

Incorporation of MATLAB into a Distributed Behavioral Robotics Architecture

A. L. Nelson, L. Doitsidis, M. T. Long, K. P. Valavanis, and R. R. Murphy

Center for Robot Assisted Search and Rescue (CRASAR)

Dept Computer Science and Engineering

University of South Florida

Tampa, Florida 33620

Email: aanelson@csee.usf.edu

Abstract—This paper presents a method that integrates MATLAB into a distributed behavioral robotics architecture. The architecture is written in Java and uses the Jini platform for distributed object registration, lookup and remote method invocation. The method described here can be used to integrate MATLAB into any Java-based behavioral architecture. The form of the integration allows a running MATLAB workspace to be accessed as a distributed object within the larger Java/Jini-based architecture. This is beneficial because MATLAB scripts and functions may be called in interpreted form and can make full use of MATLAB tool boxes and have access to the MATLAB workspace environment. This is not possible when MATLAB scripts are compiled into stand-alone C++, Java or p-code. The use of the architecture is demonstrated on an *iRobot* ATRV-JR robot and remote computer workstation. Experiments have been conducted to quantify GPS and odometry errors in outdoor environments using automated methods supported by the distributed architecture.

Keywords—mobile Distributed architecture; autonomous robot control; control architecture; MATLAB; Jini; GPS-based navigation;

I. INTRODUCTION

This paper is motivated by the challenge to provide a platform to support advanced robot control research both at the AI level [1] and at the control-theoretic level [17]. The architecture, referred to in this paper as Distributed SFX, supports the implementation of large scale complex behavior-based robot control but still provides tools for low level experimental research within a single structured paradigm [5][10].

The overall distributed behavioral architecture is implemented in Java and uses Jini to manage distributed objects and services between robots and computers. Using the JMatLink library [2], MATLAB is supported within the larger Java/Jini based system to allow for mathematical control-theoretic research and experimentation and for rapid prototyping of both 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, the work described here allows MATLAB, a standard tool

for developing control-theoretic concepts offline, to be directly incorporated into behavioral architectures. This provides advantages such as speed up of development, added flexibility of implementation.

The architecture is currently implemented on *iRobot* ATRV-JR robots and computer workstations. The robots have been fitted with a versatile array of sensors including GPS, planar laser range sensors, heat imaging video (FLIR) standard video, and inertial gyroscopes. To demonstrate the flexibility of the architecture, and to lay the groundwork for outdoor robot navigation, a series of tests have been performed. These tests included automated GPS and odometry error quantification experiments (reported on in the results section of this paper). During the tests, the distributed architecture ran both on a robot in the field and on a computer workstation in the lab. This demonstrated distributed object sharing and allowed for flexible remote invocation of robot tests from the workstation.

The paper is organized as follows. Section II reviews the state of the art in current robot control architectures. Section III presents the architecture used in this research and describes the integration of MATLAB into the architecture. Section IV discusses the robots and the hardware used in this work, and section V presents results.

II. ROBOT CONTROL ARCHITECTURES

Robot control architectures are systems that provide structured methodologies and constraints for designing and implementing robot control systems. There are many mobile robot control architectures described in the literature. For a recent review of autonomous robot control architectures see [3]. This work primarily focuses on architectures as described in [3], rather than simple programming environments. We concentrate on process architectures that provide tools and paradigms for the design of robot control systems. Software or implementation architectures, on the other hand, may be used to provide the underlying support for systems, such as communications and access to the robot sensors and actuators, but are not the direct focus of this work.

Many of the large-scale architectures are organized around structured paradigms. For example, in [4] an architecture designed to facilitate the fault tolerant control of teams of heterogeneous robots using (among other things) a mathematical model of motivation is discussed.

In [5] a distributed fault handling and robot task management architecture using a distributed Java framework is presented. These architectures are designed to support robot control within methodological and conceptual constraints.

Behavioral robotics architectures support the implementation of control within the behavior-based robotics paradigm [6]. Behavior-based robotics incorporates elements from artificial intelligence, engineering, and cognitive sciences. Conceptually, behavior-based robot control provides methods for the parallel integration of simple behaviors to generate more complex emergent robot behaviors [1]. The Distributed SFX architecture used in this paper falls into this category as well as being a hybrid deliberative-reactive system.

There are several examples of general-purpose robot control architectures designed to allow programmers as much design flexibility as easily as possible. These are not meant to support any particular control paradigm and are designed to provide the greatest possible flexibility in robot control implementation and experimentation. For example, in [7][8] a multi-robot control architecture using MATLAB as the primary computing environment is used for various intelligent robotics control experimentation. That system is designed to allow access to all sensors and actuators (on both local and remote robots) in the form of functions that appear a standard MATLAB function calls. Unicast and broadcast communications are also supported between robots and can be used to transmit anything from a single parsable set of characters to files containing interpreted code that can be added to a currently operating robot control structure. In [9], a system designed to support distributed control in robots is presented. That system is explicitly indented to provide as few restrictions on robot control programming as possible while allowing transparent platform and language independent access to services such as actuators and sensors.

A potential drawback of very flexible robot control research architectures is that they often produce single-purpose control programs that are difficult to maintain or expand. For this reason, many structured paradigm base architectures restrict the fashion in which control can be structured. On the other hand, a potential drawback to relying on structured paradigm base architectures for primary research is that they may be too restrictive to easily support some aspects of the research design and experimentation process.

There are very few robotics research architectures that explicitly enable both paradigm-based and flexible experimental research design methodologies. One reason that such systems are not well represented is that the methodologies have a number of apparent incompatibilities. For instance autonomous robot control research based in control theoretic and engineering backgrounds tends to take a very unstructured approach to software implementation of research. Groups from Computer Science on the other hand, often focus their research on the software architecture itself, and place much more emphasis on the development of paradigms, and enforcement of good programming practices. This often

excludes or limits the agile programming and unstructured rapid prototyping design methods used in engineering research.

The goal is to support general robot control research and experimentation while providing a pathway to incorporate results from such work into a larger structured distributed robotics control architecture. The architecture discussed in this paper provides for a less structured flexible form of use of the system as a development phase and research tool, while the overall system provides a method for incorporation of the research products of the development phase into the production phase structured architecture.

The architecture is being used to control real robots in outdoor environments. As a case study, experimental results quantifying GPS and odometer errors under various test conditions are reported.

III. THE RESEARCH ARCHITECTURE

The robot control architecture is designed to accommodate autonomous distributed control of robots in outdoor environments. It is designed to support proper implementation (in the software development sense) and at the same time be useful as a rapid experimentation research tool and programming environment (common in engineering research). An overview of the architecture is presented below, along with the details of communication and integration between Java and MATLAB, followed by a discussion of the implementation on the ATRV-JR robots.

A. *Distributed SFX*

Distributed SFX is a service-based distributed robot architecture designed to support behavior based control. It is a distributed Java based descendant of the Sensor Fusion Effects architecture (SFX) [10]. The system makes use of modules to implement robot sensing and control related services. The architecture also supplies high-level service managers for the coordination and functional integration of low level modular services. Modules are exported to a distributed run-time system as services with certain attributes and types. Service may then be searched for (using a distributed-object lookup service) based on functional attributes rather than details of actual implementation or physical location. This type of architecture allows a decoupling of client and server. Clients, in an abstract sense, are interested in services with particular attributes that provide or consume certain types of data. For example, processes running on one robot might need to request perceptual data from a remote source such as another robot. In this case, a service able to supply the correct type of data would be searched for using its functional attributes (type of data, location of data source, ext.). If a service is found, an interface (proxy) can be provided to the requesting process in a modular fashion regardless of where the requested services physically resides or how it is implemented at the local level.

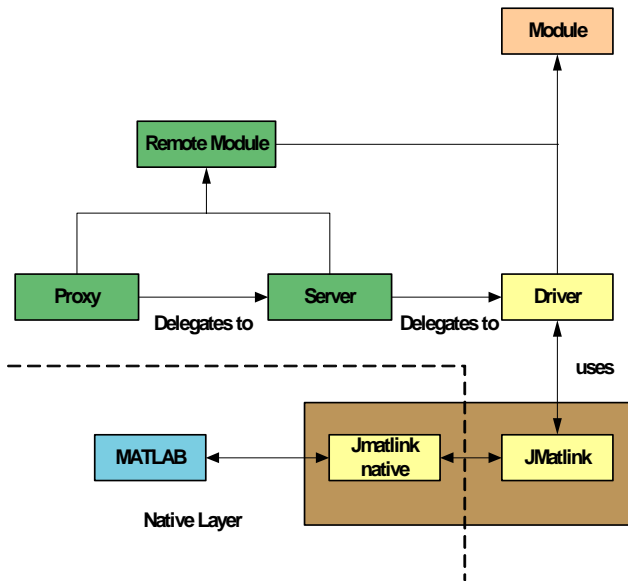


Fig. 1. Distributed SFX horizontal class hierarchy diagram. MATLAB is shown as a driver implementation module.

Heterogeneity of robot platforms is specifically supported by the architecture. This is done by representing services and components as distributed objects with consistent interfaces. Different classes of robots, such as unmanned aerial vehicles (UAV) and unmanned ground vehicles (UGV) can use a common architecture and request remote sensor and other services based on features of the required service, regardless of the actual sensor or robot providing the information.

Distributed SFX is written in Java and uses the Jini distributed service platform. Each module is roughly divided into three components- the proxy, server and driver. The proxy is the representation of the service that is transported around the network. The proxy provides the ability to move code and data, and is not merely a local representation of a remote object. The server is the representation of the service. The server deals with the distributed system and mediates between the implementation of the service (the driver) and the remote clients. The driver is the actual implementation of the service.

In a distributed and changing environment the health of the overall network can change dramatically due to communication loss, hardware failure or other causes. The use of a flexible service-oriented architecture allows for a more resilient system.

In Fig. 1 a horizontal class hierarchy diagram is shown. The figure shows the hierarchical class relationship between the distributed and native local elements of a generic behavior within the system.

B. MATLAB Support

There are a few examples of the integration of MATLAB and Java in engineering systems (see [11], [12]). However, the combination of Java and MATLAB for use in autonomous robot control is not well represented in the

literature. No major Java-based distributed robotics architectures that make use of MATLAB in its interpreted form have been reported on.

Within the larger Java/Jini based Distributed SFX framework, MATLAB is currently supported for control module design and testing and research experimentation. The JMatLink class library [2] has been used to connect Java to MATLAB. JMatLink includes methods and objects that allow Java to be used 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 of a user entering command via the MATLAB workspace command line.

A full MATLAB engine and workspace is supported. This means that MATLAB scripts and functions can be run locally on the robots as interpreted code without the need to be compiled into stand-alone executables. In addition, a full range of GUI and function based toolboxes are available for direct use on the robots. The system allows of any format of testing that is possible within MATLAB, including onboard modeling and numerical processing, but supports the integration of these developments into the larger framework in a way that is compatible with the distributed service oriented architecture.

The actual interaction between Java and MATLAB is summarized as follows: First, a thread for MATLAB is activated by its Java service implementation using the JMatLink class library method `engOpen`. This is outlined in the abbreviated Java code segment shown in Fig. 2. JMatLink is the constructor for the library and `MatlabImpl` inherits from the standard service implementation class within the Distributed SFX Java-based implementation.

```

...
import jmatlink.JMatLink;
...
private JMatLink matlink = new JMatLink();
private MatlabRunner runner;
...
public class MatlabImpl ... {
    ...
    public void activate() ... {
        ...
        engine = matlink.engOpen();
        ...
        runner = new MatlabRunner();
        runner.start();
    }
    ...
    private class MatlabRunner extends Thread {
        ...
    }
}

```

Fig. 2. Abbreviated Java code initiating a new instance of a MATLAB engine and workspace.

```

...
matlink.engPutVariable(engine, "laserData",
    laserData.readLaser);
matlink.engPutVariable(engine, "gpsData", gps.readGps);

matlink.engEvalString(engine, "sensorData.laserData =
    laserData");
matlink.engEvalString(engine, "sensorData.gpsData =
    gpsData");

matlink.engEvalString(engine, "result =
    mFunction(sensorData.");

resultData = engGetVariable(engine, "result");
....
    
```

Fig. 3. Abbreviated Java code to write data (in the form of a struct) to the MATLAB workspace and then call an interpreted MATLAB function using the data struct.

Data are written to the MATLAB workspace and a formatted field struct is constructed using the `engPutVariable` and `engEvalString` methods from the `JMatLink` class library.

Abbreviated Java code that builds a MATLAB data struct (`sensorData`) and then calls an interpreted MATLAB function (`mFunction`) with the struct as an argument is shown in Fig. 3.

The code in Fig. 3 would generally be used within a thread runner method of the `MatlabRunner` class of Fig. 2. Note that `mFunction` will be some function written in MATLAB script and `sensorData` and `result` are both MATLAB variables existing within the workspace. Results are read from the MATLAB workspace back into Java variables using the `JMatLink` method `engPutVariable`.

Fig. 4 shows the forms of support for MATLAB within the larger architecture. Development phase and production phase support are provided.

At the production level, for fully developed services, MATLAB is supported at the driver module implementation level discussed in the previous section. It can be used as the native server implementation of a service. The associated server and proxy handle the remote overhead and interaction with other services. On the left of Fig. 4 within the Distributed SFX block a native service implemented in MATLAB is shown. Note that many languages are supported for implementation of services and most services are currently implemented in Java. The purpose of the figure is to illustrate the two forms in which MATLAB is used in the system.

The MATLAB workspace on the right of Fig. 4 shows MATLAB's relationship to the larger architecture when it is being used for research and module development. This is considered the development phase. In this form, MATLAB is used as a research tool in addition to being used for standard service module development and is not strictly constrained by the overall Distributed SFX architectural paradigm.

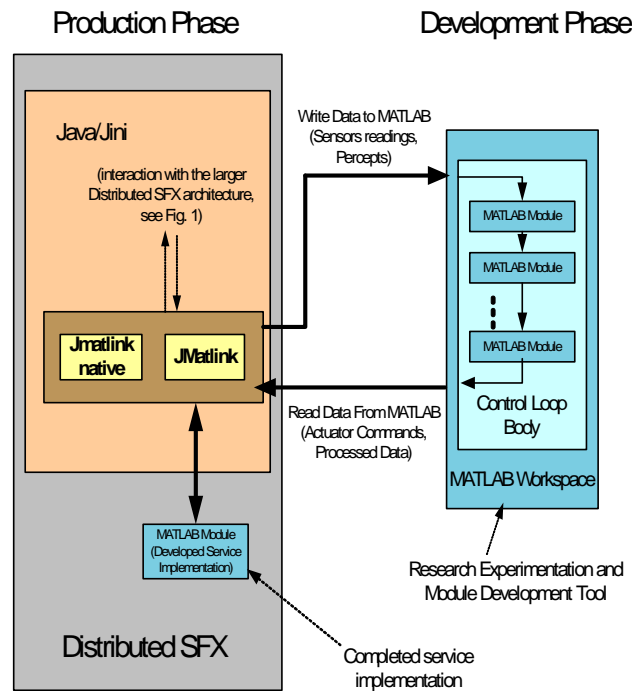


Fig. 4. Data flow diagram illustrating the relationship between MATLAB and the larger distributed Java/Jini based architecture.

For research and development, a full MATLAB workspace is supported in the development phase (Development Phase). In the production phase, fully developed modules are seen as local service implementations within Distributed SFX (Production Phase).

In summary, the architecture supports research from its primary synthesis, to its mature implementation within a structured paradigm. The architecture supports uniformity, compatibility and integration of high level AI elements and low level mathematical control-theoretic elements. In addition, the architecture supports low level research experimentation without sacrificing the integrity of the larger structured distributed robotics behavior based paradigm.

IV. ROBOTS AND HARDWARE

The Distributed SFX architecture including integrated MATLAB support has been implemented on mobile robots and computer workstations. The architecture's functionality is demonstrated using a system consisting of one robot operating outdoors and one computer workstation located inside the laboratory.

Distributed SFX has been implemented on *iRobot* ATRV-JR ground robots. These robots have been equipped with a versatile array of sensors including scanning planar lasers, FLIR heat imaging video, standard video, GPS and inertial gyroscopes. In addition, the robots have wheel odometers supplied by the manufacturer and managed by *iRobot*'s Mobility software.

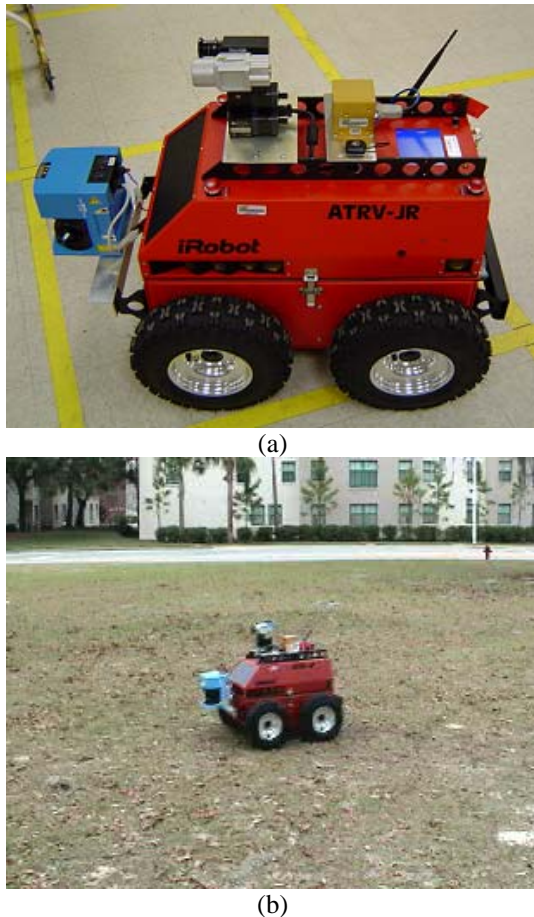


Fig. 5. An ATRV-JR robot (a) in the CRASAR robotics lab and (b) outside on the USF campus near the CRASAR robotics lab.

The ATRV-JR ground robots have PCs with 1GHz PIII processors and 2Gb ram. The onboard computers include auxiliary 10 serial port cards, firewire and UBS to accommodate sensors and other equipment. The robots were originally supplied with Linux RedHat 7.x from *iRobot* but have been upgraded to Linux 9.x. The robots are capable of running locally any software that will run on a standard PC and are accessible via wireless Ethernet (802.11g). Photos of the CRASAR ATRV-JR robots are shown in Fig. 5.

The ATRV-JR in Fig. 5 (a) is shown with a SICK scanning planar laser mounted on the front of the vehicle, a pan/tilt unit with FLIR and standard video cameras attached to the front-top equipment rack, and inertial gyroscope and GPS mounted on the center-top equipment rack.

The experiments present in this paper rely on GPS and odometry. The GPS system used is a Synergy Systems LLC M12+ with evaluation board and a HAWK GPS antenna connected via serial link to the robot's main computer. The device uses proprietary filters and error estimation algorithms. Both filtered and unfiltered readings are available to the robots.

V. RESULTS

Here we report on testing of robots in an outdoor environment using the architecture described above. Experiments to test and quantify GPS and odometer accuracy were performed in outdoor conditions. GPS, and odometer errors were measured over the course of several benchmark test driving patterns, and under varying conditions.

The tests were conducted by setting up and executing automated test sequences. The distributed architecture operated on a system including a robot in the field and a computer workstation in the lab. The experiments made use of MATLAB as supported in development mode as illustrated in Fig. 4 in section III above. The Distributed SFX architecture managed all the sensor and actuator services as distributed objects. MATLAB, was used to implement the automated test services on the robot, and to process and correlate test data within the larger Java/Jini based distributed architecture. The larger Distributed SFX architecture also managed the tests at a high level including maintenance of distributed objects and services between the lab computer workstation and the robot.

A. GPS based navigation

The example tests conducted here serve both to demonstrate the functionality of the Distributed SFX architecture and to lay the groundwork for outdoor navigation with mobile robots using GPS. In order to provide a context and motivation for the experiments, a short review of the use of GPS and odometry for positioning and navigation in mobile robots follows.

Several references can be found in the literature that discuss testing and performance of GPS receivers in mobile robots. In [13], issues concerning the use of GPS for controlling and navigating an autonomous vehicle are addressed. In that work, several experiments were conducted to measure the accuracy of GPS and the availability of satellite fix for 3D GPS. In [14] the use of a differential GPS (DGPS) is proposed for positioning in a single mobile robot. In that work, limitations and variability in accuracy as functions of number and location of satellites are studied, as well as the affects of choice of the satellite to be used and the offset of the receiver's location.

In unstructured outdoor environments, Fused GPS and wheel odometry data have been proposed for use in absolute positioning of mobile robots. For example, in [15] the authors propose a method for reducing errors in differential GPS measurements by incorporating information from robot position odometry. In that work, the authors present experimental results validating their approach. In [16] another approach for outdoor navigation using GPS data and odometry is presented. Their methodology for vehicle navigating is based on the construction of covariance matrices for automatic localization computations that include the number, and location of, visible satellites as factors. The matrices are built and maintained by sampling GPS data along a route and continuously updating the matrices to accommodate drift in GPS data quality.

In order to provide a baseline level of performance for the methods described above, and to improve the use of GPS and odometry for navigation it is beneficial to accurately measure relative and absolute GPS and odometry position error. The experiments reported on in this section provide a quantification of such errors.

B. Test Patterns

During these tests, the robot automatically followed predetermined test patterns. These included forward and backward motion along a straight line, tracing a rectangle, and tracing a circle. The controller service modules for each of these test patterns produced predetermined timed sequences of movements and recorded all sensor and actuator related data over the course of each test.

C. Data Collection and Coordinate frames

Data were recorded at each controller time step during each test-pattern driving session. Data included GPS readings, actuator commands, odometry generated position, time stamp and other sensor readings (such as Laser range values). For the test conditions used here, full control loop cycles required about 0.2 seconds to complete so data were collected at a rate of approximately 5 Hz.

In order to relate the data collected during robot driving, several coordinate frames of reference must be kept track of. These include the robot body-attached frame of reference, the robot's internal world frame of reference, and the real world frame of reference. Although these frames are technically 3 dimensional, we consider only the x and y-axis of each frame since these tests were conducted under very nearly planar conditions (mixed dirt and grass flat terrain).

The real world coordinate frame of reference is defined here to have its origin at the initial position of the robot at the start of a driving session, and to be oriented so that the positive x-axis points toward the East and the positive y-axis points toward the North. GPS data are converted to meters and related to this coordinate frame.

The robot's internal world frame is used to record and maintain position and orientation calculated from odometry. The robot internal reference frame deviates from the real world by the cumulative error generated by the odometry system.

The robot body-attached frame of reference is defined with its origin at the center of the robot and its x-axis pointed in the forward facing direction of the robot. The robot is initialized at the beginning of each driving session so that its body-attached frame is coincident with its internal world frame.

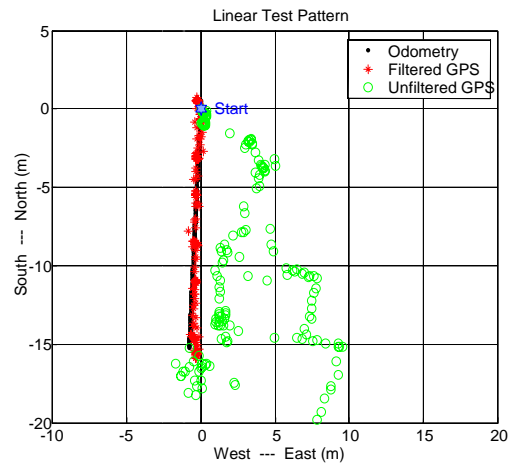


Fig. 6. GPS points and points calculated from odometry for an example linear test pattern.

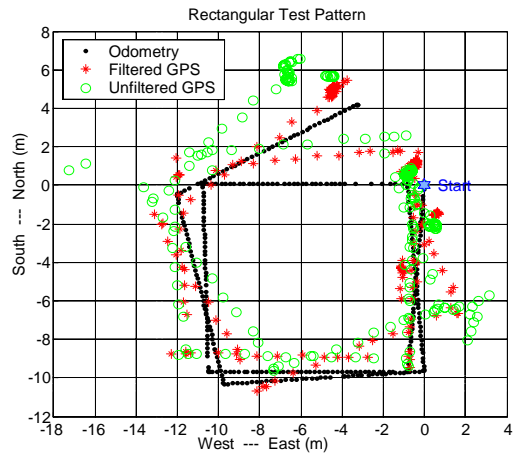


Fig. 7. GPS points and points calculated from odometry for a rectangular test pattern.

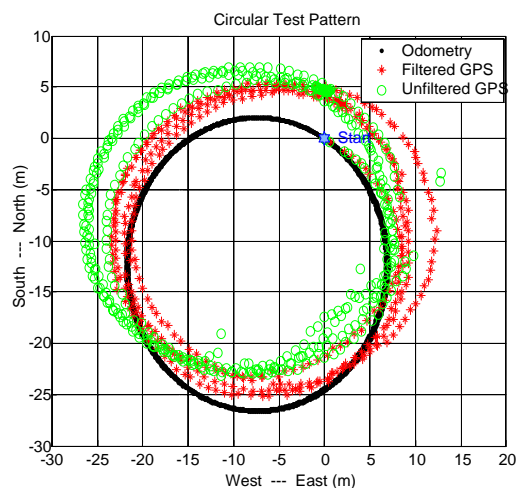


Fig. 8. GPS points and points calculated from odometry and for an example of the circular test pattern.

D. GPS and Odometry Results

For each of the test patterns, the robots were driven 5 times (for a total of 15 runs). Each test was initiated remotely from a computer workstation in the lab using the distributed architecture.

Within the MATLAB-based test service, data were collected, stored, and processed. At each iteration of control (time step), the following data were collected: GPS lock status, filtered GPS latitude and longitude coordinates, unfiltered GPS latitude and longitude coordinates, incremental x and y positions based on odometry, linear and rotational velocity based on odometry and all calculated actuator control commands. These data were used to reconstruct the robot's sensor history and internal record of position history.

Figures 6, 7 and 8 show GPS and wheel odometry position data collected on the linear, rectangular, and circular test patterns respectively. Each panel shows data collected over a single example run of each test pattern.

The paths taken by the robots during tests (as judged by GPS and odometry position measurement methods) are shown as sequences of points. All data have been converted to meters (m) and oriented as they would be in a standard map. During each run, the origin was set to be the starting point of the robot. In all three cases the positions calculate from odometry and positions measured from GPS deviate from one another over the courses of the tests. In the rectangular and circular tests, odometry positions and GPS positions deviate by approximately 2.0 meters and 6.0 meters on average respectively. This is to be expected because odometry errors are cumulative and thus influenced by the size and duration of the test patterns. The rectangular test used a square with 10 meter sides as compared to the much larger circular test pattern which used a circle with a 25 meter radius. Below, we consider absolute GPS and odometry errors.

In order to quantify GPS and odometry errors, with respect to the real world coordinate frame of reference, the robots were driven along a linear course with known intermediate points.

The cumulative linear odometry error was found to be 0.4% (1/5 of one percent) per meter traveled. Errors were averaged over 6 runs of traveling forward and back along a course of 14.8 meters over dirt and grass.

Filtered GPS position error was calculated as the average difference between each GPS reading and the robot's true position at the time of reading. The robot was driven along a known linear path similar to the one shown in Fig. 6. A total of 5 tests were conducted on 3 different days. The average GPS position error was found to be 0.91 (m) with a standard deviation of error of 0.52 (m) over all of the test runs. Note that a total of approximately 2400 data points were used to generate the average error. This actual position error is somewhat lower than, but consistent with estimated position errors reported in [13].

VI. CONCLUSION

In this paper a novel form of integration of MATLAB into a distributed behavioral robotics control architecture was presented. The architecture supports advanced robot control research both at the AI level and at the control-theoretic level. It is implemented in Java and uses Jini to manage distributed objects and services between robots and computers. The JMatLink Java library was used to incorporate MATLAB into the Distributed SFX architecture. MATLAB within the architecture and servers the dual purposes of providing a versatile robot experimentation tool, and as a rapid module and service implementation tool. Although modules and services can be implemented in many native languages, the ability to use interpreted MATLAB code is beneficial because it allows the efficient and direct transfer of research and development results generated in MATLAB into functional service implementations within the paradigm of the distributed behavioral architecture.

The architecture was used to conduct automated experiments to quantify GPS and odometry position errors in mobile robots in an outdoor environment. During the experiments all data collection, robot sensor and actuator services and higher level components were managed by the architecture as distributed objects existing on a system consisting of one robot in the field and an additional computer workstation in the lab. These results show the versatility of the architecture and lay the ground work for GPS and odometry based positioning control for the ATRV-JR mobile robot platforms used here.

ACKNOWLEDGMENTS

This work was partially supported by a grant from ONR, N 000 14-03-1-786 (2132-033-LO). L. Doitsidis was partially supported by "IRAKLITOS fellowships for research from the Technical University of Crete, EPEAEK II - 88727

REFERENCES

- [1] R. R. Murphy, *Introduction to AI Robotics*, The MIT Press, Massachusetts, 2000.
- [2] 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/>
- [3] A. D. Mali, "On the Behavior-based Architectures of Autonomous Agency," *IEEE transactions on Systems, Man and Cybernetics*, Part C: Applications and Reviews, vol. 32, no. 3, August 2002, pp. 231-242.
- [4] 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.
- [5] M. T. Long, R. R. Murphy, L. E. Parker, "Distributed multi-agent diagnosis and recovery from sensor failures," *Intelligent Robots and Systems, 2003 (IROS 2003)*,

- Proceedings of the 2003 IEEE/RSJ International Conference on*, vol. 3, Oct. 29-Nov 3, 2003, pp. 2506-2513.
- [6] M. Mataric, "Behavior-based robotics as a tool for synthesis of artificial behavior and analysis of natural behavior," *Trends in Cognitive Science*, vol. 2, no. 3, pp. 82-86, 1998.
- [7] 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.
- [8] G. J. Barlow, T. C. Henderson, A L. Nelson, E. Grant, "[Dynamic Leadership Protocol for S-nets.](#)" *Proceedings of the 2004 IEEE International Conference on Robotics and Automation (ICRA)*, Apr. 26 - May 1, 2004, New Orleans, pp. 1091-1096.
- [9] B. P. Gerkey, R. T. Vaughan, K. Stoy, A. Howard, G. S. Sukhatme, M. J. Mataric, "Most Valuable Player: A Robot Device Server for Distributed Control," *Intelligent Robots and Systems, 2003 (IROS 2003), Proceedings of the 2003 IEEE/RSJ International Conference on*, vol. 3, Oct. 29-Nov 3, 2001, pp. 1226-1231.
- [10] R. R. Murphy, R. C. Arkin, "Sfx: An Architecture For Action-oriented Sensor Fusion", *Intelligent Robots and Systems, Proceedings of the 1992 IEEE/RSJ International Conference on*, vol. 2, July 7-10, 1992, pp.1079-1086.
- [11] N. N. Okello, D. Tang, D. W. McMichael,, "TRACKER: a sensor fusion simulator for generalised tracking," *Information, Decision and Control, 1999, IDC 99, Proceedings*, 1999, Feb. 8-10,1999, pp. 359-364.
- [12] J. Contreras, A. Losi, M. Russo, "JAVA/MATLAB simulator for power exchange markets", *Power Industry Computer Applications, 2001, PICA 2001. Innovative Computing for Power - Electric Energy Meets the Market. 22nd IEEE Power Engineering Society International Conference on*, May 20-24, 2001, pp. 106-111.
- [13] S. Panzieri, F. Pascucci, G. Ulivi, "An Outdoor Navigation System Using GPS and Inertial Platform", *IEEE/ASME Transactions on Mechatronics*, vol. 7, no. 2, pp. 134-142, June 2002.
- [14] R. Willgoss, V. Rosenfeld, J. Billingsley, "High precision GPS guidance of mobile robots," *Proceedings of the Australasian Conference in Robotics and Automation*, 2003.
- [15] 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," *Robotics and Automation, 2003 (ICRA), Proceedings of the IEEE International Conference on*, vol. 2, Sept. 2003, pp. 1978-1984.
- [16] R. Thrapp, C. Westbrook, D. Subramanian, "Robust Localization algorithms for an autonomous campus tour guide," *Robotics and Automation, 2001 (ICRA), Proceedings of the IEEE International Conference on*, vol. 2, 2001, pp. 2065-2071.
- [17] L. Doitsidis, K. P. Valavanis, N. C. Tsourveloudis, Fuzzy logic based autonomous skid steering vehicle navigation, *Robotics and Automation, Proceedings of the 2002 IEEE International Conference on Robotics and Automation (ICRA '02)*, vol. 2, 11-15 May 2002, pp. 2171 - 2177.