Real-time Cooperative Behavior for Tactical Mobile Robot Teams

Subsystems Specification / A002

October 1998

Prepared by:
Georgia Tech College of Computing and
Georgia Tech Research Institute

Georgia Institute of Technology
Atlanta, Georgia 30332

Prepared for:
TACOM
Warren, Michigan 48397-5000
Contract No. DAAE07-98-C-L038

A UNIT OF THE UNIVERSITY SYSTEM OF GEORGIA
ATLANTA, GEORGIA 30332

Georgia Institute of Technology

1998

1 INTRODUCTION

This document constitutes the subsystem specifications and is submitted to TACOM as deliverable A002. It contains a description of the overall system architecture to be used within Georgia Tech's research for the DARPA Tactical Mobile Robotics Program. It further provides functional specifications for each of the major system components. More specific details regarding interfaces will be provided in future interface control documents to be submitted according to the prescribed deliverables schedule of the contract.

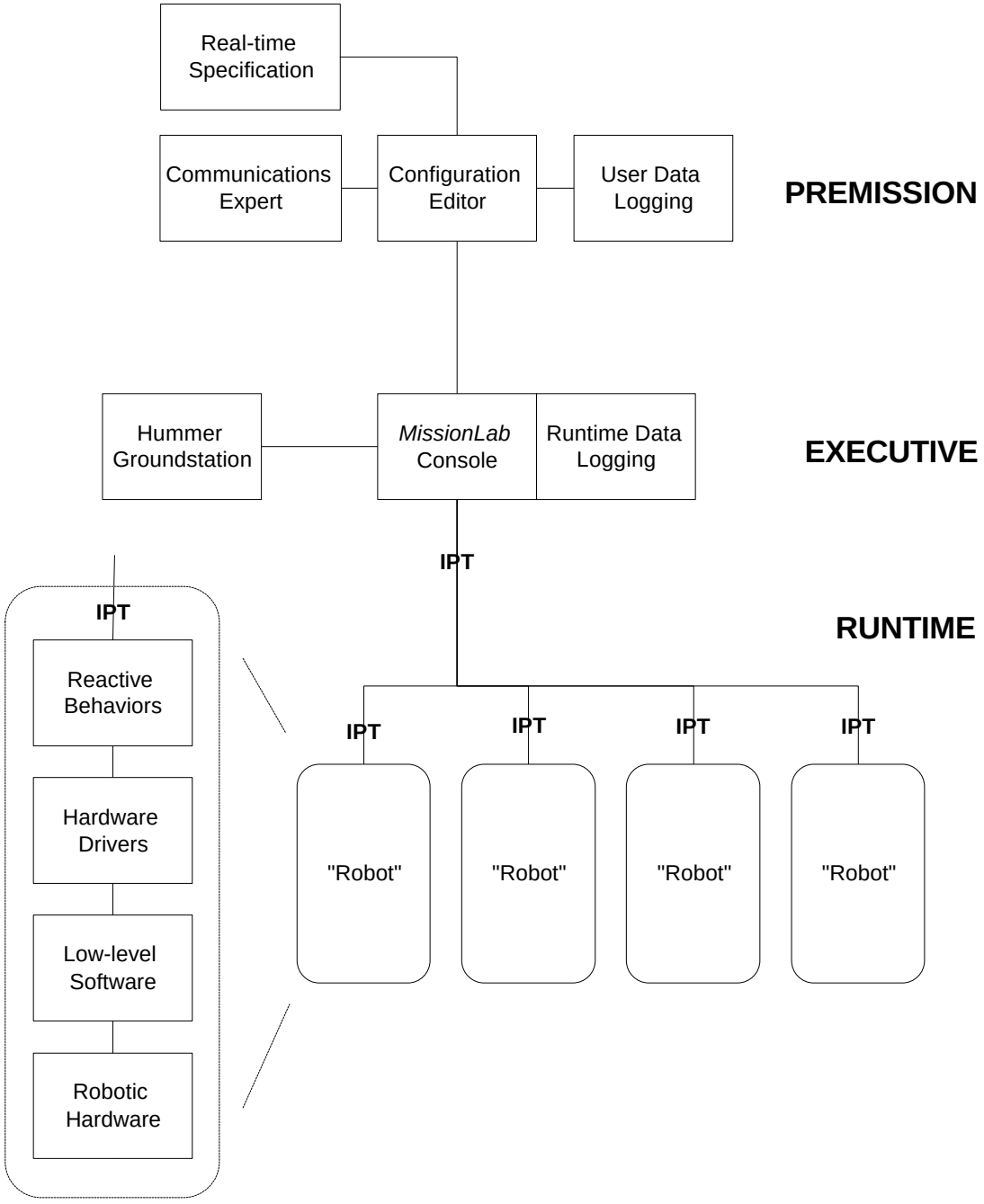

Figure 1: System Architecture

Figure 1 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 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. Section two of this document provides the functional specifications for each of these subsystems and their underlying components.

In Figure 1, typical communication paths between components are shown. Wherever separate threads of execution exist, this communication is implemented with IPT, to be described later. In other cases, communication may take the form of dedicated point-to-point links or conventional parameter-passing during the invocation 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 already purchased for the project.

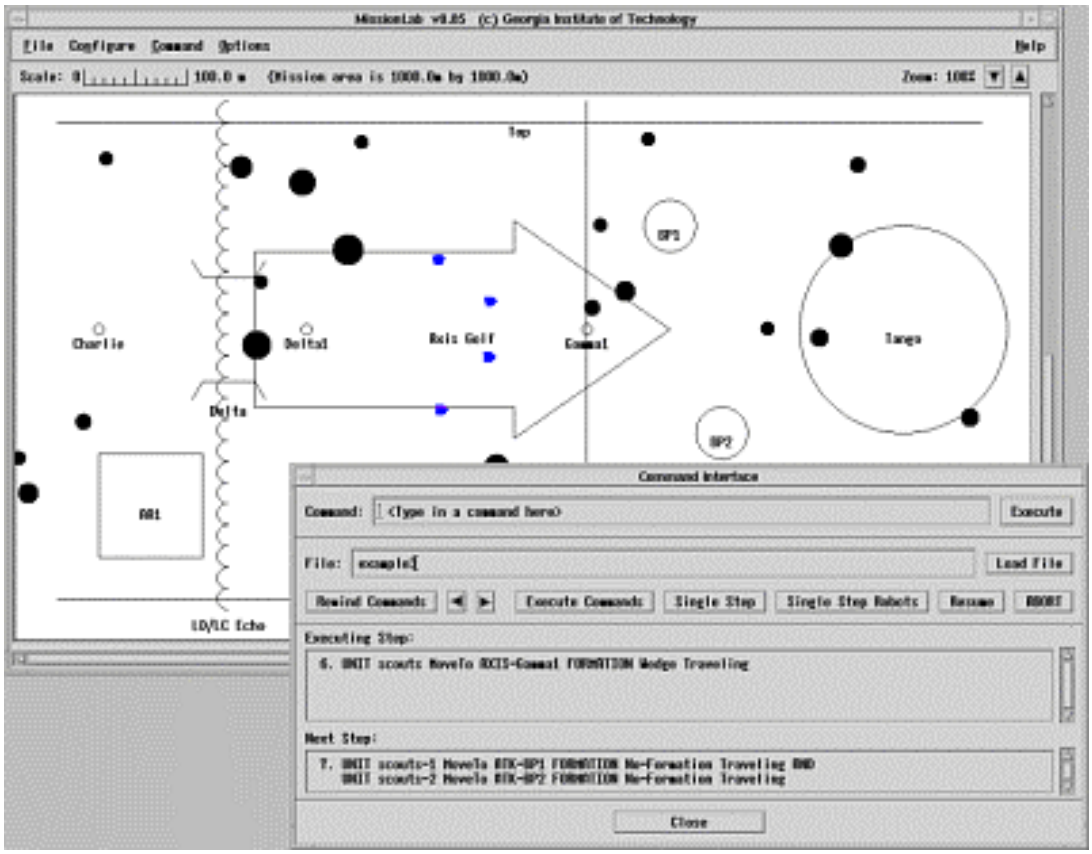

Figure 2: *MissionLab* Console (mlab).

2 SUBSYSTEM COMPONENT SPECIFICATION

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.

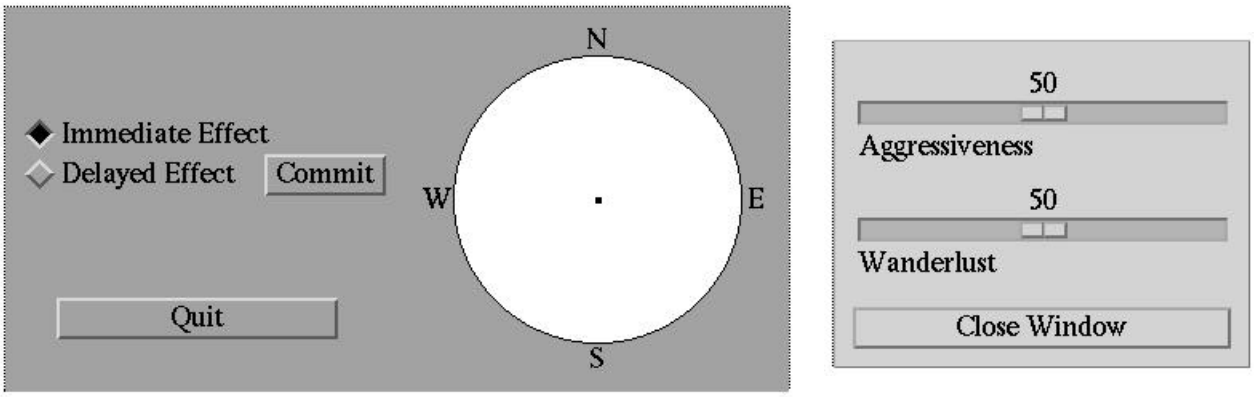

Figure 3: Teleautonomous operation in MissionLab. Dialog boxes allow operator to specify direction and "personality."

2.1.1 MissionLab Console

The *MissionLab* console (mlab) (Figure 2) 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), described below, 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

- 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 3).

• 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

The *MissionLab* console also provides a display (Figure 4) 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.

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.

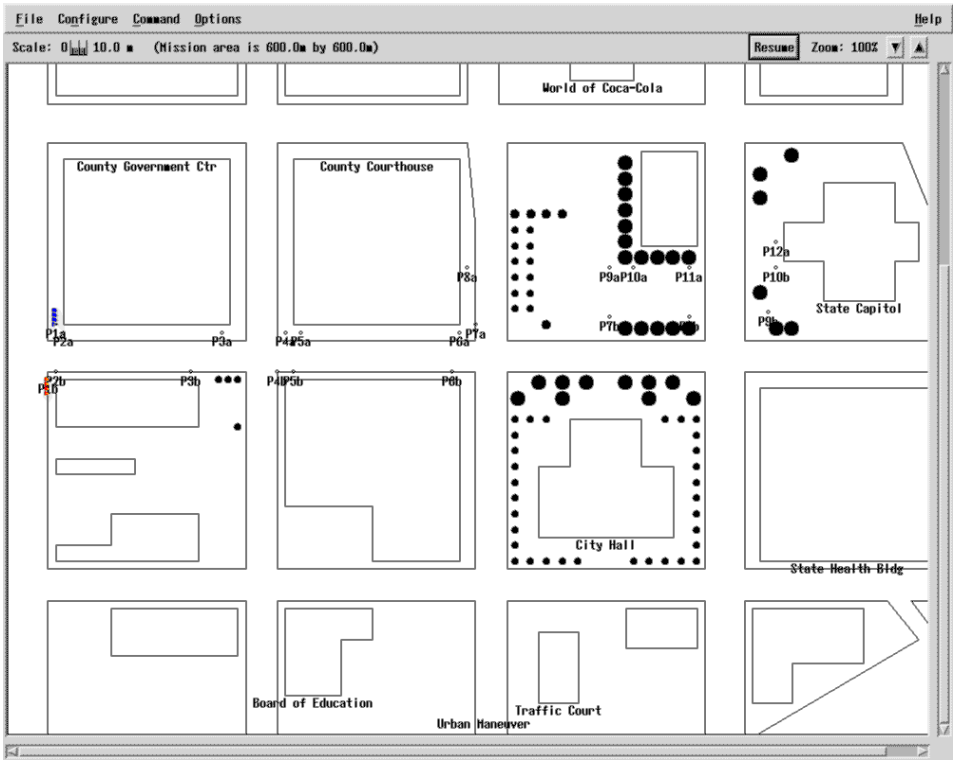

Figure 4: Display during execution of simulated mission (display of actual robot mission is similar).

2.1.2 Runtime Data Logging

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

2.1.3 Hummer Groundstation

A mobile groundstation based on a commercial Hummer (Figure 5) carries the operator environment to the field. It is capable of semi-autonomous operation, with actuated brake, throttle, and steering. Sensor capablities include a gyro-compass and inclinometer, with provision for the integration of GPS. The onboard controller is a PC-104 stack, with interfaces to all actuators as well as additional isolated I/O.

A tailgate ramp will facilitate the deployment of smaller robots in a marsupial fashion. The ramp could be actuated for either remote (teleautonomous) or fully-autonomous operation.

Figure 5: Hummer groundstation.

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 will provide 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 6). 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.

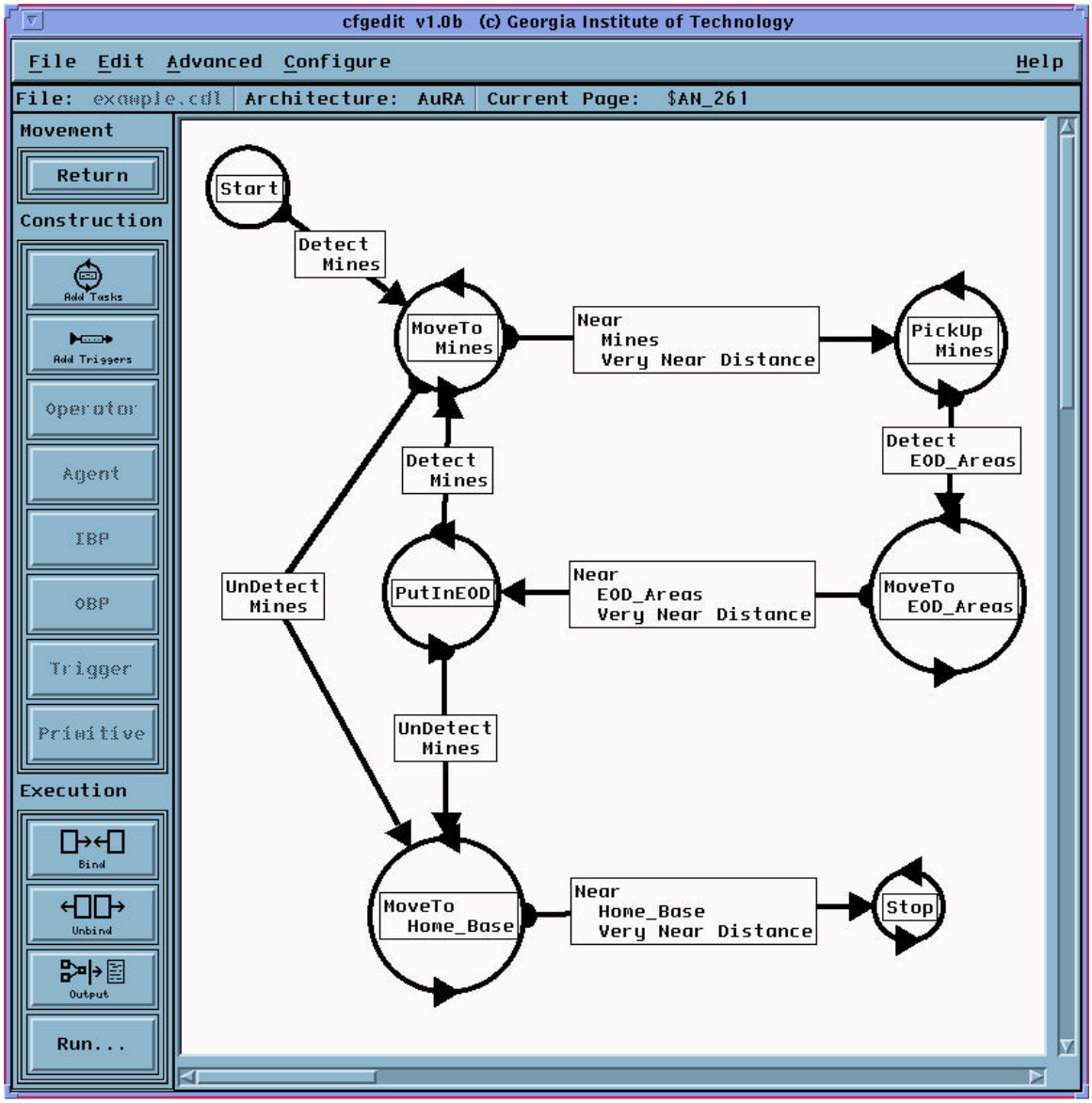

Figure 6: Graphical configuration using *cfgedit.*

2.2.2 Communications Expert

The communications expert (under development) will provide mission-specific recommendations for communications methods and topology (e.g., broadcast, point-to-point). It will enable a user to automatically configure the communications links between robots necessary to support a mission. This follows the same pattern as our earlier work in the design of a formation expert used in the DARPA UGV Demo II program.

2.2.3 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).

Figure 7: Usability Laboratory.

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 is expected among the various robots that are supported, but the expected set of routines for the TMR platforms (Pioneer AT and 97-20 "Urby") 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" (go to another orientation)
  - steer – rotational equivalent of "drive" (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 (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 (see PSOS and ARC below). These will include 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.

2.3.4 Robotic Hardware

Although (as described earlier) the Hummer groundstation has a robotic capability, the primary elements of robotic hardware for this project consist of the chosen robot platform itself and its most complex sensor system, each of which has an internal microcontroller. The specifications of these two components are described in this section.

2.3.4.1 Pioneer AT robot

The Pioneer AT "Outlaw" (Figure 8) is the 4-wheel drive, all-terrain version of the original Pioneer robot, a product of ActivMedia Inc. that is manufactured by Real World Interface. Its features include reversible-DC motor drives with wheel encoders, a magnetic compass, and onboard sonar sensors for forward and side object ranging and mapping. A rear-looking sonar sensor will be added.

Figure 8: Pioneer AT robot platform.

The Pioneer AT satisfies the platform and mobility specifications originally given for DARPA BAA 98-08. The Pioneer specifications are given below, but do not include the impact of accessories such as the pan-tilt-zoom vision system.

Pioneer AT Platform Specifications	
Characteristic	**Published specification**
Length	45cm
Width	50cm
Height	24cm
Weight	11.3kg
Run time	2-3 hrs
Body clearance	5.0cm
Translate speed max	1.5 m/sec
Traversable step max	8.9cm
Traversable gap max	12.7cm
Traversable slope max	60% grade

2.3.4.2 Newton Labs Cognachrome Vision System

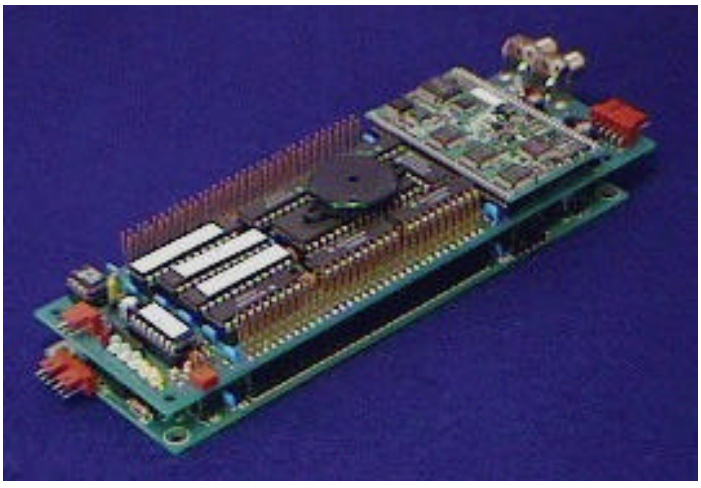

Figure 9: Newton Labs Cognachrome system

The Cognachrome Vision System (Figure 9) is capable of tracking multiple objects in the visual field, distinguished by color, at a full 60 Hz frame rate, with a resolution of 200x250 pixels. The board can also perform more general-purpose processing on low-resolution 24-bit RGB frame grabs from 64x48 to 64x250 pixels in resolution. Input to the Cognachrome Vision System is standard NTSC, and the board also outputs a 60 Hz NTSC image of the objects it is tracking.

In the Pioneer configuration, the Cognachrome Vision System outputs tracking data over serial ports to either the Pioneer microcontroller or another onboard computer. Number of tracking channels, number of objects to track per channel, and statistics to calculate per channel are all configurable.

Newton Labs Cognachrome Specifications	
Characteristic	**Published Specification**
Size	2.5" x 6.25" x 1.25" (64mm x 160mm x 32mm)
Weight	8 oz (230 g)
Power	
5v (digital)	400ma
5v (analog)	110ma
12v (analog)	20ma
Video input and output	NTSC
Tracking Resolution	200 x 250 pixels
Tracking statistics, per object	Centroid (x,y) Area Angle of orientation of major axis Aspect ratio
Number of tracking channels	3 (Each of the three color channels can track multiple objects of the given color)
Maximum simultaneous objects tracked	Approximately 25 (tracking performance will fall below 60 Hertz after 5 to 7 objects, depending on size and statistics computed)

2.4 Interprocess communications subsystem (IPT)

IPT is an object oriented **I**nter**P**rocess communications **T**oolkit that uses a message-based paradigm to connect various processes, or modules, in a system together. IPT was developed at Carnegie Mellon University to provide communication support for the UGV Demo II program.

Modules use IPT to establish connections between themselves and to send and receive messages. These connections can be considered as direct lines of communications setting up point-to-point links between modules without going through any "center."

Each message has an instance number and a type. The instance number can be used to keep track of the sequencing of messages, and the type contains information about how the message is formatted. Message data can be formatted to allow unpacking into C or C++ structures. Messages can be handled by user defined handling routines or by searching through a message queue.

A central communications process called the IPT *server* is the means by which all modules initiate communications. The IPT server has three jobs

1. To establish a consistent mapping between message type names (which are strings) and message type ID's (which are integers). This mapping means that each message will have associated with it a four-byte integer rather than an arbitrarily long type name. Having the server make this mapping ensures consistency across all modules that the server works with.
2. To act as a telephone operator connecting modules together. A module can thus request a connection to another module by name without having to know what machine that other module is on or how to connect to that other module.
3. To act as a central repository for log information. For example, individual IPT modules can be run in "logging mode." In logging mode an IPT module will send the headers of all messages it sends and receives to the IPT server. A system developer can use this log information to track down problems in the system. Once modules are connected, the server doesn't take up any more CPU cycles. It doesn't die, because IPT is a dynamic system. Modules are allowed to connect and disconnect throughout the lifetime of the system, and the server needs to be around in order to make and break these connections in an orderly fashion.

2.5 Real-time specification (MetaH)

MetaH is a language for specifying software and hardware architectures for real-time, securely partitioned, fault-tolerant, scalable multiprocessor systems. *MetaH* allows developers to specify how a system is composed from software components like processes and packages and hardware components like processors and memories. An associated toolset performs syntactic and semantic checks and also other functions such as real-time schedulability analysis and application composition.

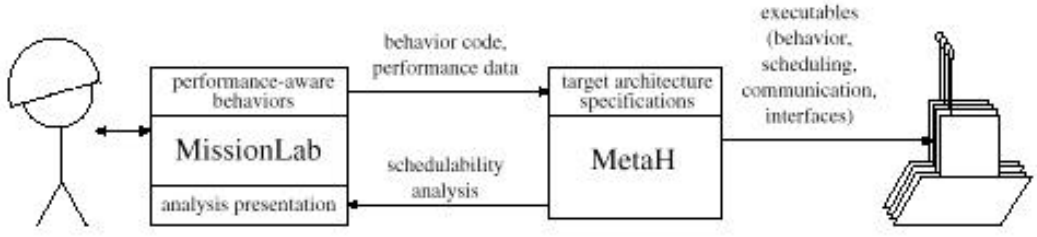

Figure 10 : Relationship of *MissionLab* and real-time specification tools.

The *MetaH* schedulability analysis tool builds and solves real-time schedulability models for any specified architecture. The schedulability analysis determines whether every processor and every inter-processor channel will be feasibly scheduled in every mode of operation. The schedulability model includes all executive and kernel overhead times.

Figure 10 illustrates our approach to providing timing analysis, using the *MissionLab* and *MetaH* toolsets. The warfighter uses *MissionLab* to specify a mission behavior for a team of robots, where *MissionLab* supports concepts and an interface that are meaningful to, and powerful for, this user for this purpose. *MissionLab* translates these specifications into lower-level behavioral code, which would be passed to the *MetaH* toolset along with performance data. The *MetaH* toolset uses such information to perform a real-time schedulability analysis to determine schedule feasibility and resource utilizations.