controllers for competitive game playing with teams of mobile robots," *Journal of Robotics and Autonomous Systems*, Vol. 46, No. 3, pp: 135-150, Mar 2004.
33. G. J. Barlow, T. C. Henderson, A L. Nelson, E. Grant, "Dynamic Leadership Protocol for S-nets," *Proceedings*, *2004 IEEE International Conference on Robotics and Automation* (ICRA), New Orleans, LA, April 2004.

34. R. R. Murphy, R. C. Arkin, "Sfx: An Architecture For Action-oriented Sensor Fusion", *Proceedings of IEEE/RSJ Intelligent Robots and Systems*, Vol. 2, July 7-10, pp: 079-1086, 1992.
35. N. N. Okello, D. Tang, D. W. McMichael, "TRACKER: a sensor fusion simulator for generalized tracking," *Proceedings of Information, Decision and Control, IDC,* pp: 359-364, February 1999.
36. R. Willgoss, V. Rosenfeld, J. Billingsley, "High precision GPS guidance of mobile robots," *Proceedings of the Australasian Conference in Robotics and Automation*, 2003.
37. K. Ohno, T. Tsubouchi, B. Shigematsu, S. Maeyama, S. Yuta, "Outdoor Navigation of a Mobile Robot Between Buildings based on DGPS and Odometry Data Fusion,", *Proceedings of the IEEE International Conference on Robotics and Automation*, Vol. 2, pp: 1978-1984, Taipei, TW, 2003.
38. K. P. Valavanis, G. N. Saridis, Intelligent Robotic Systems: Theory, Design and Applications, 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, M. Russo, "JAVA/MATLAB simulator for power exchange markets", *Power Industry Computer Applications*, PICA 2001. Innovative Computing for Power - Electric Energy Meets the Market. *22nd International Conference on IEEE Power Engineering Society,* pp: 106-111. May 20-24, 2001.
41. Sukkarieh, S.; Nebot, E.M.; Durrant-Whyte, H.F.; "A high integrity IMU/GPS navigation loop for autonomous land vehicle applications," *IEEE Transactions on Robotics and Automation*, vol. 15, no.3, pp. 572-578, June 1999.
42. Brock, O.; Khatib, O., "High-speed navigation using the global dynamic window approach," Proceedings *of the 1999 IEEE International Conference on Robotics and Automation*, vol. 1, 10-15 May 1999, pp. 341-346.
43. P. A. Bernstein, "A Model for Distributed System Services," *Communications of the Association for Computing Machinery*, vol. 39, no. 2, Feb. 1996.
44. K. Arnold, B. O'Sullivan, R. W. Scheifler, J. Waldo, A. Wollrath, *The Jini Specification*, Addison-Wesley, 1999.