Tactical Mobile Robot Mission Specification and Execution

Ronald C. Arkin
Thomas R. Collins
Yoichiro Endo

Mobile Robot Laboratory, College of Computing
Georgia Institute of Technology, Atlanta, Georgia, 30332
{arkin,tc3,endo}@cc.gatech.edu

Abstract

Georgia Tech, as part of DARPA's Tactical Mobile Robotics (TMR) Program, is developing a wide range of mission specification capabilities for the urban warfighter. These include the development of a range of easily configurable mission-specific robot behaviors suitable for various battlefield and special forces scenarios; communications planning and configuration capabilities for small teams of robots acting in a coordinated manner; interactive graphical visual programming environments for mission specification; and real-time analysis tools and methods for mission execution verification. This paper provides an overview of the approach being taken by the Georgia Tech/Honeywell team and presents a range of preliminary results for a variety of missions in both simulation and on actual robots.

1. Introduction and Overview

As part of DARPA's Tactical Mobile Robotics Program, Georgia Tech is providing certain basic capabilities suitable for robotic missions in urban settings: flexible reactive behaviors suitable for specific urban warfare and information gathering missions; a usability-tested mission specification system for rapid development of mission scenarios; and real-time analysis capability (in conjunction with Honeywell Technology Center).

This paper focuses on the behavioral and mission specification aspects of our program. Schema-based behavioral control[1] in the context of a usability-tested mission specification system[7, 8] provides the framework for this research.

Mission development in the field proceeds as depicted in Figure 1. An operator develops a mission using the *MissionLab* mission specification system. It is compiled through a series of languages that bind it to a particular robot and software architecture (for this work a Pioneer AT robot or Urbie) (Fig. 2). It is then tested in a faster than real-time simulation before downloading to the actual robots for execution. After deployment the console serves as a monitor and control interface for the robot during the mission, permitting rapid intervention if needed by the operator.

This paper first describes the software architecture for this project. Hardware specifications are then provided, followed by preliminary results for both simulation and laboratory experiments. A summary concludes the paper.

2. Software Architecture

Figure 3 depicts the overall system architecture being developed for this effort. It contains 3 major subsystems: Executive, Premission, and Runtime. The executive subsystem is the major focus for operator interaction. It provides an interface to both the runtime simulators and actual robot controllers, as well as the premission specification facilities and the physical operator groundstation itself. The premission subsystem's role is to provide an easy-to-use interface for designing robot missions and a means for

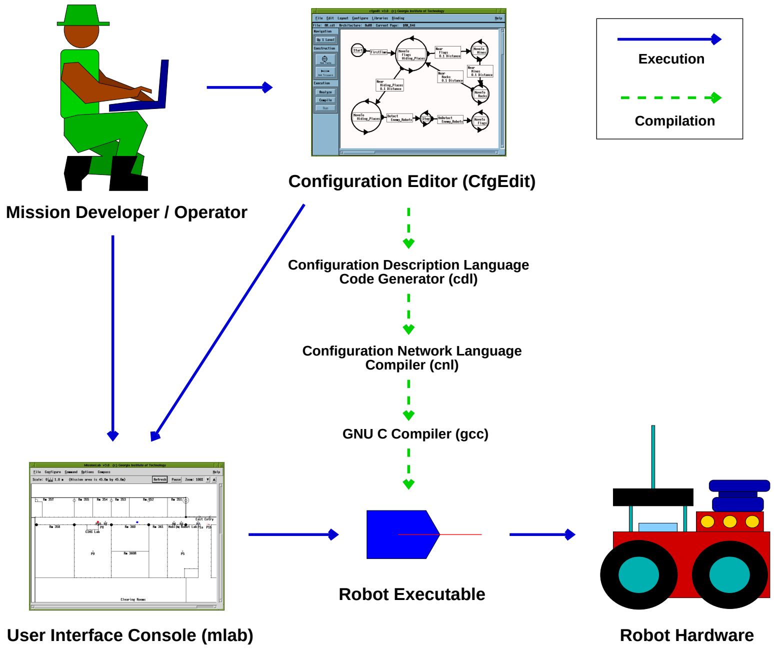

Figure 1: Deployment Concept

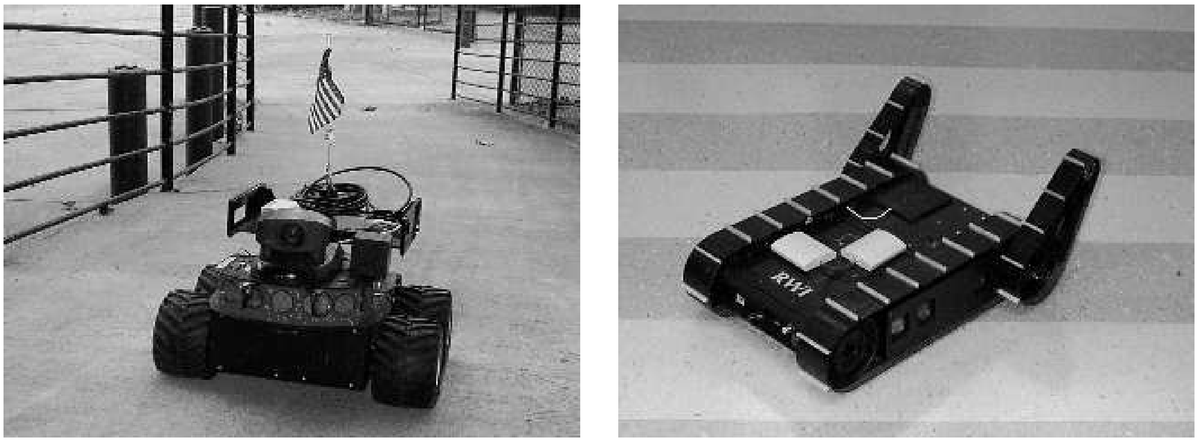

Figure 2: (Left) Pioneer AT (Right) Urbie Robot

evaluating overall usability. The runtime control system, which is located on each active robot, provides the execution framework for enacting reactive behaviors, acquiring sensor data and reporting back to the executive subsystem to provide situational awareness to the team commander. Additionally, a separate support system is provided for interprocess communications.

In Figure 3, typical communication paths between components are shown. Wherever separate threads of execution exist, this communication is implemented with IPT[5]. In other cases, communication may take the form of dedicated point-to-point links or conventional parameter-passing during the invocation

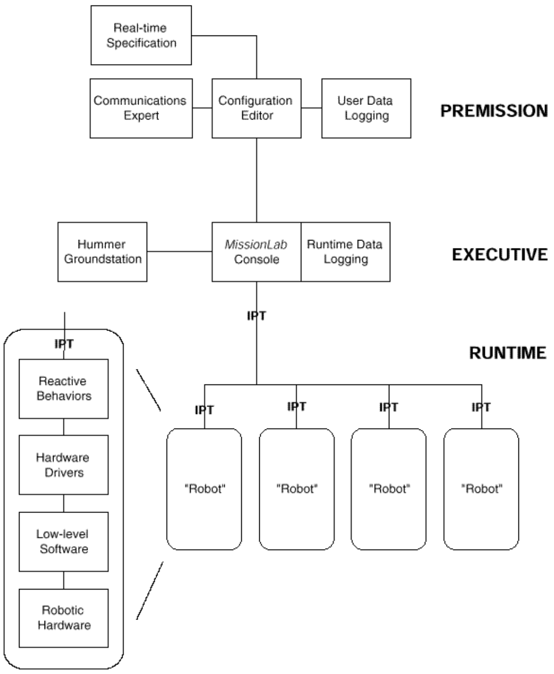

Figure 3: System Architecture

of processes. The figure shows a "robot" as the combination of reactive behaviors, appropriate hardware drivers, both actuator-specific and sensor-specific low-level software, and the robot hardware itself. This assemblage of components provides a uniform, hardware-independent interface to the executive subsystem which is equally suitable for simulated robots. The runtime system consists of one or more instances of these assemblages, with four shown in this particular case, corresponding to the robots available for the project.

The remainder of this section provides functional specifications for each of the major system components.

2.1 Executive Subsystem

The executive subsystem consists of the *MissionLab* console, faster-than-real-time simulator, and runtime data logging components.

2.1.1 *MissionLab* Console

The *MissionLab* console (mlab) (Figure 4) serves as the central interface for the execution of a mission. The mlab program presents results of either simulations or actual robotic missions directly to the operator. It requires the use of the interprocess communications subsystem (IPT) to maintain contact with the robots and other active processes. The *MissionLab* console provides the following capabilities:

- Loads precompiled robot control programs and overlay description files
- Configures the display
  - generating obstacles for simulations
  - altering the scale of the display
  - changing the virtual time for simulations

- scaling the size of the display (zooming)

- Provides a command interface that permits interactive step-by-step command issuance by the operator using CMDL, a structured English language that
  - has the ability to execute, stop, pause, restart, rewind, single step, and abort missions during execution
  - has the ability to use team teleautonomy by directing robots to particular regions of interest or by altering their societal personality (Figure 5).
- Provides display options
  - leave trails where the robots have been
  - highlight obstacles that affect the robot
  - show instantaneous directional reaction of robot to its environment

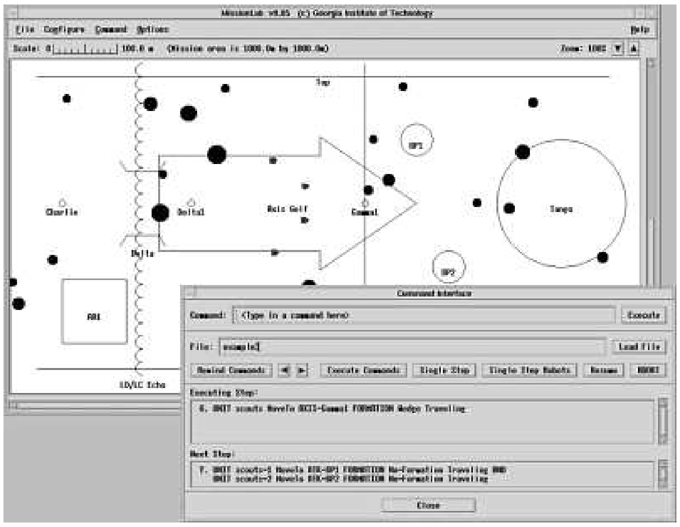

Figure 4: *MissionLab* Console (mlab)

.

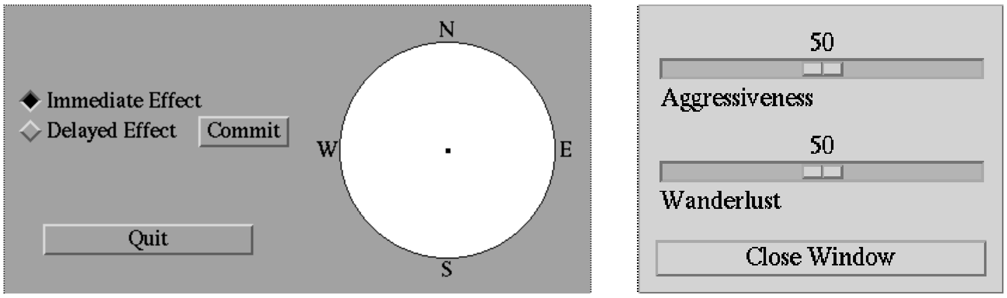

Figure 5: Teleautonomous operation in *MissionLab*. Dialog boxes allow operator to specify direction and "personality".

The *MissionLab* console also provides a display (Figure 6) that shows either: (1) the output of a simulated robotic mission that is run faster than real-time and can serve to determine whether or not a premission specification has been successfully completed; or (2) An operator mission display screen where robots in the field report back their position and relevant mission data that is shown on the mlab display to provide situational awareness and context for higher level decisions regarding aborting, continuing, or biasing the mission in various ways.

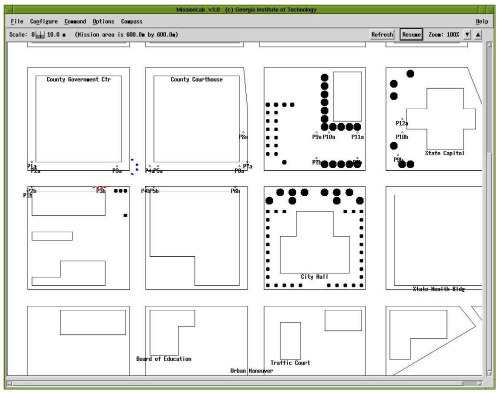

Figure 6: Display during execution of simulated mission (display of actual robot mission is similar).

2.1.2 Runtime Data Logging

The executive subsystem also includes a runtime data logging capability that is used to provide a means to evaluate the performance and effectiveness of a mission. This may include measures regarding the risks that the robots undertook during the mission, other related safety factors, time and distance to completion, etc.

2.2 Premission Subsystem

The premission subsystem involves the specification, creation, and construction of behavior-based robots suitable for specific missions. It provides a user-friendly graphical programming environment and a series of language compilers used to transform the high-level iconic description into executable code suitable for the executive subsystem. In addition, it provides data logging tools that are geared for usability studies leading to the enhancement of the user interface.

2.2.1 Configuration Editor

The configuration editor (cfgedit) provides a visual programming entry point into the system (Figure 7). It is geared to average end-users and requires limited training to use. The interactive iconic interface generates configuration description language (CDL) code which, when compiled and bound to a particular architecture and robots, generates a meta-language. In this project this is CNL, the configuration network language, that serves as a precursor to the C++ code that is ultimately generated when the CNL code is compiled. This resulting C++ code forms the executable code for the robot controller itself. Within the executive subsystem, this code is then directed either to the simulation or the actual robots for execution.

2.2.2 Usability Data Logging

Additional software is used to record user actions during premission planning. This includes data such as the number and type of keystrokes and mouse clicks, time to create certain objects, and other relevant data. These data are then used to interpret the skill by which a user is capable of achieving within the system, and after subsequent usability analysis, is used to refine the design interface itself. It is a support tool geared for formal usability studies.

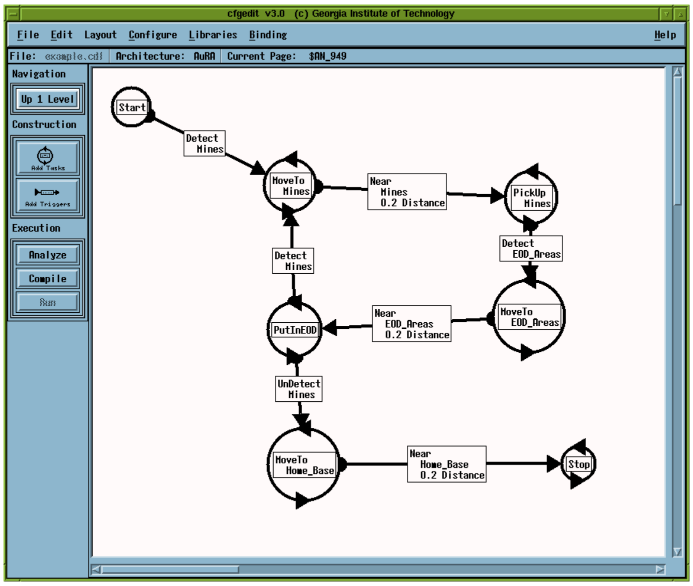

Figure 7: Graphical configuration using cfgedit.

2.3 Runtime Subsystems (1 per robot)

The runtime control code created by the premission subsystem and then tested in simulation within the executive subsystem is then sent to the actual robotic systems for execution. Thus there is one runtime subsystem located on each robot required for the mission. IPT provides interprocess communication between the robots and the mlab console. The runtime system consists of a set of reactive behaviors and sensor strategies to interpret and react to the world; hardware drivers customized to interface designated robots to the *MissionLab* system; low-level robot control code generally provided by the robot manufacturer; and the actual robotic and sensor hardware.

2.3.1 Reactive Behaviors

A collection of reactive behaviors is compiled and downloaded to each robot for execution of the mission. These reactive behaviors embody the mission specification designed within cfgedit. They process sensor data as rapidly as possible and issue commands to the lower level software for timely execution on the robot. These behaviors include activities such as obstacle avoidance, waypoint following, moving towards goals, avoiding enemies, and seeking hiding places, all cast into mission-specific reusable assemblages. Action-oriented perceptual code already supports both Newton Labs Cognachrome real-time color vision systems and ultrasonic data. Team behaviors, such as team teleautonomy, formation maintenance, and bounding overwatch are also bundled for execution as necessary. The output of these behaviors is sent to the groundstation for monitoring purposes as well as to the robot for execution.

2.3.2 Hardware Drivers

In order for *MissionLab* to be able to build an executable program to run on a given robot, it requires an ensemble of routines to set up, control, and receive feedback from the actual (or simulated) robot. Some variation in capabilities appears among the various robots that are supported, but the expected set of routines for the TMR platforms (Pioneer AT and Urbie) include:

- Movement commands: move (direct to robot to go to another position); drive (direct the robot to maintain a velocity); turn (rotational equivalent of move i.e., go to another orientation); steer (rotational equivalent of drive , i.e., change angle at constant rate); stop (stop all motion); stopdrive (stop translational motors); stopsteer (stop rotational motors).

- Range measurement commands: range_start (turn on ranging sensors); range_stop (turn off ranging sensors); range_read (take range readings).
- Position feedback commands: getxy (get current position in defined coordinate system); setxy (set the defined coordinate system, i.e., establish origin); initxy (initialize position sensors).
- System monitoring commands: wait_for_drive_to_stop (block further activity while translational motors are active); wait_for_steer_to_stop (block further activity while rotational motors are active); drivestat (provide translational motor status); steerstat (provide rotational motor status);
- Initialization and termination: open_robot (initialize robot and establish connection as required); close_robot (terminate robot and relinquish connection as required).

Additional drivers are required for sensors which are not tightly integrated into the onboard robot control system. These includes such vision-related capabilities as specifying the characteristics of a target and requesting target tracking status (and position, if available).

2.3.3 Low-level Software

Low-level software includes embedded software and firmware that is typically provided by the vendors of robots and sensors in order to access the basic capabilities of those devices. For this project, this classification includes PSOS, running on the robot controller, and ARC, running on the vision system. The onboard microcontroller of the Pioneer robot is equipped with the Pioneer Server Operating System (PSOS) software. PSOS serves the serial communication port provided for the receipt of commands and the return of status information. As such, most of the functions listed in the previous section for the robot driver result in the transmission of a message to PSOS, which in turn handles the request. The Cognachrome vision system behaves similarly, with its serial port served by an embedded operating system called ARC. This multitasking system allows the vision tracking parameters to be changed and issues tracking results at specified intervals. ARC provides a C development environment and runtime system for the generation and support of vision programs that exceed the basic capabilities provided with the Cognachrome system.

3. Hardware Architecture

Like other reactive or hybrid software architectures, the Autonomous Robot Architecture[2] (AuRA) used for this research runs most seamlessly on a hardware architecture which supports the abstraction of the hardware at a layer corresponding to logical sensors and logical effectors[4, 6]. In such a hardware context, there are well-defined processes occurring on either side of equally well-defined interfaces. In the case of effectors, these processes generally conform to most common notions of servo control, either open-loop or closed-loop. The corresponding sensor processes can be highly complex, including real-time image processing at frame rates. A less obvious characteristic of these logical sensors and effectors is that under ideal conditions, their processing loops are sufficiently fast so that they can be treated as a small, perhaps negligible, latencies. Although this may seem to be a severe requirement, it allows the reactive behaviors to be formulated with minimal consideration of feedback control issues that cross boundaries between the low-level embedded processors and the reactive hardware architecture itself. One way of summarizing this design philosophy is to spare no expense in the design or acquisition of embedded processor subsystems that implement sensor or actuator functionality, because it pays large dividends in the integration process.

3.1 Robot platform and interface

Whenever it is possible to use existing commercial off-the-shelf (COTS) mobile robotic platforms, the interface to that platform becomes the layer boundary described above. Typical COTS platforms include servo control of locomotion with fully-integrated odometry, as well as some basic mobility sensors. In contrast to mission sensors, which perform mission-specific functions, mobility sensors provide the basic perceptual capability needed to perform the simplest locomotion tasks in the presence of an unstructured environment. Mobility sensors are sometimes considered to be a subset of organic sensors, which include all platform-specific sensors such as those required to maintain knowledge of platform status and health (power, temperature, etc.).

For the TMR systems, the Pioneer-AT mobile robot platform manufactured by ActivMedia (Peterborough, New Hampshire) was selected because of its compliance with stated locomotion requirements. The Pioneer-AT has 4 driven wheels with wide, compliant tires suitable for traversal of mixed terrain with small obstacles (two- to three-inch obstacles and slightly larger "negative" obstacles). The platform includes wheel encoders and a front-facing semi-circle of ultrasonic sensors. All of this functionality is accessed via the platform's embedded processor, a 68HC11 running PSOS, a proprietary operating system. The external interface to PSOS is implemented on an RS-232 serial channel.

Just as in previous instances of interfacing new robot hardware platforms to AuRA, it was necessary to develop a library of interface routines that implement standard AuRA primitives on the new platform. In this case, this amounts to sending the defined serial commands and parsing the PSOS responses. Whereas on some platforms there is nearly a one-to-one mapping of AuRA primitives with platform-supplied commands or responses, for the Pioneer it is necessary to implement a server process to maintain certain derived variables. This arises for two reasons: 1) several different and unrelated sensor values are returned at the same time in a single data "packet" and must be saved for possible requests from AuRA, and 2) the odometry calculations performed within PSOS are truncated at small distances and must be augmented at sensor data rates. In effect, although this server process runs within AuRA (not on the embedded processor), it essentially performs an embedded function and allows the assumption described above to remain valid: no complex, high data-rate computations are performed within the true reactive layer of the architecture. From both a hardware and software architectural standpoint, this represents a departure from previous ports of AuRA and *MissionLab*, since there is a high-rate processing loop running on the same processor that implements the robot "executable," which in prior implementations only responded at "reactive" rates.

The complete hardware architecture is shown in Figure 8. The robot executable program can either run on a remote host (normally the same machine that implements the operator console) or on a platform carried onboard the robot. The latter option provides the greatest flexibility, including a truly autonomous capability even in the the absence of a datalink with the console.

3.2 Sensor subsystems

The Pioneer-AT platforms were preconfigured with the Newton Research Labs Cognachrome vision system (Renton, Washington), which effectively acts as a logical vision sensor capable of colored blob detection. For our TMR purposes, blob detection is a placeholder for more-sophisticated image processing functions, such as stereoscopic or omnidirectional vision that are being developed concurrently by other contractors and COTS vendors. Like the platform itself, the vision system provides a serial interface with a defined command/response protocol.

The front-facing semi-circle of ultrasonic sensors was not optimal for the application. Although the robot is capable of turning very nearly in place and can thus act as a near-holonomic platform, it is not equipped with a 360-degree arrangement of any obstacle detection sensor that can provide a uniform

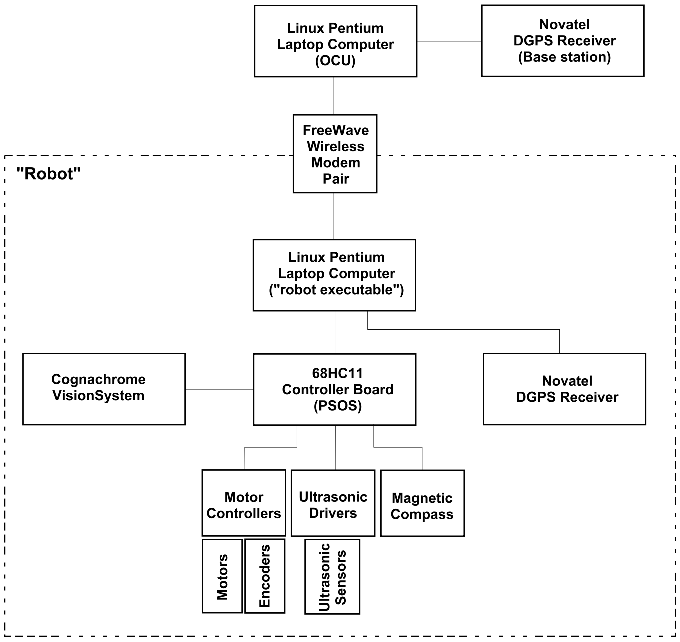

Figure 8: Hardware Architecture

representation of obstacles in all directions. In our reactive architecture, such a sensor arrangement produces a composite steering vector which, in conjunction with a holonomic platform, generally allows for agile locomotion in an obstacle-strewn environment. In order to achieve this with the Pioneer-AT, most of the eight ultrasonic sensors were physically relocated, producing a sparse, yet equally-spaced, circular arrangement. This hardware-oriented approach to the problem was pursued in parallel with a alternative software approach that would work with the original sensor configuration. This software approach implemented, in effect, virtual ultrasonic sensors by allowing recent readings to persist in spite of platform rotation and minor amounts of translation.

The only other hardware customization of the robotic platform was the addition of a differential GPS (DGPS) receiver to facilitate precision outdoor navigation between specific waypoints. DGPS receivers represent a mature technology with established standard interfaces, so the addition of this capability was largely an integration process. Of course, knowing one's position accurately is of limited usefulness if orientation is not accurately known as well. Without accurate orientation knowledge, the robot would generally head in a random direction and not make immediate progress toward its current waypoint until a course correction is made. This is significant, because the Pioneer-AT has only two built-in orientation sensors, and neither is very accurate. The first sensor is odometry, which is highly dependent on the ground surface and quickly generates cumulative errors. The second sensor is an optional magnetic compass, which is only suitable in ideal environments free from variations in the local geomagnetic field. While it is possible to use relatively low-cost rate gyros to recover orientation accurately over the duration of short missions[3], we desired a solution that required no additional weight, volume, power, or integration effort. This brings us back to the rather awkward prospect of selecting a heading based on a "best guess" and correcting it as soon as possible based on DGPS position updates. If this were done at the reactive level, the "false starts" would be somewhat lengthy and noticeable. But by adhering to the basic principle of allowing maximal processing within the logical sensor, a better solution is achieved:

the DGPS receiver can in fact compute highly accurate velocities even after short translations, so the velocity heading (as returned by the receiver itself in standard message protocols) is actually a satisfactory orientation representation. Even with no reasonable guess of the current heading, false starts are typically only about one foot long before an accurate course correction is made. Then, if odometry is only used between DGPS updates, subsequent course directions are seldom large enough to be noticed.

3.3 Wireless communication

Although a robot with an onboard laptop computer is capable of fully autonomous operation without communication with the console, there are many uses for a wireless datalink. These include the display of robot status, teleoperation or teleautonomous control, communication between robots, and behavioral reconfiguration during a mission. The TMR configuration includes a FreeWave wireless serial modem (Boulder, Colorado) that can either be used as a peripheral of the onboard computer or as a replacement for the onboard computer (when the robot executable runs on the console itself). We have achieved consistently reliable communication between several floors of research buildings and also between buildings on campus. Under ideal line-of-sight conditions outdoors, it is possible to maintain a link of over 20 miles, but we have rarely tested anything more than a few tenths of a mile.

In order to implement the IPT communication protocol over the wireless serial links, it is necessary to run PPP (Point-to-Point Protocol) to produce a TCP/IP network layer. This adds an additional level of complexity to the system that would not be required with wireless Ethernet hardware, but at this time the range and reliability of the FreeWave modems appears to justify this choice. With increasing improvements in the bandwidth and robustness of wireless Ethernet, we expect to utilize these devices in the future.

3.4 Future platform

Work has already begun on developing an interface to the Urban Robot, or "Urbie," built by the RWI division of IS Robotics (Jaffrey, New Hampshire). This platform will be able to traverse rougher terrain and even climb steps, using its articulated treads. It includes an onboard computer running Linux, so there will be no need to use an onboard laptop, but vision sensing is limited due to the lack of a system like the Cognachrome. Ultrasonic sensor placement is somewhat improved over the default Pioneer-AT configuration, but still sparse. The interface to the Urbie is architecturally similar to the approach taken with the Pioneer AT, but it ties into the onboard controller at a higher level of abstraction, taking advantage of the API provided by RWI's Mobility software architecture. This approach utilizes CORBA to provide a uniform interface across different robot platforms. If it is sufficiently flexible and robust, it may minimize future efforts in porting *MissionLab* to other platforms that comply with the Mobility architecture.

4. Preliminary Results

Typical tasks for the TMR program involve crisis assessments of remote areas that are occupied by enemy forces and that cannot be approached directly by US personnel or intelligence assets. For example, inspecting biological/chemical contamination levels of rooms in a target building that is under insurgent control or monitoring hostile activities at denied areas in an airfield seized by enemy forces are among those scenarios. In order to prepare for such large scale missions, simulations for the urban warfare scenarios have been developed, and an in-lab experiment using the Pioneer-AT robot has been conducted.

4.1 Simulations

In order to simulate the TMR missions with *MissionLab*, four cases of typical urban warfare scenarios were determined. The followings are descriptions of those cases:

- **Case-1** [Urban Maneuver]: Two teams of robots maneuver across an urban area, approaching a target building by overwatching each other. Each team takes paths close to the nearest walls to cover from crossfire while maintaining their formation. Downtown Atlanta was chosen as this site, and the state capitol was selected as the target building.
- **Case-2** [Room Clearing]: A pair of robots clears rooms in a building. When one robot is inspecting the interior of a room, the other robot provides cover at the entrance of the room. The third floor of Manufacturing Research Center (MaRC) at Georgia Tech was chosen as this demo site.
- **Case-3** [Stealth Maneuver]: A robot attempts to approach a target building while it avoids being detected by enemy forces. After being deployed from the woods, the robot maneuvers across a parking lot by using parked cars for cover, then crosses a street when it is determined to be clear from enemy forces.
- **Case-4** [Double-phase Approach]: Multiple robots, deployed from different locations, approach and inspect denied areas in an airfield, such as the control tower, TV station, and hangar. Two phases are involved in the approach of an aircraft hangar, which is located in an area that can be approached only by a tiny robot. First, a larger robot, carrying the very small robot, traverses overland to get close to the hangar; then in the second phase, the tiny robot, is launched and probes inside the hangar.

Each of the four cases was successfully simulated with *MissionLab*. The outputs of the missions displayed on the *MissionLab* console are shown in Figure 9 for all four cases. The portions that contain simulated robots in are enlarged in the figure.

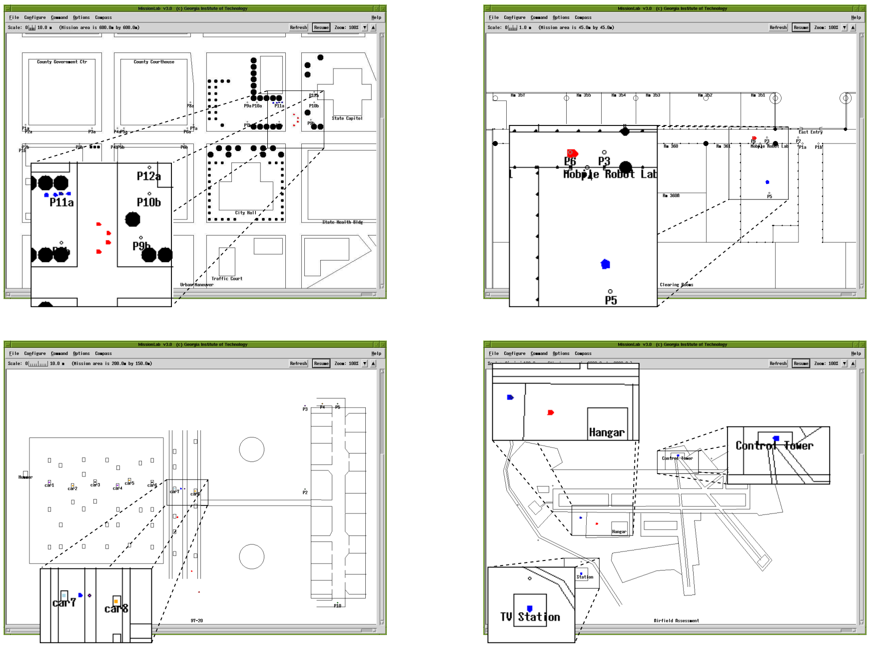

Figure 9: Simulation outputs with partially enlarged images - **Case-1** [Urban Maneuver] (top left), **Case-2** [Room Clearing] (top right), **Case-3** [Stealth Maneuver] (bottom left), and **Case-4** [Double-phase Approach] (bottom right).

4.2 Robot Experiments

In order to test the capability of MissionLab executing real robots, an in-lab experiment was conducted at Manufacturing Research Center (MaRC). The objective of this mission was to deploy a Pioneer-

AT robot from a room, maneuver the robot through a corridor, and inspect a room located two doors down the hall to check for the existence of a specified object (an enemy). Information known prior to the experiment included a floor plan of the building (Figure 10) and the color of the target object: red. In this experiment, the robot was configured for interior use: no DGPS was on-board.

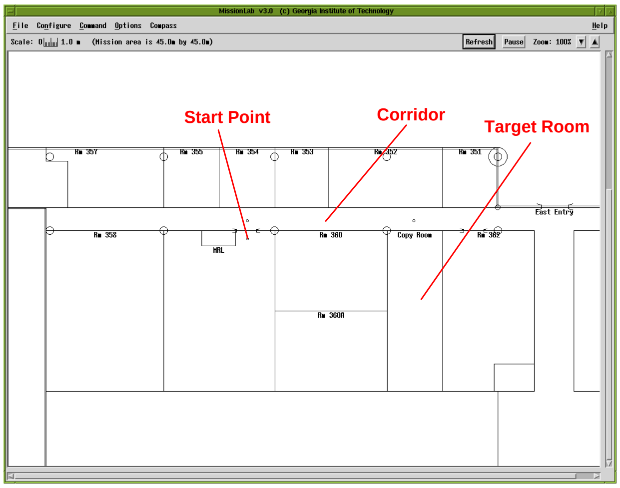

Figure 10: MaRC Floor Plan

During the premission phase, the remote-room-inspection mission was constructed with the configuration editor. As shown in Figure 11, it was assembled with seven finite states and and six perceptual triggers. The behavioral states and triggers utilized by the mission are:

- **Start** [state]: The robot is initialized.
- **GoTo** [state]: The robot moves to a location specified in global coordinates.
- **CorridorFollowing** [state]: The robot moves to a goal while attempting to stay on a path within a corridor which is specified as two points: "start" and "end".
- **Stop** [state]: The robot stops moving.
- **FirstTime** [trigger]: The transition occurs immediately unconditionally.
- **MovedDistance** [trigger]: The transition occurs when the robot has moved a desired distance.
- **AtDoorwayUltrasound** [trigger]: The transition occurs when the robot detects a doorway or hallway.
- **Detect** [trigger]:The transition occurs when the robot detects the specified object.

After being initialized by the **Start** state, the **GoTo** state brings the robot out of the room, until it moves 1.25 meters. The **CorridorFollowing** state then maneuvers the robot through the corridor. After the robot follows the corridor for about 7 meters, it starts seeking an opening in the righthand side of the wall using sonar, which is the entry door for the target room. When it is at the door, the robot moves into the room in another **GoTo** state, unless it detects the red object. If the red object is seen, the robot stops entering the room.

The pictures in Figure 12 and Figure 13 show the remote-room-inspection room as executed by the robot. The trial in Figure 12 was configured so there is no target object in the room. For the second trial, a red object was present (Figure 13). Both results are shown in Figure 14.

As seen in the Figures 12-14, the robot successfully accomplished this simple remote-room-inspection mission for both with-colored-object and without-colored-object cases. In this experiment, it was found that there were some discrepancies between the traces of the robots displayed in the MissionLab console

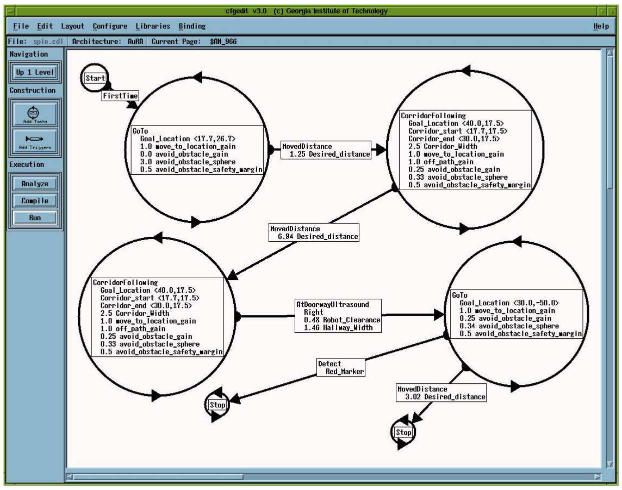

Figure 11: The Finite State Acceptor diagram of the mission

Figure 12: The Remote-Room-Inspection Mission (Trial-1): the robot entered the room since the red object was not present in the room.

Figure 13: The Remote-Room-Inspection Mission (Trial-2): the robot stopped entering the room because the red object was detected.

(Figures 14) and the actual paths the robots took. For example, even though for the without-colored-object case the robot entered the target room through the door, the output in the MissionLab console shows that as if the robots entered the room about 1 meter before it reached to the door. These display errors were assumed to be a result of both the map not being drawn to exact scale and dead-reckoning error. The robot, however, easily overcame these errors by detecting the door opening of the target room directly with a perceptual sensor (sonar) rather than using dead-reckoning with the shaft-encoders. Cfgedit allows such changeover of states and/or triggers very easily, permitting operators to create missions to fit environmental requirements rapidly.

5. Summary

A flexible software architecture embodied within the *MissionLab* software system has been described. It features behavioral control, rapid reconfiguration, and a visual programming environment. Its current hardware implementation in the context of the Tactical Mobile Robotics Program Testbed as fielded on Pioneer AT robots was also presented. Simulation results for a range of potential military missions

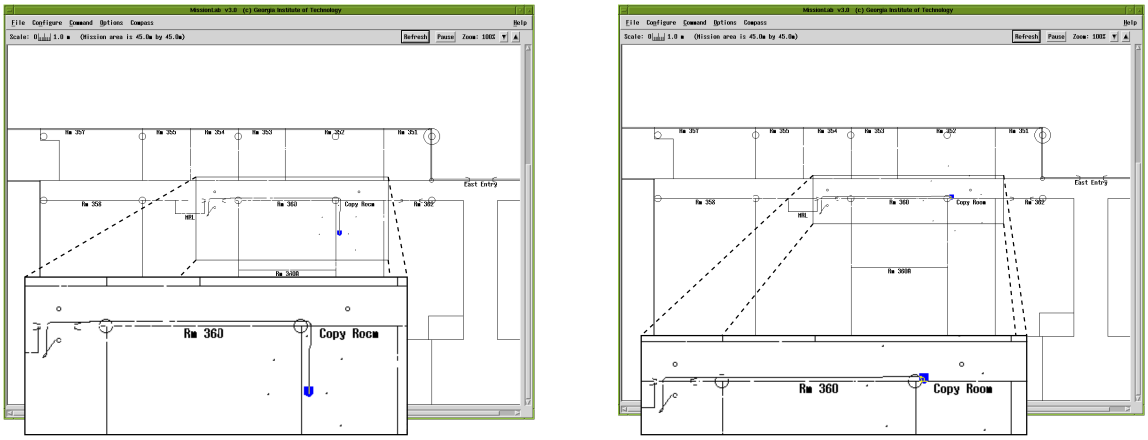

Figure 14: Outputs of the Remote-Room-Inspection Mission: without (left) and with (right) the red object in the target room. (Images partially enlarged.)

were described as well as a simple laboratory experiment on an actual robot. These results are being further validated at the TMR testsite at Fort Sam Houston in San Antonio, Texas for a range of relevant operational scenarios.

Acknowledgments

This research is funded by DARPA under contract #DAAE07-98-C-L038. The authors would like to thank Michael Cramer, Jonathan Diaz, Sunuondo Ghosh, Karen Haigh, William Halliburton, David Johnson, David Musliner, Kristy Robeson, and Donghua Xu for their participation on this project.

References

1. Arkin, R.C., "Motor Schema-Based Mobile Robot Navigation", *International Journal of Robotics Research*, Vol. 8, No. 4, August 1989, pp. 92-112.
2. Arkin, R.C. and Balch, T., "AuRA: Principles and Practice in Review", *Journal of Experimental and Theoretical Artificial Intelligence*, Vol. 9, No. 2-3, April-Sept. 1997, pp. 175-189.
3. Borenstein, J.; Feng, L., "Gyrodometry: a new method for combining data from gyros and odometry in mobile robots", *Proceedings 1996 IEEE International Conference on Robotics and Automation* , Vol. 1, pp 423-428.
4. Collins, T. R., Arkin, R. C., and Henshaw, A. M. "Integration of Reactive Navigation with a Flexible Parallel Hardware Architecture", *Proceedings 1993 IEEE International Conference on Robotics and Automation* , Vol. 1, pp. 271-276.
5. Gowdy, J., "IPT: AN Object Oriented Toolkip for Interprocess Communication", Tech Report CMU-RI-96-07, Robotics Institute, Carnegie Mellon University, March, 1996.
6. Henderson, T. C., Shilcrat, E. "Logical Sensor Systems", *Journal of Robotic Systems*, Vol. 1, No. 2, March 1984, pp. 169-193.
7. MacKenzie, D. and Arkin, R., "Evaluating the Usability of Robot Programming Toolsets", *International Journal of Robotics Research*, Vol. 4, No. 7, April 1998, pp. 381-401.
8. MacKenzie, D., Arkin, R.C., and Cameron, R., "Multiagent Mission Specification and Execution", *Autonomous Robots*, Vol. 4, No. 1, Jan. 1997, pp. 29-52.