15:24:38 OCA PAD AMENDMENT - PROJECT HEADER INFORMATION 05/20/96

Active

Project #: C-36-X25 Cost share #: Rev #: 6
Center # : 10/24-6-R8016-0A0 Center shr #: OCA file #:
Work type : RES
Contract#: N00014-94-1-0215 Mod #: A00001 Document : GRANT
Prime #: Contract entity: GTRC

Subprojects ? : N CFDA: 12.300
Main project #: PE #:

Project unit: COMPUTING Unit code: 02.010.300
Project director(s):
ARKIN R C COMPUTING (404)894-8209

Sponsor/division names: NAVY / OFC OF NAVAL RESEARCH
Sponsor/division codes: 103 / 025

Award period: 931115 to 970331 (performance) 970331 (reports)

Sponsor amount	New this change	Total to date
Contract value	0.00	659,567.00
Funded	0.00	659,567.00
Cost sharing amount		75,000.00

Does subcontracting plan apply ?: N

Title: FLEXIBLE REACTIVE CONTROL FOR MULTI-AGENT ROBOTIC SYSTEMS IN HOSTILE ENVIRON.

PROJECT ADMINISTRATION DATA

OCA contact: Jacquelyn L. Bendall 894-4820

Sponsor technical contact

TERESA A. MCMULLEN, CODE 333
(703)696-4302

OFFICE OF NAVAL RESEARCH
BALLSTON TOWER ONE
800 NORTH QUINCY STREET
ARLINGTON, VIRGINIA 22217-5660

Sponsor issuing office

RESIDENT REPRESENTATIVE
(404)730-9270

OFFICE OF NAVAL RESEARCH
101 MARIETTA STREET, SUITE 2805
ATLANTA, GA 30323-0008

Security class (U,C,S,TS) : U
Defense priority rating :
Equipment title vests with: Sponsor GIT X

ONR resident rep. is ACO (Y/N): Y
ONR supplemental sheet

Administrative comments -
MODIFICATION NO. A00001 EXTENDS PERIOD OF PERFORMANCE TO 3-31-97.

CA8120

Georgia Institute of Technology
Office of Contract Administration
PROJECT CLOSEOUT - NOTICE

01-APR-1997 15:24

Closeout Notice Date 01-APR-1997

Project Number C-36-X25 Doch Id 45476

Center Number 10/24-6-R8016-0A0

Project Director ARKIN, RONALD

Project Unit COMPUTING

Sponsor NAVY/OFC OF NAVAL RESEARCH

Division Id 3314

Contract Number N00014-94-1-0215 Contract Entity GTRC

Prime Contract Number

Title FLEXIBLE REACTIVE CONTROL FOR MULTI-AGENT ROBOTIC SYSTEMS IN HOSTILE ENVIR

Effective Completion Date 31-MAR-1997 (Performance) 31-MAR-1997 (Reports)

Closeout Action:	Y/N	Date Submitted
Final Invoice or Copy of Final Invoice	Y	
Final Report of Inventions and/or Subcontracts	Y	
Government Property Inventory and Related Certificate	Y	
Classified Material Certificate	N	
Release and Assignment	N	
Other	N	

Comments

Distribution Required:

Project Director/Principal Investigator	Y
Research Administrative Network	Y
Accounting	Y
Research Security Department	N
Reports Coordinator	Y
Research Property Team	Y
Supply Services Department	Y
Georgia Tech Research Corporation	Y
Project File	Y

NOTE: Final Patent Questionnaire sent to PDPI

C-36-X25
1

ANNUAL REPORT FY1994

Flexible Reactive Control for Multi-Agent Robotic Systems in Hostile Environments

ONR/ARPA Grant #N00014-94-1-0215

Prepared by: Ronald C. Arkin (P.I.), Jonathan Cameron,
Doug MacKenzie, Tucker Balch, and Khaled Ali
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332
email: arkin@cc.gatech.edu
Fax: (404) 853-9376
Phone: (404) 894-8209

Contents

1. Introduction

This document constitutes the 1994 Annual Report for the ONR/ARPA Grant #N00014-94-1-0215 entitled Flexible Reactive Control for Multi-Agent Robotic Systems in Hostile Environments. This project is supported by ARPA's Real-time Planning and Control Program and has as a customer ARPA's UGV Demo II program. This first annual report reflects the first year's accomplishments within the context of an overall three year research program.

The goals of this research are to produce intelligent, flexible, reactive behaviors and methods for specifying and communicating information between multiagent teams. In particular we have been studying three closely related subjects:

- Formation Control - to allow teams of robotic agents to move in a coordinated manner through a potentially hostile environment without interfering with other active navigational behaviors.
- Teleautonomous Control of Multi-agent Teams - to allow a massive reduction in cognitive workload for the control of a group of robotic vehicles by permitting commands to be specified at the team level rather than at the individual agent level.
- Team Mission Specification Methods - to provide robust and flexible mission specification for reactive team military scenarios.

It is intended that all of these projects will be fielded in some capacity for the upcoming ARPA Demo C and Demo II scenarios involving 2-4 HUMMVs. This report surveys the progress made to date in each of these areas.

2. Formation Control

This research concerns the development of behaviors for formation maintenance in heterogeneous societies of mobile robots. The target platform is a set of four robotic vehicles to be employed as a scout unit by the U.S. Army. Formations are important in this application as they allow the unit to utilize its sensor assets more efficiently than if the robots are arranged randomly.

The robots in this work are heterogeneous in that each is assigned a unique identification number (ID). A robot's position in formation depends upon its ID. There are no other differences between robots. In future work the robots may differ substantially in their sensor capabilities, making it appropriate for specific robots to occupy particular positions in a formation.

Formation control is one part of a more complex behavioral assemblage which includes other components for task achievement. In addition to maintaining their position in formation, robots must simultaneously move to a goal location and avoid obstacles.

The formation behaviors presented here are implemented as *motor schemas*. Readers not familiar with motor schema-based reactive control are referred to [4].

2.1 Simulation Environment

Results were generated in simulation on a Sun SPARC IPC under SunOS and the X11 graphical windowing system. The simulation environment is a 1000 by 800 meter two dimensional "playing field" upon which various sizes and distributions of circular obstacles may be scattered. Each simulated robot is a separately running C program that interacts with the simulation via a Unix socket. The simulation displays the environment graphically and maintains world state information which it transmits to the robots as they request it. Figure 1 shows a typical simulation run. The robots are displayed as five-sided polygons (rectangles with points), while the obstacles are black circles. The robots' paths are depicted with solid lines.

Sensors allow a robot to distinguish between three perceptual classes: robots, obstacles and goals. The robots' sensors and actuators are implemented in the main simulation. When one of the robot's perceptual processes requires obstacle information for example, a request for that data is sent via the socket to the simulation. The information returned is a list of angle and range data for each obstacle in sensor range. Robot and goal sensor information is provided similarly. The robot moves by transmitting its desired velocity to the simulation process. The simulation process automatically maintains the position and heading of each robot.

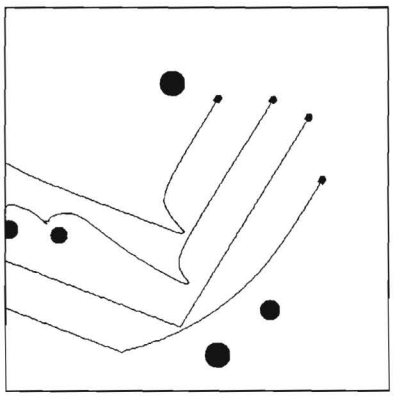


Figure 1: Typical simulation run showing four robots in a wedge formation executing a 90 degree left turn.

2.2 Formations

Several formations for four robots are considered:

- *line* - where the robots travel line-abreast.
- *column* - where the robots travel one after the other.
- *diamond* - where the robots travel in a diamond.
- *wedge* - where the robots travel in a "V".

For each formation, each robot has a specific position (based on it's ID). Figure 2.2 shows the formations and robots' positions within them.

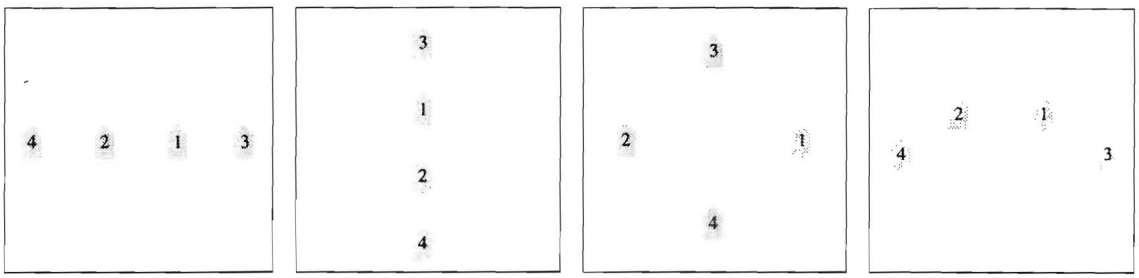


Figure 2: Formations for four robots (from left to right: line, column, diamond, wedge)

2.3 Behavioral Integration

The focus is on formation maintenance, but formation behaviors must meld with other behaviors such as obstacle avoidance. In the examples here, the task for each robot is to move to a goal location, avoid obstacles, avoid collisions with other robots and maintain formation position simultaneously. The primitive behaviors, $move-to-goal$, $avoid-static-obstacle$, $avoid-robot$, and $maintain-formation$, respectively, implement these behaviors. Each behavior generates a vector representing desired direction and magnitude of movement. A gain value indicates the relative importance of the individual behaviors. The high-level combined behavior is generated by multiplying the output of each primitive behavior by its gain, then summing the results. Gains and other schema parameters are listed in Table 1. See [4] for a more complete discussion of the parameters of the $avoid-static-obstacle$ and $move-to-goal$ motor schemas.

2.4 Approach

Formation maintenance is accomplished in two steps: first, a perceptual process, $detect-formation-position$, determines the robot's proper position in the formation; second, the motor schema $maintain-formation$ generates a movement vector towards it.

Parameter	Value	Units
avoid − static − obstacle		
gain	2.0	
sphere of influence	50	meters
minimum range	5	meters
avoid − robot		
gain	2.0	
sphere of influence	20	meters
minimum range	5	meters
move − to − goal		
gain	1.0	
maintain − formation		
gain	1.0	
desired spacing	50	meters
controlled zone radius	25	meters
dead zone radius	0	meters

Table 1: Parameter values used for motor schemas.

2.4.1 Formation Position References

Each robot must compute its proper position in the formation for each movement step. Three techniques for formation position determination are being explored:

- **Unit-center-referenced**: a unit-center is computed by averaging the x and y positions of each robot. Each robot determines its formation position relative to that center.
- **Leader-referenced**: each robot determines its formation position in relation to the navigational leader (the unit leader is not necessarily the navigational leader). The navigational leader does not attempt to maintain formation; the other robots are responsible for formation maintenance.
- **Neighbor-referenced**: each robot maintains a position relative to one other robot.

These relationships are depicted in Figure 3. Arrows show how the formation positions are determined. Each arrow points *from* a robot *to* the associated reference. The perceptual schema *detect−formation−position* must use one of these references to determine the proper position for the robot. The spacing between robots is determined by the *desired spacing* parameter of *detect − formation − position*.

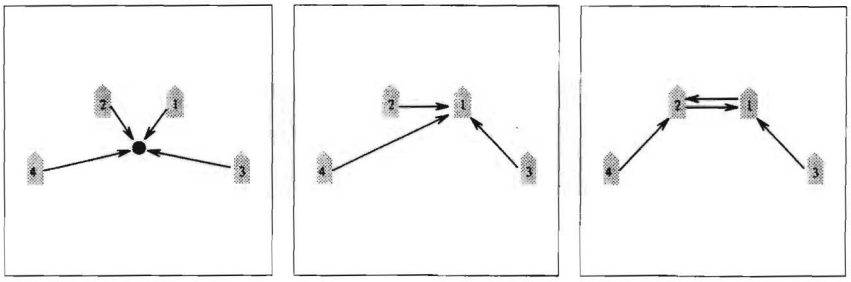


Figure 3: Formation position determined by various reference techniques (from left to right: unit-center, leader, neighbor)

2.4.2 The *maintain – formation* Motor Schema

Once the desired formation position is known, the *maintain – formation* motor schema generates a movement vector towards it. In all cases the direction of the vector is towards the desired formation position. The magnitude of the vector depends upon how far the robot is from the desired position. Figure 4 illustrates three zones (defined by distance from the desired position) used for magnitude computation. The radii of these zones are parameters of the *maintain – formation* schema. In the example case, robot 3 is attempting to maintain a formation position to the right of and behind robot 1. Robot 3 is in the controlled zone, so a moderate force towards the desired position (forward and right) is generated by *maintain – formation*. The magnitude of the vector is computed as follows:

- **Ballistic zone**: magnitude set at maximum (the schema's gain value).
- **Controlled zone**: magnitude varies linearly from maximum at the farthest edge of the zone to zero at the inner edge.
- **Dead zone**: in the dead zone vector magnitude is always zero.

2.5 Results

The *line*, *column*, *wedge* and *diamond* formations have been implemented using the unit-center-referenced and leader-referenced approaches. Neighbor-referenced formations are under development.

Figure 5 illustrates robots moving in each of the basic formations using the leader-referenced approach. In each of these simulation runs the robots were first initialized on the left side of the simulation environment, then directed to proceed to the lower center of the frame. After the formation was established, a 90 degree turn to the left was initiated. Results are similar for the unit-center-referenced formations.

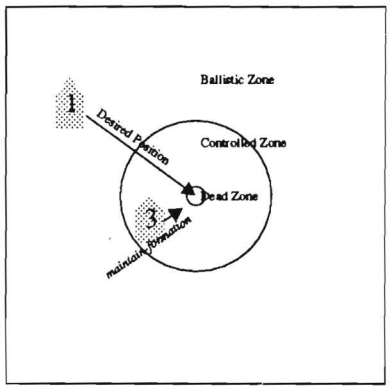


Figure 4: Zones for the computation of *maintain – formation* magnitude

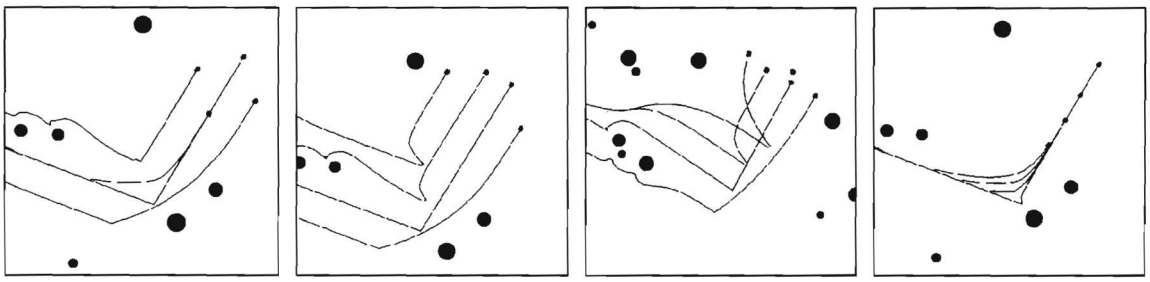


Figure 5: Four robots in leader-referenced *diamond*, *wedge*, *line* and *column* formations.

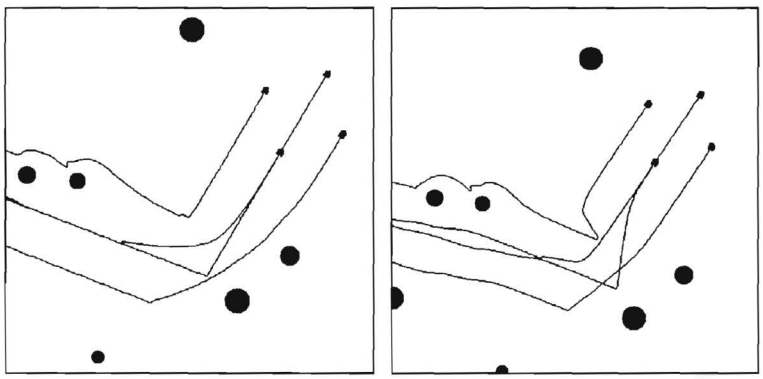


Figure 6: Comparison of leader-referenced (left) and unit-center-referenced (right) diamond formations.

Qualitative differences between the two approaches can be seen as a formation of robots moves around obstacles and through turns (see Figure 6). For leader-referenced formations any turn by the leader causes the entire formation to shift accordingly, but if another robot turns, the rest of the formation is not affected. In unit-center-referenced formations any robot move or turn impact the entire formation. In turns for leader-referenced formations, the leader simply heads in the new direction; other robots must adjust their position to move into formation. In unit-center-referenced turns, the whole formation initially appears to spin about a central point, as it aligns with the new heading.

2.6 Discussion

Which formation type and formation reference is best? The answer depends on many factors, including task, environment, and mission constraints. More simulations and analysis of qualitative results will help robot system designers match formations to applications. Sensor and mission constraints may also dictate specific types of formations. In any case, these applications probably rule out the use of unit-center-referenced formations:

- **Human Leader:** A human cannot reasonably compute a formation's unit-center on the fly, especially while simultaneously avoiding obstacles. A leader-referenced formation is most appropriate in this application.
- **Communications Restricted Applications:** The unit-center approach requires a transmitter and receiver for each robot and a protocol for exchanging position information. Conversely, the leader-referenced approach only requires one transmitter for the leader, and one receiver for each following robot. Bandwidth requirements are cut by 75% in a four robot formation.
- **Passive Sensors for Formation Maintenance:** Unit-center-referenced formations place a great demand on passive sensor systems (e.g. vision). In a four robot visual formation for instance, each robot would have to track three other robots which may spread across a 180 degree field of view. Leader- and neighbor-referenced formations only call for tracking one other robot [1].

2.7 Technical Transfer

The formation behaviors are being transferred to the UGV project at two levels: first as part of onboard software for UGV Demo C, and second as part of a military mission description and simulation package called MissionLab.

[1] The neighbor-referenced approach is used by U.S. Army Infantry Squads and Air Force fighters for visual formations.

At Demo C, formation routines will be installed on two UGVs to implement a "follow-the-leader" behavior during road following. At a Technical Demo to be held in conjunction with Demo C, we expect to additionally demonstrate a comprehensive suite of formation behaviors for two robots. Tech transfer for this application requires integration with the DAMN Arbiter, and the LinkManager software aboard the UGVs. Work towards this integration has already begun: interface descriptions are presently being worked out with Martin Marietta, and a version of the DAMN Arbiter is in use at Georgia Tech for integration testing.

In MissionLab, a military mission simulator, formation software has been combined with other behaviors to enable robots to carry out complex missions. Using the script-like language in MissionLab, a commander may specify complete missions for a team of robots then execute them in simulation. Section 4.3 of this report describes this in more detail.

3. Teleautonomy

Reactive multi-agent robotic societies can be potentially useful for a wide-range of tasks. This includes operations such as foraging and grazing (e.g., [1,11,7]) which have applicability in service (vacuuming and cleaning), industrial (assembly) and military (convoy and scouting) scenarios.

Although promising results have been achieved in these systems to date, purely reactive systems can still benefit from human intervention. Many purely reactive systems are myopic in their approach: they sacrifice global knowledge for rapid local interaction. Global information can be useful and it is in this capacity that a human operator can interact with a multi-agent control system.

A related problem in teleoperation is that a human operator is potentially overwhelmed by the large amount of data required to control a multi-agent system in a dynamic environment. This phenomenon is referred to as cognitive overload. The approach described in this section provides a mechanism to significantly reduce the human operator's cognitive and perceptual load by allowing the reactive system to deal with each robot's local control concerns. Two principal mechanisms to achieve this are by (1) allowing the operator to act either as a constituent behavior of the society or (2) to allow him/her to supervise the societal behavioral sets and gains, acting only as needed based upon observable progress towards societal task completion.

In this research, the human operator is allowed to control whole societies of agents; not one robot at a time, but rather controlling global behavior for the entire multi-agent system. This is a straightforward extension of our work in both multi-agent robotic systems [1] and teleautonomy [3]. The end product is a simple way for a commander to control large numbers of constituent elements without concern for low-level details (which each of the agents is capable of handling by themselves). In essence, the human

operator is concerned with global social *strategies* for task completion, and is far less involved with the specific behavioral tactics used by any individual agent.

3.1 Forms of Teleoperation

Telerobotic control is facilitated by two methods. The first method allows the human operator to give directional information to the robots. The second method allows the operator to interactively adjust the parameters of the reactive behaviors, thereby changing the overall behavior of the robot society.

3.1.1 Human Operator as a Behavior

The most common form of teleoperation for robots is to give the robot or robots directional information. In our approach, we are concentrating on giving directional information to the society in general, rather than to any particular robot or robots. In addition, the robots do not blindly follow the directional information of the human operator as a remotely controlled vehicle would. Instead, each robot interprets the directional instructions based on its particular situation in the world.

In our initial work, the robots used a motor schema-based reactive control system. For a more detailed account of this approach, see [4]. Each of a robot's primitive behaviors, or schemas, outputs a vector. The direction of the vector indicates the direction that the behavior wants the robot to go in. The magnitude of the vector indicates the amount that the behavior wants the robot to go in that direction. The vectors that are output from all the robots' behaviors are summed and normalized. The robot uses the resulting vector as the direction in which to move.

In our system, the human operator acts as one of the robots' behaviors when giving directional instructions. This is called the teleautonomy behavior. The human operator injects another vector into the system, just as if he were one of the robots' other behaviors. The direction of the vector indicates the direction that the human operator wants the society of robots to move in. The magnitude of the vector indicates the importance that the human operator thinks should be placed on moving in the specified direction. This vector is then summed and normalized along with all the other vectors from the other behaviors, and the resulting vector is used to indicate the direction of movement of the robot.

Thus, the robots do not blindly follow the instructions of the human operator, and each robot interprets the instructions based on his personal situation. For instance, if the human operator instructs the society of robots to move north and there is an obstacle directly north of robot #3, then robot #3 will not collide with the obstacle. The vector from the human operator will point in the direction of the obstacle, but the vector from the robot's **avoid-static-obstacle** behavior will point away from the obstacle. Because these vectors will cancel each other out, robot #3 will not collide

with the obstacle. However, all of the other robots will proceed north, assuming that their particular situation allows them to.

3.1.2 Human Operator as a Behavioral Supervisor

Each of the robots' behaviors has one or more parameters associated with the behavior. The values of these parameters determine exactly how the behavior will function. For instance, one parameter of the **avoid-static-obstacle** behavior is the gain. Increasing the gain for the **avoid-static-obstacle** behavior increases the magnitude of the vector output by the **avoid-static-obstacle** behavior. This has the effect of causing the robot to want to avoid obstacles more.

By adjusting the values of one or more behavioral parameters, the human operator can alter the overall behavior of the robots.

The human operator has the option to adjust the values of individual behavior parameters or to adjust the overall robots' behavior, or personality, in terms of more abstract personality traits, such as Aggressiveness.

First, the human operator can directly manipulate the behavior parameters. Every parameter for all behaviors is represented in the Telop interface. The human operator can adjust a single parameter at a time. This allows a human operator knowledgeable about the behaviors to fine tune the robot society's overall behavior.

The human operator can also manipulate the behavior parameters in terms of abstract personality traits. There is a set of abstract parameters, which are intended to represent general kinds of behavioral or personality adjustments that the human operator might want to make. In our current system the abstract parameters are Aggressiveness, Wanderlust, and Perceptiveness. The value of an abstract parameter controls the value of two or more individual parameters. The three abstract parameters used each control the values of two individual parameters, but an abstract parameter could control considerably more.

The abstract parameter Aggressiveness determines the amount that the robot is focussed on achieving its goal. Increasing the Aggressiveness parameter causes an increase in the **move-to-goal** behavior gain and a decrease in the **avoid-static-obstacle** behavior sphere of influence. The personality effect of this is to cause the robots to focus more on getting to their goal location and worry less about what is in their way. Likewise, decreasing the Aggressiveness parameter causes a decrease in the **move-to-goal** behavior gain and an increase in the **avoid-static-obstacle** behavior sphere of influence. The personality effect of this is to cause the robots to be more careful of obstacles and be less concerned about getting to their goal location.

The abstract parameter Wanderlust determines the desire of the robot to randomly explore the terrain it is in. Wanderlust controls the gain and the persistence of the **noise** behavior. Increasing the Wanderlust causes the robots to move more randomly.

The abstract parameter Perceptiveness determines the distance from the robot beyond which perception of obstacles and other robots is ignored. This is important, because if the robot's perception is noisy at the outer limits, the operator may want the robot to ignore the data coming from that region.

3.2 Teleoperation Interface

The Telop system includes a graphical user interface to facilitate the communication between the human operator and the robots. The interface consists of three parts: the main window, the meta-slider window and the detail-slider window.

3.2.1 Main Window

The Main Window contains an on-screen joystick and some other general controls (see Figure 7).

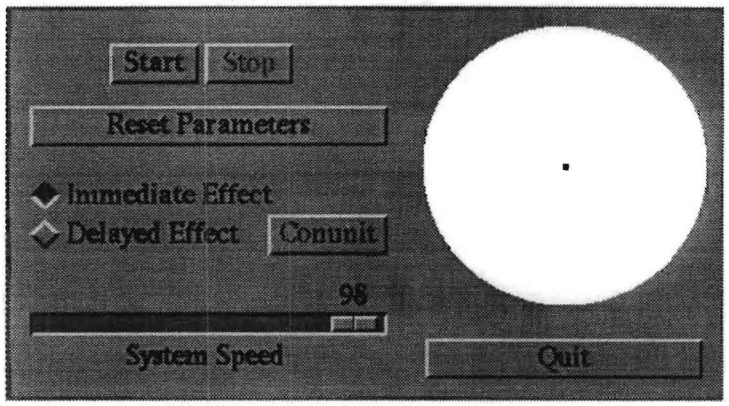


Figure 7: Main Window

The purpose of the on-screen joystick is to provide the human operator a means to give directional information to the robots, as described in the section **Teleoperator as a Behavior**. The position of the joystick indicates the vector that is injected into the system from the teleautonomy behavior. The direction that the joystick is depressed provides the directional component of the vector, and the amount that it is depressed provides the magnitude of the vector.

The main window also contains a toggle button to start and stop the robots from moving, a button to reset all of the behavior parameters to their default values, a slider-bar that controls the speed of the robots, and controls for modes that affect the timing of parameter changes.

The default mode of the system is the Immediate Effect mode. When the system is in this mode, any changes to the parameters, both individual and abstract, are immediately noticed and used by the robots' behaviors.

In the Delayed Effect mode, parameter changes are not immediately noticed by the behaviors until a Commit button is pressed. The behaviors continue using the parameters that the system had when the Delayed Effect mode was entered or the Commit button was last pressed, whichever is most recent. This allows the human operator to change the values of multiple parameters and then have the changes take place simultaneously when the Commit button is pressed.

3.2.2 Meta Window

The Meta Window contains three meta-sliders, and a button that pops up the Detail Window (see Figure 8).

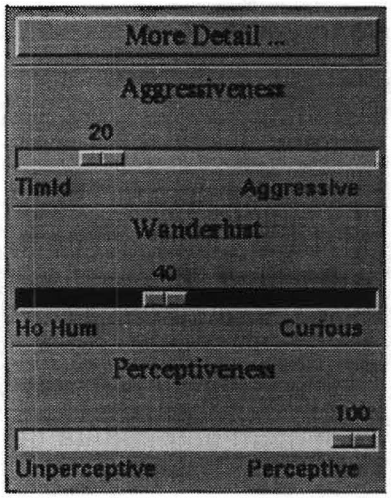


Figure 8: Meta Window

The meta-sliders are slider-bars that control the value of abstract parameters. There is one meta-slider for each of the abstract parameters: Aggressiveness, Wanderlust, and Perceptiveness. We call them meta-sliders because, as the user moves a meta-slider, it then in turn moves two or more detail-sliders, as described below, that correspond to the individual behavior parameters associated with that abstract parameter.

Each meta-slider is a different color. The detail-sliders in the Detail Window that are associated with a meta-slider are the same color.

3.2.3 Detail Window

The Detail Window contains several detail-sliders and a button that removes the detail window (see Figure 9). The detail-sliders are slider-bars, each of which controls the value of an individual behavior parameter.

The detail-sliders are physically arranged into groups. Some groups contain slider-bars for behavior parameters used in a particular state of the robots' task. Other

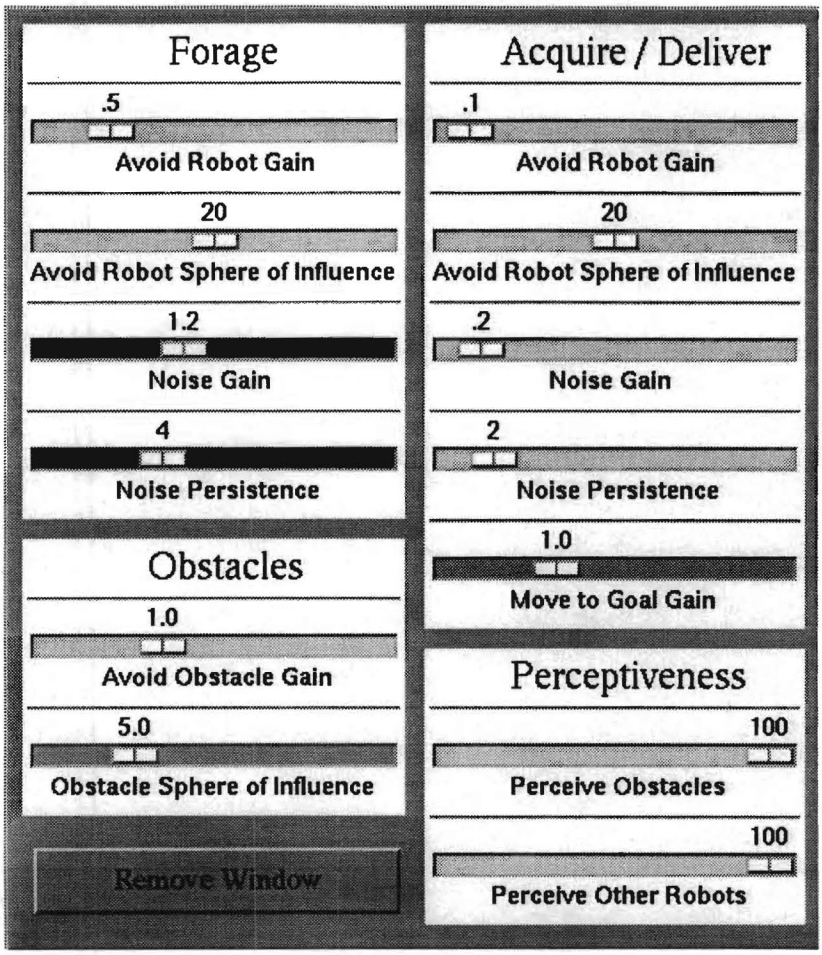


Figure 9: Detail Window

groups contain slider-bars that deal with behavior parameters for a general concept, such as the Obstacle group, which contains slider-bars associated with a behavior for avoiding obstacles.

The groupings that we are using now are based on our own particular task. The slider-bars would either need to be regrouped based on the task they are being used for, or there would have to be a menu of predetermined tasks. Choosing a task from this menu would automatically regroup the slider-bars for that particular task.

Most detail-sliders are the same color, but those detail-sliders that are associated with one of the meta-sliders are the same color as the meta-slider they are associated with.

3.2.4 Usability Tests

A set of usability tests has been conducted on the interface of Telop. These tests yielded useful information for making the interface more helpful and usable. We have made changes to the Telop interface based on the suggestions from these studies.

The evaluators in the usability tests were people working in the robotics group at the Georgia Institute of Technology. In the future, we plan to conduct at least one more set of usability tests on Telop. At that time, we plan to use military students as evaluators.

3.3 Teleautonomy Tests

A set of experiments were done to test the usefulness of the teleautonomy behavior for certain tasks. Only the teleautonomy behavior was tested. The behavioral parameters were not modifyed during the testing. All of the tests were done in a simulation environment.

3.3.1 Simulation Environment

The system is tested on a graphical simulation environment prior to its port to our Denning robots. The objects represented in the simulation environment include robots, obstacles, and attractors. Each robot's trail is depicted by a broken line. Every robot uses the same set of behaviors (a homogeneous society), but the sensory input for each is different, depending on the robot's location within the environment. The robots can sense objects within a certain radius around them. They have the ability to distinguish whether a sensed object is an obstacle, another robot, or an attractor.

The agents have a limited form of communication between themselves. A robot is capable of communicating its current behavioral state or the location of an attractor that it is acquiring or delivering [6]. The communication is simulated by using shared

memory. Each agent only looks at this shared memory when there is no attractor within its sensing range.

In tasks that require the movement of attractors, more than one robot is allowed to contribute to the transport of the object at the same time. The net effect of this cooperation is simulated by having the robots move the attractor farther during each time unit if there are more robots carrying it. The distance traveled while carrying an attractor is determined by the mass of the object and the number of robots carrying it.

3.3.2 Tasks

The use of teleoperation in multi-agent systems was tested for three different tasks. The tasks were foraging, grazing (vacuuming), and herding the robots into a pen. In all three tasks, a teleoperator provided input at his own discretion.

In the foraging task, the robots wander around looking for attractors. When a robot finds a target object, it communicates its location to the other agents while simultaneously moving to acquire it. After its acquisition, the robot carries the attractor back to a home base, then deposits it, and finally returns back to the task of searching for more attractors. If a robot cannot detect an attractor within its sensory radius, it checks to see if any other agent has communicated the location of another candidate goal object. If so, then the robot proceeds to acquire it.

In the grazing task, the robots are placed in an environment studded with obstacles. Initially, all of the floor that is not covered with an obstacle is "ungrazed". Each section of the floor that is ungrazed is treated as if it had an attractor on it. That is, a robot can sense an ungrazed section of floor from a distance, and it can also communicate the presence of an ungrazed section of the floor to the other robots. When an agent passes over an ungrazed region it becomes clean. The task is completed when a certain percentage of the floor, specified in advance, has been grazed. The robots normally wander randomly until an ungrazed floor area is detected.

In the herding task, there is a pen with an opening formed of obstacles in the simulation environment. All the agents are initially outside of the pen. The robots remain in the *forage* state for the duration of the run and wander aimlessly in random directions. The robots are repulsed by the obstacles and the other robots. The task is to get all of the robotic agents inside the pen at the same time.

3.3.3 Results

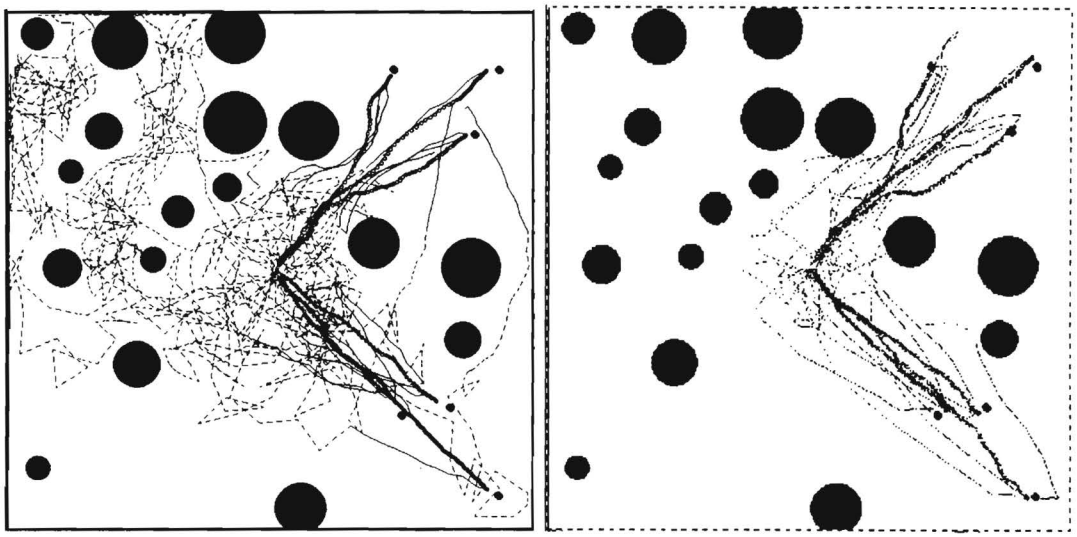
For the foraging and grazing tasks, tests were conducted that compared the total number of steps taken by the robots to complete the tasks with and without the help of a teleoperator. For the herding task, no comparison could be made between teleoperation and no teleoperation, because the likelihood of all the robots wandering

into the pen by themselves at the same time is virtually nil. Interesting information was gained about this task nonetheless.

In the tests conducted for the foraging task, three robots were used to gather six attractors. The density of obstacles in the environment was 10%. The total number of steps required to finish the task was measured both with and without a teleoperator. If teleoperation is used wisely, it can significantly lower the total number of steps required to complete the task by greatly reducing the time spent in the *forage* state (i.e., the number of steps that the robots spend looking for attractors). If none of the agents currently sense an attractor, then the teleoperator can assist by guiding the robots in a fruitful direction. However, once the robots can sense an attractor, the teleoperator should stop giving instructions, unless the instructions are to deal with a particularly troublesome set of obstacles. In general, the robots perform more efficiently by themselves than when under the control of a teleoperator if the agents already have an attractor in sight. The human's instructions tend to hinder the robots if they are already moving to acquire or return an attractor. Indeed, when teleoperation is used at all times, the overall number of steps required for task completion often increases when compared to no teleoperation at all. However, if the human only acts to guide the robots toward an attractor when none are currently detected, significant reductions in time for task completion are possible. The average over several experimental runs of the total number of time steps required for task completion when teleoperation was used in this manner was 67% of the average task completion time when no teleoperation was used.

An example trace of a forage task without teleoperation is shown in Figure 10(a). Another trace of the same forage task with a human teleoperator helping the robots find the attractors when they did not have one in sensing range is shown in Figure 10(b). The robots all started at the home base in the center of the environment. In the run without teleoperation, the robots immediately found the two closer attractors at the lower right. Then they quickly found the two closer attractors at the upper right. At this point, the robots did not immediately detect the remaining two attractors. Two of the three agents proceeded by chance to the left and upper left sides of the environment, wandering unsuccessfully while seeking an attractor. Eventually, the other robot found the attractor in the lower right corner, and the other two robots moved to help with its return. After delivering it to the home base, the robots wandered again for a while without finding the last attractor. Finally, the last attractor was detected and successfully delivered to home base. In the same world with the help of a human teleoperator, the two protracted periods of wandering while searching for attractors are avoided. This indicates the types of environments where the use of teleoperation for the forage task is most beneficial. The greatest benefit from teleoperation can be seen when there are one or more attractors that are far from both the home base and the start locations of the robots. Typically, this is when the robots do not sense the target objects without wandering for a while.

For the grazing task, five robots were used. A sample run of a grazing task is



(a) (b)

Figure 10: Foraging task.(a) Without Teleoperation (b) With Teleoperation

shown in Figure 11. In this case, the robots performed poorly when a large amount of teleoperation was involved. Teleoperation only proved useful when the robots had difficulty in locating a section of ungrazed floor. When the robots had already detected an ungrazed area, they performed better without any input from the teleoperator. The agents' performance degraded considerably, often taking several times longer to complete the task, if teleoperation was used when a robot had already located an ungrazed floor area. Moreover, since remaining untreated areas tend to be clustered together in large patches, the agents typically do not need to spend long periods of time looking for another ungrazed spot (which is opposite the case of the foraging task discussed above). Therefore, the use of teleoperation did not help significantly with the grazing task. When teleoperation was used solely to help the robots find ungrazed floor area when they were not already cleaning, only a 4% improvement in average task completion time performance was observed when compared to not using teleoperation. Thus, when used wisely, teleoperation helped somewhat but not to a large extent.

For the herding task, five robots were herded into a pen that was 36 units long by 18 units wide, with a 12 unit long door in one of the longer sides. All of the robots started at one spot on the side of the pen with the door. In most test runs, the teleoperator encountered no difficulty with this task. He was able to herd the robots into the pen with no problem. In some of the test runs, there were a few minor difficulties, such as robots wandering back out of the pen after having been herded in. However, the teleoperator was still able to complete the task without much frustration and in a reasonable amount of time. The results of a test run for the herding task is shown in Figure 12.

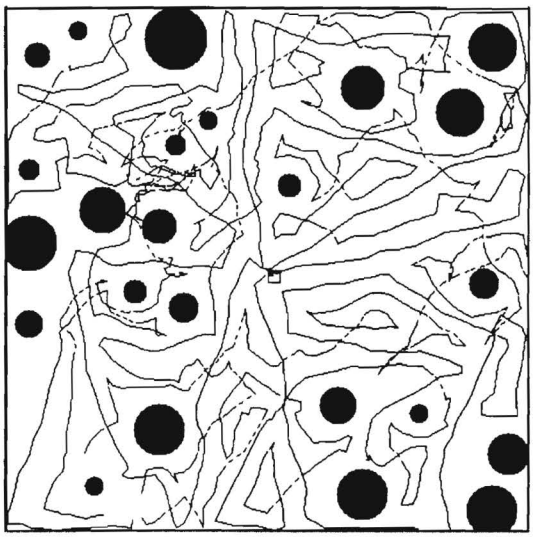


Figure 11: Grazing Task

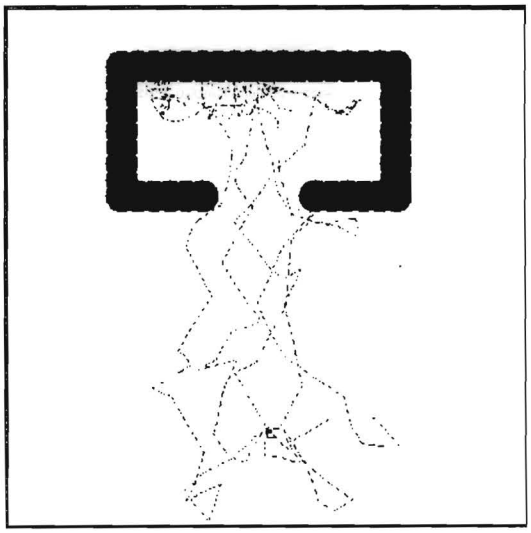


Figure 12: Herding Task

3.3.4 Analysis

Some conclusions can be ascertained from the studies conducted thus far. It should be remembered, however, that these are preliminary studies, and there are many variables that have not yet been explored. For instance, we intend to explore the effects of teleoperation while varying the number of robots for a particular task, to study the role and impact of different inter-agent communication methods on teleoperation, and to conduct an analysis of what types of environments teleoperation is most suited for.

The use of the **teleautonomy** schema in conjunction with the robots' other behaviors proved particularly effective for the foraging task, while being less so for the grazing task (vacuuming). Herding the robots into a pen was arduous but feasible using this method. During foraging, the best results were observed when teleoperation was used only to guide the robots in the direction of an attractor if one had not been previously sensed. For the vacuuming task, teleoperation was not significantly better than no teleoperation, although minor improvements were observed. The best results were again seen when teleoperation was used in guiding the robots towards dirty areas that were outside the sensor (or communication) range of the agents.

Trying to herd the robots into a pen is possible, although a frustrating task. Two conceivable improvements can be used for this task regarding teleoperation. The first is to allow the teleoperator to turn off the input from the teleoperation schema for specific robots but not for others, allowing the operator to concentrate on the outside robots without worrying what effects his actions will have on robots already inside the pen. The other improvement is to allow the teleoperator to completely stop a robot's movement when it is inside the pen. In this way, the output of the teleoperation schema could be thought of as producing a vector that nullifies the vectors produced by the robot's other schemas. However, both of these strategies involve producing different output for the **teleautonomy** schema for different robots. This means that the teleoperator would have a greater burden, defeating the purpose of this research in reducing the cognitive workload.

Another important point is that if the teleoperator is given unrestricted control of the magnitude of the vector produced by the teleoperation schema, it is possible for the teleoperator to force a robot to collide with obstacles and other robots. The teleoperator must be careful when increasing the gain of the **teleautonomy** schema so that this does not occur. It can be a delicate task to override the output of the **noise** schema, which is necessary to cause the robots to quickly move in a particular direction, while not overriding the **avoid-static-obstacle** behaviors.

3.4 Integration with ARPA UGV Project

The Telop system is being developed for use as part of the ARPA UGV project. The integrated demo of Demo C will include the teleautonomy behavior. A tech demo

at Demo C and the integrated demo at Demo II will include both the teleautonomy behavior and the slider bars for modifying behavioral parameters.

3.4.1 Teleautonomy Behavior

We are currently working to make Telop compatible with the DAMN arbiters, which are the arbitration systems used on the vehicles in the UGV project. The main window of the interface has been modified in appearance and functionality to accommodate the differences between the combination mechanism used in our schema system and the arbitration mechanism used in the DAMN arbiters. The joystick now controls the direction and the speed of the society of robots. When the joystick is used, the interface sends messages (via TCX) to the teleautonomy behaviors. These messages indicate the direction and the speed that the society of robots should move in. Two slider bars have been added to adjust the weight that the DAMN turn arbiters and the DAMN speed arbiters use when considering the votes from the teleautonomy behavior. The communication with the DAMN arbiters to change the weights is done using TCX.

We are in the process of implementing the teleautonomy behavior in the same form as the other DAMN behaviors. There are actually two teleautonomy behaviors, one for turning and the other for speed. Each robot will be running one teleautonomy turn behavior and one teleautonomy speed behavior. When the behaviors receive a message from the Telop interface indicating a direction and speed for the society to move in, the behaviors will compute a turn radius and speed based on the situation of the particular robot that the behavior is resident on. This conversion from holonomic to nonholonomic instructions has not yet been implemented. Then the behaviors send their votes for this turn radius and speed to the DAMN arbiter. The votes for turn radii are actually distributed among all the possible turn radii as a gaussian centered at the desired turn radius.

We are also creating an interface for modifying the default parameters that the Telop interface will use when it is started by either MRPL or the STX interface. This default editor will also allow modification of the system defaults for the parameters.

The interfaces are implemented using UIM/X for ease of integration with the other interfaces in the project.

3.4.2 Parameter Modification

For a technical demo at Demo C and for the integrated demo at Demo II, we are implementing detail-sliders and meta-sliders for controlling behavioral parameters both in terms of the individual parameters and abstract personality traits. The detail-sliders will correspond to the possible behaviors running on the vehicles. The grouping of the detail-sliders will be based on the tasks and concepts that are part of the military scenario used in the UGV demos. The meta-sliders will then be set up to control appropriate detail-sliders based on the personality trait that they represent.

There will be two detail-sliders for the weights of each behavior in the arbiters. The first will control the weight in the turn arbiter, and the second will control the weight in the speed arbiter. In addition, we will analyze the internal parameters present in the behaviors used with DAMN. If it is appropriate for a particular parameter, we will include a slider bar in the Telop interface for modifying this parameter.

When a slider bar is moved, the Telop interface will send messages to the appropriate arbiters or behavior on each vehicle. The messages will instruct the arbiter to change the weight that it uses when considering the votes of a particular behavior.

4. Configuration Design Support for Mission Specification

Configuration design tools are being developed which will support graphical construction of abstract configurations which can then be compiled to execute on the MRPL system for UGV Demo II. Planned capabilities include a graphical editor for creating configurations, support for automated configuration evaluation, and a Multi-Robot Programming Language (MRPL) code generator.

4.1 The Configuration Description Language (CDL)

The context free Configuration Description Language (CDL) provides a solid theoretical foundation for specifying architecture and robot independent configurations of behavior-based robots. Societies of robots come in three types; trivial, homogeneous teams, and heterogeneous castes. The language specifies the coordination between members of homogeneous teams, members of heterogeneous castes, assemblages of behaviors for individual robots, as well as perceptual strategies within primitive sensorimotor behaviors.

A preliminary version of the language has been developed. The grammar G generating the language will be described by the notation [10] $G = (V, T, Q, S)$, where V is the set of variables, T is the set of terminal symbols, Q is the set of productions, and S represents the highest-level society (the start variable). Using this notation, G is described as

$$\begin{aligned} G \;=\; & (\{S, X, R, A, C, Y, B, P, Z\}, \\ & \{\mathbf{p}_i, \mathbf{m}_j, \mathbf{a}_k, /_l, *_m, +_n, -_o, \%_p, @_q, \#_r, =_s, \{, \}, [,], \langle, \rangle, `, ', (,)\}, \\ & Q, \\ & S) \end{aligned}$$

and Q consists of the productions

$$
\begin{array}{lcl}
S & \rightarrow & R \mid \text{`}/_l\ XS\text{'} \mid \text{`}*_m\ \ XS\text{'} \\
X & \rightarrow & XS \mid S \\
R & \rightarrow & \{A\} \\
A & \rightarrow & B \mid [+_n\ YA] \mid [-_o\ YA] \mid [\%_p\ YA] \mid [@_q\ YA] \\
Y & \rightarrow & YA \mid A \\
B & \rightarrow & \langle P\mathbf{m}_j \rangle \mid \langle \mathbf{a}_k P\mathbf{m}_j \rangle \mid \langle P \rangle \mid \langle \mathbf{a}_k P \rangle \\
P & \rightarrow & \mathbf{p}_i \mid (\#_r\ ZP) \mid (=_s\ ZP) \\
Z & \rightarrow & ZP \mid P
\end{array}
$$

Where

- S is a society
- X is a list of one or more societies
- R is a single robot
- A is a behavioral assemblage
- Y is a list of one or more assemblages
- B is a primitive sensorimotor behavior
- P is a perceptual module or a coordinated perceptual group
- Z is a list of one or more perceptual modules
- $\mathbf{p}_i, i \in$ *natural numbers* is an instance of a perceptual process
- $\mathbf{m}_j, j \in$ *natural numbers* is an instance of a motor process
- $\mathbf{a}_k, k \in$ *natural numbers* is an instance of an active perception motor process
- $/_l, l \in$ *natural numbers* is an instance of a caste (heterogeneous) society operator
- $*_m, m \in$ *natural numbers* is an instance of a team (homogeneous) society operator
- $+_n, n \in$ *natural numbers* is an instance of an assemblage cooperation operator
- $-_o, o \in$ *natural numbers* is an instance of an assemblage competitive operator
- $\%_p, p \in$ *natural numbers* is an instance of an assemblage sequencing operator
- $@_q, q \in$ *natural numbers* is an instance of the generic assemblage coordination operator
- $\#_r, r \in$ *natural numbers* is an instance of a perceptual fusion operator
- $=_s, s \in$ *natural numbers* is an instance of a perceptual sequencing operator
- ` ' delineates societies
- { } delineates agents (robots)
- [ ] delineates coordinated assemblages
- $\langle\ \rangle$ delineates primitive sensorimotor behaviors
- ( ) delineates a group of coordinated perceptual modules.

Sensors are explicitly represented to allow parameterization and to facilitate hardware binding. Perceptual modules function as virtual sensors which extract features from one or more sensation streams and generate, as output, a stream of features (individual percepts). Motor modules use one or more feature streams (perceptual inputs) to generate an action stream (a sequence of actions for the robot to perform). Perceptual coordination is the process of linking one or more perceptual modules to motor modules and is partitioned into three categories[2]: sensor fission, action-oriented perceptual fusion, and sensor "fashion". Active perception utilizes a special motor module which generates an action stream to modify the information the sensor is providing. A primitive behavior consists of one or more perceptual modules and a motor module generating a stream of actions based on perceptual inputs. An assemblage can be treated as a single sensorimotor behavior even though it may be recursively composed of many primitive behaviors and coordination strategies. Each individual robot is controlled by a single assemblage.

4.2 Code Generators

The Configuration Network Language (CNL) will be used to represent the output of the CDL compiler. CNL has been designed and implemented as a dataflow language, where the executable functions represented by the nodes are specified using a standard programming language. A compiler generating C++ code for the schema architecture has been developed and is in regular use in the mobile robot lab. A compiler generating MRPL code for the ARPA architecture will be developed this fall.

4.3 MissionLab - Simulation System Implementation

The MissionLab simulation system architecture is shown in Figure 13. The basic parts are a simulation system, a display system, and robot control software. The simulation system is shown in a dotted box.

The simulation system includes a module which can read and interpret overlay files, a database where the scenario information is stored, a module which can read and interpret command files (and construct an internal list of instructions), and a command interpreter which will execute the commands from the internal list of commands. The commands are digested and sent to the individual robots to be executed. This also includes some simple execution monitoring. For details of the simulation system, see [8].

The robot control software is a preconfigured set of assemblages for accomplishing a variety of tasks. Each assemblage is a set of schemas (or low-level behaviors). An executive inside the robot software receives the command from the simulation system, loads information from the command (such as goal locations), and selects the proper set

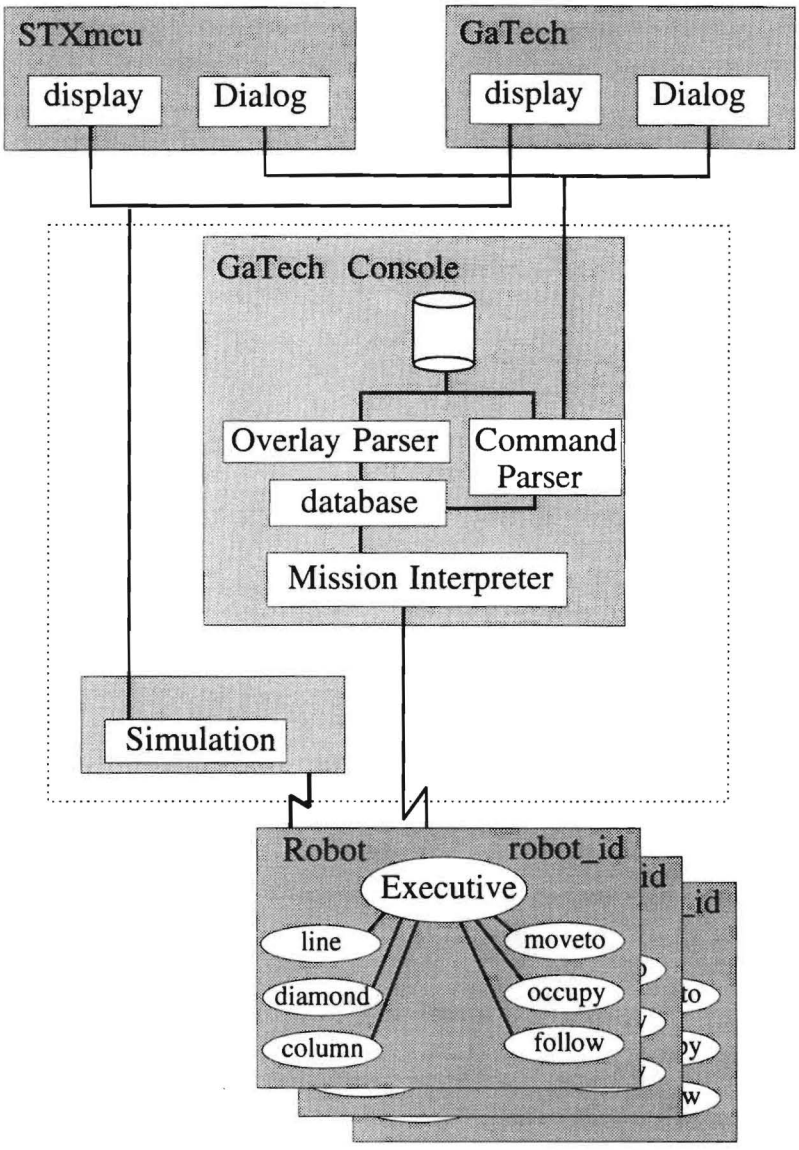


Figure 13: Overview of the current MissionLab system.

of behaviors to execute the desired task. Ultimately, MRPL behaviors will be encoded as an alternate behavior strategy.

The simulation system spawns separate control processes to control the robots. Each of these processes runs a copy of a robot control program. If the robot were real, these processes would communicate with the robot (perhaps via radio links). In our case, the robots are simulated. These separate processes are running independently, in parallel. They communicate with the simulation system using the TCX inter-process communication package [9].

The simulation system also coordinates information exchange between the robots. When commands are executed, the simulation system sends the command to the appropriate robot with whatever modifications are necessary. For instance, a "unit" might consist of several robots. When the unit is commanded to do something, the simulation system uses the console database to determine what robots need to receive the command. The simulation system then sends the command to each robot in the unit. In some cases, the command must be modified slightly before it is sent—such as in a start command so that all the robots in the unit are not positioned on top of each other.

Map overlay information is loaded from overlay description files. This includes information including locations of control measures such as boundaries, assembly areas, lines of departure/lines of contact, attack positions, passage points, gaps, axes of motion, phase lines, battle positions, and objectives. A sample file is shown in Figure 14. Actual map coordinates are not included for simplicity.

```
SCENARIO "Demo-C"
SITE "Demo B site, Colorado"
ORIGIN X Y
CONTROL MEASURES:
Boundary Yankee x1 y1 x2 y2 ... xn yn
LDLC Echo x1 y1  x2 y2  ...  xn yn
AA Alpha x1 y1  x2 y2  x3 y3 ...  xn yn
ATK Bravo x1 y1  x2 y2  x3 y3 ...  xn yn
PP Charlie x y
Gap Delta x1 y1  x2 y2
Axis Foxtrot x1 y1  x2 y2  ...  xn yn
PL Gamma x1 y1  x2 y2  ...  xn yn
BP 1 x1 y1  x2 y2  ...  xn yn
BP 2 x1 y1  x2 y2  ...  xn yn
OBJ Zulu x1 y1  x2 y2  ...  xn yn
```

Figure 14: Example overlay description file without real map coordinates.

4.3.1 Script-based Military Scenario Executive Coordination Operator

A coordination operator has been developed which will interpret a set of steps necessary to accomplish a multiagent mission. Each step is a series of commands to send to the teams of robots involved all at the same time. The step is not complete until all the robots have completed their commands. The tools will load the steps and execute them in our simulation environment with minimal human interaction. Each robot will be configured with a set of behaviors to allow it to execute its commands with appropriate reactive control software.

The robots are configured with a set of behaviors which allow them to perform the necessary tasks (see the lower part of Figure 13). Currently we have behaviors for moving to a specified location, following a line, and occupying a position. Concurrently, the robot can be using any of several formations behaviors, such as line formation, column formation, diamond formation, wedge formation, and no formation. We are also working towards implementation several coordinated movement techniques such as traveling overwatch and bounding overwatch. The executive coordination node in each robot control software receives commands from the simulation system and activates the appropriate behaviors to accomplish the task.

There are two task description files: an overlay description file and a command description file. The overlay description file is a file containing the mission overlay information which was described in the previous section. The command description file contains background information and commands. The background information part includes such things as which overlay file to use, starting positions, and the composition of the units involved. The command information part includes a series of steps to be executed. Each step can be composed of several commands to be executed in parallel.

An example of a command description file is given in Figure 15. Notice that the file references an overlay file. Also notice the readable nature of the command language.

5. Communication in Multiagent Robotic Teams

Multiple cooperating robots are able to complete many tasks more quickly and reliably than one robot alone. Communication between the robots can multiply their capabilities and effectiveness, but to what extent? In this research, the importance of communication in robotic societies is investigated through experiments on both simulated and real robots. Performance was measured for three different types of communication for three different tasks. The levels of communication are progressively more complex and potentially more expensive to implement. For some tasks, communication can significantly improve performance, but for others inter-agent communication is apparently unnecessary. In cases where communication helps, the lowest level of communication is almost as effective as the more complex type. The bulk of these results are derived from thousands of simulations run with randomly generated ini-

```
MISSION NAME "Demo C simulation"
SCENARIO "Demo-C"
OVERLAY test.odl
SP Home 0 0
UNIT Wolf (Wolf-1 ROBOT BFV) (Wolf-2 UGV SSV HUMMER)
COMMAND LIST:
  0. UNIT Wolf START SP-Home 10 0
  1. UNIT Wolf MOVETO AA-Alpha FORMATION diamond traveling-overwatch
 1a. UNIT Wolf OCCUPY AA-Alpha FORMATION Column
  2. UNIT Wolf MOVETO ATK-Bravo FORMATION Column
  3. UNIT Wolf OCCUPY ATK-Bravo FORMATION Diamond
  4. UNIT Wolf MOVETO PP-Charlie FORMATION Column
  5. UNIT Wolf FOLLOW Gap-Delta FORMATION Column
  6. UNIT Wolf FOLLOW Axis-Foxtrot FORMATION Diamond Traveling-Overwatch
     PHASE-LINE PL-Gamma 06-10-94:23:10
  7. UNIT Wolf PASS-PHASE-LINE PL-Gamma
  8. UNIT Wolf-1 MOVETO BP-1 FORMATION Line Bounding-Overwatch AND
     UNIT Wolf-2 MOVETO BP-2 FORMATION Line Bounding-Overwatch
  9. UNIT Wolf-1 OCCUPY BP-1 FORMATION DIAMOND AND
     UNIT Wolf-2 OCCUPY BP-2 FORMATION DIAMOND
 10. UNIT Wolf-1 MOVETO OBJ-Zulu FORMATION Line Bounding-Overwatch AND
     UNIT Wolf-2 MOVETO OBJ-Zulu FORMATION Line Bounding-Overwatch
 11. UNIT Wolf OCCUPY OBJ-Zulu FORMATION Diamond
 12. UNIT Wolf STOP
```

Figure 15: Example Command Description File

tial conditions. The simulation results help determine appropriate parameters for the reactive control system which was ported for tests on Denning mobile robots.

Three different types of communication are evaluated in this research. Using a minimalist philosophy, the first type actually involves no direct communication between the agents. The second type allows for the transmission of state information between agents in a manner similar to that found in display behavior in animals. The third type (goal communication) requires the transmitting agent to recognize and broadcast the location of an attractor when one is located within detectable range.

Our research to date focuses on three tasks: foraging, consuming, and grazing. Foraging consists of searching the environment for objects (referred to as attractors) and carrying them back to a central location. Consuming requires the robot to perform work on the attractors in place, rather than carrying them back. Grazing is similar to lawn mowing; the robot or robot team must adequately cover the environment.

The impact of communication on performance in reactive multiagent robotic systems has been investigated through extensive simulation studies. Performance results for three generic tasks illustrate how task and environment can affect communication payoffs. Initial results from testing on mobile robots are shown to support the simulation studies.

The principal results for these tasks are:

- Communication improves performance significantly in tasks with little environmental communication.
- Communication is not essential in tasks which include implicit communication.
- More complex communication strategies offer little or no benefit over low-level communication.

Future work in this area is concerned with societal performance in fault-tolerant multiagent robotic systems; where unreliable communication may be present and the robotic agents have the potential for failure.

REFERENCES

[1] Arkin, R.C., "Cooperation without Communication: Multi-agent Schema Based Robot Navigation", *Journal of Robotic Systems*, Vol. 9(3), April 1992, pp. 351-364.

[2] Arkin, R.C., "The Multiple Dimensions of Action-Oriented Perception: Fission, Fusion, Fashion", Working notes of *AAAI 1991 Fall Symposium on Sensory Aspects of Robotic Intelligence*, Monterey, CA, Nov. 15-17, 1991.

[3] Arkin, R.C., "Reactive Control as a Substrate for Telerobotic Systems", *IEEE Aerospace and Electronics Systems Magazine*, Vol. 6, No. 6, June 1991, pp. 24-31.

[4] Arkin, R.C., "Motor Schema-Based Mobile Robot Navigation", *International Journal of Robotics Research*, Vol. 8, No. 4, August 1989, pp. 92-112.

[5] Arkin, R.C. and Ali, K.S., "Integration of Reactive and Telerobotic Control in Multi-agent Robotic Systems", *From animals to animats 4: Proc. 4th International Conference on Simulation of Adaptive Behavior*, Brighton, England, Aug. 1994, pp. 473-478

[6] Arkin, R.C., Balch, T., and Nitz, E., "Communication of Behavioral State in Multi-agent Retrieval Tasks", *Proc. 1993 IEEE International Conference on Robotics and Automation*, Atlanta, GA, May 1993, Vol. 3, pp. 588-594.

[7] Brooks, R., Maes, P., Mataric, M., and More, G., "Lunar Base Construction Robots", *IEEE International Workshop on Intelligent Robots and Systems (IROS '90)*, pp. 389-392, Tsuchiura, Japan, 1990.

[8] Cameron, J.M. and MacKenzie, D.C., "Specifying complex military scenarios", Georgia Institute of Technology, *Working paper, contact authors.*

[9] Fedor, C., "TCX: Task Communication," (User manual for TCX, available through the Robotics Institute), Carnegie Mellon University, Feb. 15, 1993.

[10] Hopcroft, J.E. and Ullman, J.D., *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, pp. 79, 1979.

[11] Mataric, M., "Minimizing Complexity in Controlling a Mobile Robot Population", *1992 IEEE International Conference on Robotics and Automation* , Nice, pp. 830-835.

6. Publications to Date Resulting from this Research

- FORMATION BEHAVIORS

  1. Balch, Tucker, 1994 "Motor schema-based formation control for multi-agent robot teams", *Working paper, contact author.*

- TELEAUTONOMOUS CONTROL

  1. Arkin, R.C. and Ali, K., 1994. "Integration of reactive and telerobotic control in multi-agent robotic systems", *Proc. Third International Conference on Simulation of Adaptive Behavior, (SAB94) [From Animals to Animats]*, Brighton, England, Aug. 1994, pp. 473-478.
  2. Ali, Khaled S., 1994 "Telop: Teleoperation of multi-agent reactive robotic systems", *Working paper, contact author.*

- MISSION SPECIFICATION

  1. MacKenzie, D. and Arkin, R.C., 1993. "Formal specification for behavior-based mobile robots", *Mobile Robots VIII*, Boston, MA, Nov. 1993, pp. 94-104.
  2. MacKenzie, Douglas C., 1994 "A Design Methodology for the Configuration of Behavior-Based Mobile Robots", *Ph.D. Thesis Proposal*, 1994.
  3. Cameron, Jonathan M. and MacKenzie, Douglas C., 1994 "Specifying complex military scenarios", *Working paper, contact authors.*

- INTER-ROBOT COMMUNICATION

  1. Balch, T. and Arkin, R.C., 1994. "Communication in reactive multiagent robotic systems", to appear in *Autonomous Robots*, Vol. 1, No. 1.

C- 36 - X25
#2

ANNUAL REPORT FY 1995

Flexible Reactive Control for Multi-Agent Robotic Systems in Hostile Environments

ONR/ARPA Grant #N00014-94-1-0215

Prepared by: Ronald C. Arkin (P.I.), Doug MacKenzie,
Tucker Balch, and Khaled Ali
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332
email: arkin@cc.gatech.edu
Fax: (404) 853-9376
Phone: (404) 894-8209

Contents

1. Introduction

This document constitutes the 1995 Annual Report for the ONR/ARPA Grant #N00014-94-1-0215 entitled Flexible Reactive Control for Multi-Agent Robotic Systems in Hostile Environments. This project is supported by ARPA's Real-time Planning and Control Program and has as a customer ARPA's UGV Demo II program. This second annual report reflects this year's accomplishments within the context of an overall three year research program.

The goals of this research are to produce intelligent, flexible, reactive behaviors and methods for specifying and communicating information between multiagent teams. In particular we have been studying three closely related subjects:

- Formation Control - to allow teams of robotic agents to move in a coordinated manner through a potentially hostile environment without interfering with other's active navigational behaviors.
- Teleautonomous Control of Multi-agent Teams - to allow a massive reduction in cognitive workload for the control of a group of robotic vehicles by permitting commands to be specified at the team level rather than at the individual agent level.
- Team Mission Specification Methods - to provide robust and flexible mission specification for reactive team military scenarios.

This report surveys the progress made to date for each of these areas. Each section includes a review of the previous year's accomplishments, the achievements for this fiscal year, technology transfer achieved, and goals for the upcoming year.

2. Formation Control

This portion of our research centers on the development of behaviors for formation maintenance in heterogeneous societies of mobile robots. The target testbed is a set of four robotic vehicles to be employed as a scout unit by the U.S. Army as part of ARPA's UGV Demo II program. Formations are important in this application as they allow the unit to utilize its sensor assets more efficiently than if the robots are arranged randomly.

2.1 Summary of Results for FY 94

Motor schemas, or primitive behaviors, for relative positional maintenance were developed and integrated with other navigational behaviors to help robots complete

navigational tasks while in formation. Four formations, based on existing military doctrine and two methods for determining correct vehicle position have been investigated. The formations developed are shown in Figure 1. These particular formations were selected because they are used by mechanized scout platoons on the battlefield. The robots in this work are heterogeneous in that each is assigned a unique identification number (ID). A robot's designated position in a given formation depends upon its ID. There are no other behavioral differences between the robots.

Formation control is one part of a more complex behavioral assemblage which includes other components for high-level task achievement. In addition to maintaining their position in formation, robots must also move to a specified goal location while avoiding collisions with obstacles.

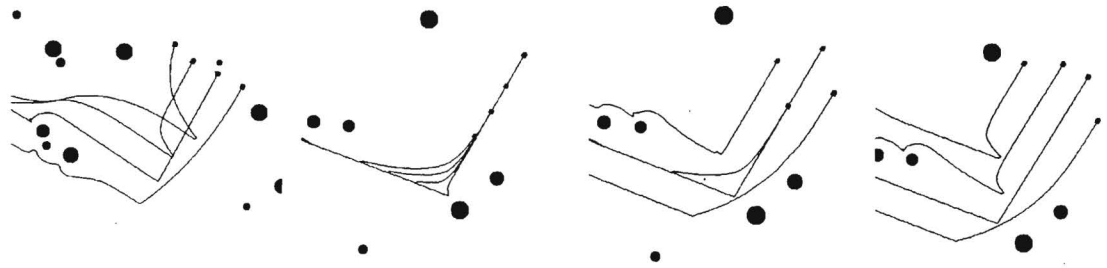


Figure 1: Implemented formations for four robots.
From left to right: line, column, diamond, wedge.

Formation maintenance is accomplished in two steps: first, a perceptual process, **detect-formation-position**, determines the robot's proper position in formation based on current environmental data; second, a motor schema **maintain-formation** generates a movement vector toward the correct location. Each robot must compute its proper position in the formation for each movement step.

Two techniques for formation position determination were investigated: unit-center and leader referenced. In unit-center referenced formations a unit-center is computed by averaging the x and y positions of all the robots involved in the formation. Each robot determines its own formation position relative to that center. In leader referenced formations each robot determines its formation position in relation to the lead robot (Robot 1). The leader does not attempt to maintain formation; the other robots are responsible for formation maintenance. These relationships are depicted in Figure 2. Arrows show how the formation positions are determined. Each arrow points *from* a robot *to* the associated reference. The perceptual schema **detect-formation-position** uses one of these references to determine the proper position for the robot. The spacing between robots is determined by the *desired spacing* parameter of **detect-formation-position**.

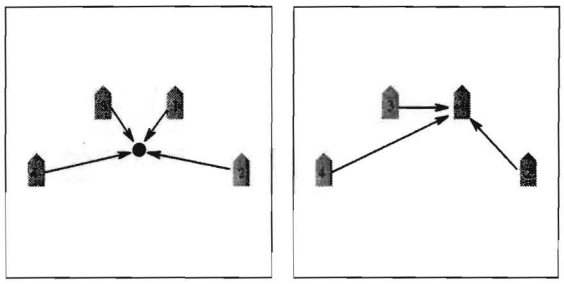


Figure 2: Formation position determined by different reference techniques. From left to right: unit-center, leader.

2.2 Research Accomplishments for FY 95

Research in formation behaviors continued in FY 1995 in three areas: first, a series of experiments in simulation were conducted to determine the best formations for particular situations; second, an expert system, called the Formation Expert was developed to help mission planners select appropriate formations for particular missions; third, the behaviors were ported to another robot architecture for use on Lockheed-Martin UGV's;

2.2.1 Quantitative Formation Results

The performance of a group of four simulated robots was evaluated quantitatively for both turning conditions and for navigation across an obstacle field. Results were generated using Georgia Tech's *MissionLab* robot simulation environment (see Section 4). The experimental results were presented at the International Conference on Multiagent Systems in July 1995 and are summarized here:

- For turning:
  - For turns in a unit-center-referenced formation, diamond formations perform best.
  - For turns in a leader-referenced formation, wedge and line formations perform best.
  - Performance for column formations is significantly worse than that for line, wedge and diamond formations.
- For travel across an obstacle field the best performance is found using column formations. This result reflects the fact that column formations present the smallest cross-section as they traverse the field. Once the lead robot offsets to avoid an obstacle, the others can follow in its "footsteps."
- Overall, unit-center-referenced formations fare better than leader-referenced formations.

2.2.2 The Formation Expert

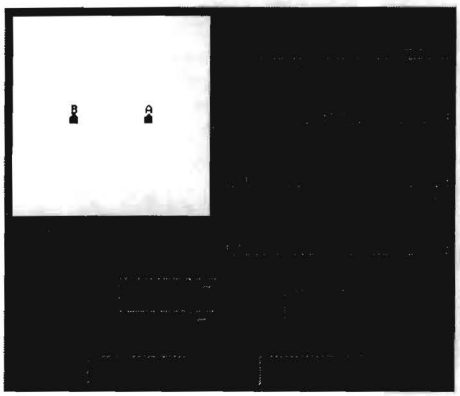


Figure 3: The Formation Expert GUI.

To aid UGV mission planners in selecting appropriate formations for particular situations, we developed an expert system called the Formation Expert. The Formation Expert automatically analyzes a mission plan then suggests parameter settings for formation behaviors based on that context. A user may adjust these recommended parameters with a graphical user interface (GUI). The displayed diagram of the formation is adjusted to reflect changes the user makes as he moves slider bars or pushes buttons (Figure 3).

In order to make its recommendations, the Formation Expert consults a *rule base*, which specifies conditions under which particular formations are appropriate. The rule base is an easy to understand text file that can be revised by a user to reflect new situations or better formations for certain situations. Also, since the Formation Expert is generic it may be easily adapted for use in other UGV domains as well.

2.2.3 Port to Lockheed-Martin UGV's

Behaviors for *line*, *wedge* and *column* formations for two robots were ported to the DAMN architecture for use on Lockheed-Martin UGVs. The primary difference between the DAMN architecture and the motor schema paradigm used at Georgia Tech centers on the use in DAMN of separate arbiters for heading and speed. The motor schema approach unifies these control components.

The unit-center-referenced approach was used for formation on UGVs. Recall that while in formation, each robot is assigned a specific position to maintain relative to the unit-center. On UGVs the unit-center is computed several times each second using GPS and communication equipment aboard the UGVs. After the GPS information is received and processed, each robot is able to determine how far out of position it is, and where it ought to move to get back into position.

Control commands which keep the robot in position are separated into fore-aft and side-side corrections. Fore-aft corrections are made by adjusting the robot's speed, while side-side corrections are made by adjusting the robot's heading. The size of speed and steering corrections depend on how far out of position the robot is. In general, the further out of position a robot is, the more aggressively it strives to move back into position.

These formation behaviors played a critical role in July 1995 at UGV Demo C. The event, held at Lockheed-Martin's facility in Denver, Colorado, included two demonstrations of formation; in the first Tech Demo, and in the final Integrated Demo. Figure 4 shows the UGVs in formation.

Figure 4: UGVs in formation.

2.3 Technology Transfer for FY 95

As mentioned above, formation behaviors were transferred to Lockheed-Martin for the ARPA-sponsored UGV project. The formation behavior software will continue to be used and expanded through the rest of the UGV Demo II project.

The Formation Expert has been incorporated into STX/MCU developed by Hughes STX, as well as the mission planning system developed for UGV Demo II by the University of Michigan. It is expected that other researchers on the project may adapt the Expert for use in their domains as well. The University of Texas at Arlington has utilized the formation behaviors in their research on sensor coordination between multiple robots. Finally, formation behaviors are included in the public distribution of MissionLab (see the section on Configuration Design).

2.4 Goals for FY 96

Research in formation control has not yet addressed various modes of robot failure. Communications, sensor, and motor failures can significantly impact a formation. Mechanisms to deal with these failures might include automatic reconfiguration of the formation (renumbering) and application of fault-tolerant communications strategies.

To date, formations have been demonstrated in simulation for four robots, but only for two actual mobile robotic vehicles at a time. We intend to upgrade one of the robots in our laboratory and integrate it with an existing team of two Denning robots. This will permit formation research with a three-robot team. Demo II will include formations of three and perhaps four robots. Additional consultation with Lockheed-Martin will include extending the existing software for these formations at Demo II.

3. Team Teleautonomy

This research concerns the development and implementation of methods to allow a human operator to control a team of robots. Our approach provides a mechanism to significantly reduce the human operator's cognitive and perceptual load by allowing the reactive system to deal with each robot's local control concerns. Two principal mechanisms to achieve this are by allowing the operator to act either as a constituent behavior of the society or to allow him/her to supervise the societal behavioral sets and gains, acting only as needed based upon observable progress towards task completion.

3.1 Summary of Results for FY 94

In this research, which is implemented in the TELOP system, the operator is allowed to control whole societies of agents; not one robot at a time, but rather controlling global behavior for the entire multiagent system. The end product is a simple way for a commander to control large numbers of constituent elements without concern for low-level details (which each of the agents is capable of handling by themselves).

Telerobotic control is facilitated by two methods. The first method allows the human operator to give directional information to the robots. The second method allows the operator to interactively adjust the parameters of the reactive behaviors, thereby changing the overall behavior of the robot society.

In the first method, the human operator give directional information to the robot team. He controls the output of a **teleautonomy** behavior by using an on-screen "joystick". The teleautonomy behavior then produces a vector output in the direction that the joystick was depressed and with a magnitude relative to the amount that the joystick was depressed. This vector is sent to each of the robots and is then summed

and normalized with the vectors from the other active behaviors on each robot [1]. The robot then executes the resultant vector.

In the second method, the operator interactively changes the overall behavior of the robot team by adjusting the parameters of the reactive behaviors. The human operator manipulates the behavioral parameters in terms of abstract personality traits. Making parameter changes in terms of personality traits allows a user, with no knowledge about the underlying behaviors and their parameters, to effectively modify the robots' behavior. In our current system, the abstract parameters include *Aggressiveness* and *Wanderlust*. The value of an abstract parameter controls the values of several individual low-level parameters. The operator uses slider bars to modify the value of an abstract personality trait.

A set of experiments was performed to test the usefulness of the **teleautonomy** behavior for certain tasks. These tasks include foraging, grazing, and herding. For the foraging task, if teleoperation is used wisely, it can significantly lower the total number of steps required to complete the task by greatly reducing the time spent in the *forage* state (i.e., the number of steps that the robots spend looking for attractors). For the grazing task, teleoperation was not significantly better than no teleoperation. For the herding task, the **teleautonomy** behavior was discovered to be an acceptable tool, but possible improvements were determined.

A set of usability tests was conducted on the interface for TELOP. These tests yielded substantial information for making the interface more helpful and usable. Changes to the Telop interface were made based on the results of these usability tests.

3.2 Research Accomplishments for FY 95

The TELOP system has been adapted to run on real robots. This involved integrating it with the *MissionLab*[1] [3] system. TELOP can now be used both in simulation and on real robots through *MissionLab*.

Further experiments were carried out on real robots. These experiments tested the use of both the **teleautonomy** behavior and the abstract parameters. Two generic tasks were tested on real robots. These tasks include directing the robots out of a box canyon and squeezing the robots through small spaces. TELOP was tested on a pair of Denning MRV-2 mobile robots, each about three feet tall with a diameter of 32 inches. Each robot is equipped with a ring of 24 ultrasonic sensors and shaft encoders. A Sun Sparcstation 5 served as the base station, running TELOP through *MissionLab*. The base station communicates with the robots using FreeWave radio links. The base station and human operator were on the third floor of the Manufacturing Research

[1] *MissionLab* is a system for specifying and simulating multiagent robotic missions. *MissionLab* takes high-level military-style plans and executes them with teams of real or simulated robotic vehicles. The source code for *MissionLab* is available on the World Wide Web at the location http://www.cc.gatech.edu/ai/robot-lab/research/MissionLab/MissionLab.html.

Center at Georgia Tech, and the robots were running on the first floor. The feedback to the operator consisted of the graphical depiction of the robots actions relayed in real-time by *MissionLab* and walkie-talkie communication between the operator and a human who was on the first floor observing the robots.

3.2.1 Tasks

Multiagent teleautonomy was tested on a pair of Denning mobile robots for two tasks including navigating the robots out of a box canyon and squeezing them through a tight space. In both tasks, the robots were using the **teleautonomy**, **avoid-static-obstacle**, **avoid-robot**, **move-to-goal**, **noise**, and column **formation** [2] behaviors.

In the first task, a box canyon, constructed from chairs, was set up in the room. The robots were started on the side of the room facing the opening of the box canyon. The robots were instructed to go to a location on the other side of the box canyon, such that the box canyon lay directly along the straight-line path from the start location to the destination. Since the robots operate purely reactively in this mode, they normally would get stuck in the box canyon and need to be helped out by the human operator.

The task setup for the second task was the same as the first, except that the box canyon had a gap in it. The gap was sufficiently small so that the robots could not squeeze through it with the usual default gain setting for the **avoid-static-obstacle** behavior. The robots were provided with this default setting at the start of the task, so they would normally become stuck in the box canyon. The human operator should then be able to increase the robots' *Aggressiveness* to forcibly squeeze them through the gap.

3.2.2 Experimental Results

For the two tasks involving the two Denning mobile robots, the runs were videotaped, and a screen capture was taken of the tracking of the actual robots' movement from the *MissionLab* interface. *MissionLab* monitors the movement of the robots using information from their shaft encoders. This movement is plotted over an underlay depicting the task environment.

For the box canyon task, as expected, the two robots got stuck in the box canyon while heading to the destination location (see Figure 5a). The human operator was able to use the on-screen joystick to steer the robots out of the box canyon, and around the side of it (see Figure 5b). After the robots were completely around the lip of the box canyon and were no longer in any danger of falling back into it, the operator released the joystick. Then the robots continued on to their destination autonomously. A trace of the robots' movement is shown in Figure 6.

For the squeezing task, the robots became trapped within the box canyon while heading to their destination (Figure 7a). This is a result of the default gain for the

(a) (b)

Figure 5: Box canyon task: (a) The robots are stuck in the box canyon. (b) The robots are being maneuvered out of the box canyon using the teleautonomy behavior.
The camera was located 4 floors above the robots, giving a birds-eye view of the action. The robots have circles of white tape on top of them to make them more visible.

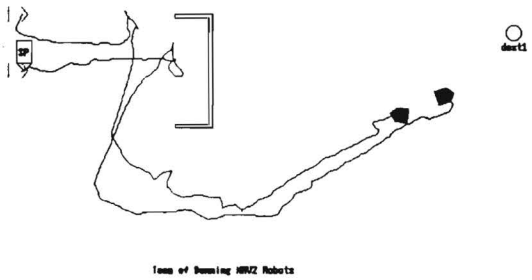
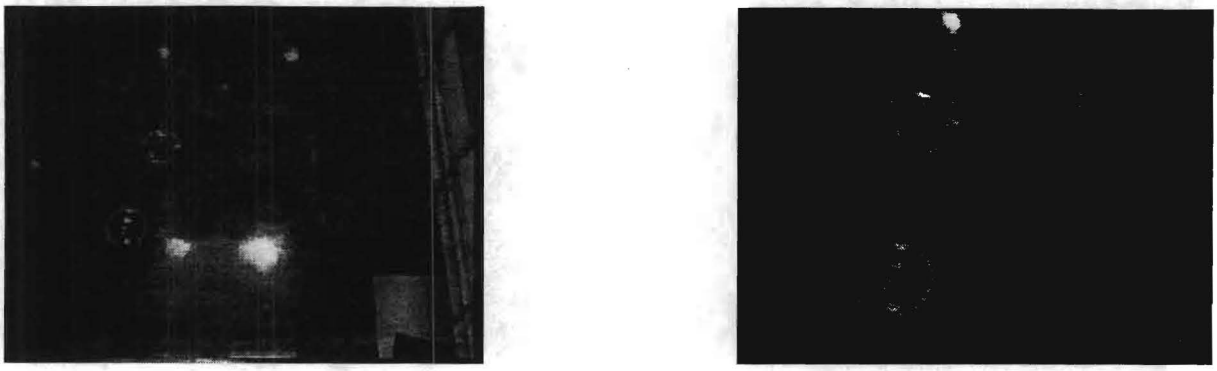


Figure 6: *MissionLab* trace of the robots movement during the box canyon task. The robots became stuck in the box canyon, and were then herded around the side of the box canyon by the human operator using the teleautonomy behavior.



(a) (b)

Figure 7: Squeezing task: (a) The robots are stuck in the box canyon with a gap. (b) The robots are being squeezed through the gap in the box canyon by making them more aggressive.
The camera was located 4 floors above the robots, giving a birds-eye view of the action.

avoid-static-obstacle behavior being set too high and the gain for the **move-to-goal** behavior set too low for the robots to pass through the gap. The human operator slowly increased the robots' aggression until the robots successfully squeezed through the passageway (see Figure 7b). A trace of the robots' movement is shown in Figure 8.

When increasing the aggression of the robots, the operator should make small incremental increases until the robots squeeze through the small space. Then the operator should decrease the aggression again. If, however, the operator increases the aggression too much, the robots may charge through obstacles on their way to the goal (although this may be consistent with what the operator wants).

3.3 Technology Transfer for FY 95

The TELOP system has recently been integrated with the ARPA UGV Demo II architecture using the STXmcu mission control system for use on teams of HMMWVs. The teleautonomy behavior was demonstrated at a technical demo during Demo C of the UGV project in the summer of 1995. The teleautonomy behavior and the personality sliders are intended to be used in Demo II during the summer of 1996.

We made Telop compatible with the DAMN arbiters, which are the arbitration systems used on the vehicles in the UGV project. This required changing the implementation of TELOP, but the underlying ideas remain the same.

The main window of the interface was modified in appearance and functionality to accommodate the differences between the combination mechanism used in our schema system and the arbitration mechanism used in the DAMN arbiters. The joystick now

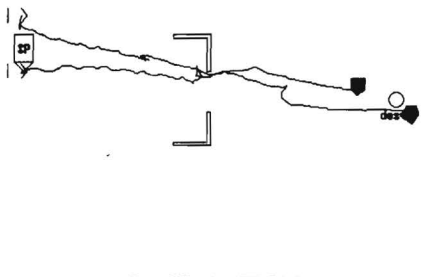


Figure 8: *MissionLab* trace of the robots movement during the squeezing task. The robots became trapped in the box canyon with a gap, but the operator squeezed them through the passage by increasing their aggressiveness.

controls the direction and the speed of the society of robots. When the joystick is used, the interface sends messages (via IPT) to the teleautonomy behaviors. These messages indicate the direction and the speed that the society of robots should move in. A slider bar was added to adjust the weight that the DAMN turn arbiter uses when considering the votes from the teleautonomy behavior. The communication with the DAMN arbiters to change the weights is done using IPT.

We implemented the teleautonomy behavior in the same form as the other DAMN behaviors. There are actually two teleautonomy behaviors, one for turning and the other for speed. Each robot runs one teleautonomy turn behavior and one teleautonomy speed behavior. When the behaviors receive a message from the Telop interface indicating a direction and speed for the society to move in, the behaviors compute a turn radius and speed based on the situation of the particular robot that the behavior is resident on. Then the behaviors send their votes for this turn radius and speed to the DAMN arbiter. The votes for turn radii are actually distributed among all the possible turn radii as a gaussian centered at the desired turn radius.

An interface for modifying the default parameters that the Telop interface uses when it is started by either MRPL or the STX interface was also created. This default editor also allows modification of the system defaults for the parameters.

Personality-sliders for controlling behavioral parameters in terms of abstract personality traits were implemented. So far, one personality-slider for the Aggressiveness traithas been developed. We will be implementing more if we receive a go-ahead at the next UGV Demo II Workshop. The personality sliders adjust the weights that the DAMN arbiter uses when considering the votes from each voting behavior. Each personality-slider adjusts certain weights that will cause the robots to act more or less like the personality for which the slider-bar is named. For instance, the *Aggressiveness* slider-bar adjusts the weights of the **avoid-obstacle** and **follow-path** behaviors.

When a slider bar is moved, the Telop interface sends messages to the appropriate arbiters on each vehicle. The messages instruct the arbiters to change the weights that are used when considering the votes of a particular behavior. The interfaces are implemented using UIM/X for ease of integration with the other interfaces in the Demo II program.

In addition, TELOP has been integrated with the *MissionLab* system, which is available for downloading on the World Wide Web. The user can edit a file to dynamically create new slider-bars for the personality traits that he or she wishes to manipulate without having to recompile the system. The desired personality slider-bars are indicated using a simple syntax. This allows other researchers to experiment with the TELOP system.

3.4 Goals for FY 96

Further work on the control of abstract personality traits will be undertaken, with the intention that they be utilized in UGV Demo II. To date, we have implemented one personality slider for aggressiveness. Other suitable personality sliders will be created as needed.

The **teleautonomy** behavior was demonstrated at Demo C. We expect that both the **teleautonomy** behavior and the personality sliders to be part of Demo II.

Further experiments and usability studies will be conducted to evaluate the teleautonomous control concepts and interfaces. These studies will compare the methods of teleautonomous control that we have developed with other forms of telerobotic control along a prescribed set of dimensions.

4. Configuration Design Support for Mission Specification

Configuration design tools have been developed which support the graphical construction of abstract configurations which can then be bound to the UGV architecture and compiled into MRPL code executable on the UGV's. Capabilities of the *MissionLab* toolset include a graphical configuration editor, MRPL and C++ code generators, a multiagent simulator and an integrated operator console. The mission scenario language and corresponding interpreter permit the specification of complex multiagent missions in a structured user-friendly language. The combination of the mission scenario language and graphical configuration editor should allow operators with minimal training (such as soldiers) to construct complex multiagent missions.

4.1 Summary of Results FY 94

The *Societal Agent* architecture was developed to capture the recursive composition of configurations. Specifying a reactive behavioral configuration for use by a multia-

gent team executing a mission requires both a careful choice of the behavior set and the creation of a temporal chain of behaviors which executes the mission. This difficult task is simplified by applying an object-oriented approach to the design of the mission using a methodology called *temporal sequencing*. Temporal sequencing partitions the mission into discrete operating states with perceptual triggers causing transitions between those states. Several smaller independent configurations (assemblages) can then be created which each implement one state. Assemblages consist of groups of basic behaviors and coordination mechanisms that allow the group to be treated as a single, coherent behavior. Upon instantiation, the assemblage is parameterized based upon the requirements of these specific mission requirements. These assemblages can be re-parameterized and used in other states within this mission or archived as high level primitives for use in subsequent projects.

The *MissionLab* system, an implementation of the *Societal Agent* architecture, supports graphical construction of configurations using a visual configuration editor. This editor, *CfgEdit*, supports the recursive construction of reusable components at all levels, from primitive motor behaviors to societies of cooperating robots by allowing creation of coordinated assemblages of components which are then treated as atomic higher-level components available for later reuse. The Configuration Editor allows deferring commitment (binding) to a particular robot architecture or specific vehicles until the configuration has been developed. This explicit binding step simplifies developing a configuration which may be deployed on one of several vehicles which may each require use of a specific architecture. The process of retargeting a configuration to a different vehicle when the available vehicles or the system requirements change is similarly eased. The capability exists to generate either MRPL code for the ARPA UGV architecture or C++ code targeted for the Autonomous Robot Architecture (AuRA) which is executable within the *MissionLab* system. The AuRA executables drive both simulated robots and several types of Denning vehicles (DRV-1, MRV-2, MRV-3). The architecture binding process determines which compiler will be used to generate the final executable code, as well as which libraries of behavior primitives will be available for placement within the editor.

The mission scenario language and corresponding interpreter permit the specification of complex multiagent missions in a structured, relatively user-friendly, language. The mission coordination operator has the expressive power of a finite state machine but allows the user to specify the sequence of steps making up the mission using a domain-specific language with high-level primitives and mnemonic names. At run time, the mission coordination operator communicates with the operator console to allow the mission to be entered interactively or predefined missions to be executed from saved files.

Using these tools, various multiagent missions have been demonstrated in simulation and on our Denning robots. The *MissionLab* toolset was demonstrated as part of UGV demo C and has been made publicly available in source code form via anonymous FTP or WWW access.

4.2 Research Accomplishments for FY 95

The last year has seen completion of the *MissionLab* toolset. The *MissionLab* robot software development system provides support for users in the various stages of mission development (*e.g.*, behavior implementation, assemblage construction, and mission specification). The primitive behavior implementor must be familiar with the particular robot architecture in use and a suitable programming language such as C++. The assemblage constructor uses a library of behaviors to build skill assemblages using the graphical configuration editor. This allows visual placement and connection of behaviors without requiring programming knowledge. However, the construction of *useful* assemblages still requires knowledge of behavior-based robot control. Specifying a configuration for the robot team consists of selecting which of the available skills are useful for the targeted environments and missions. Specification of the mission sequence can occur at run-time using a domain-specific structured language. Military terminology and nomenclature are used in *MissionLab* to facilitate specification of missions by military users unfamiliar with robot control techniques. The overall philosophy, however, is by no means restricted to this application domain.

Consider specification of a configuration implementing a janitorial task for a mobile robot. Specifically, the robot should wander around looking for empty soda cans, pick them up, wander around looking for a recycling basket, and then place the can into the basket. Figure 9 is a schematic representation of an FSA for such a robotic trash collector constructed using *CfgEdit*. The circles represent the possible operating states with the label indicating the assemblage agent active during that state. The arcs are labeled with the perceptual triggers causing the transition where relevant. Powering up in the *start* state, the robot begins to wander looking for a suitable soda can, operating in the *Look_for_can* state. When a can is perceived, the *Pick_up_can* state is activated and if the can is successfully acquired, a transition to the *Look_for_basket* state occurs. Loss of the can in either of these states causes the FSA to fall back to the previous state and attempt recovery. When a recycling basket is located, the *Put_can* state becomes active and the can is placed in the basket. A transition back to the *Look_for_can* state repeats the process.

Figure 10 shows *MissionLab* executing a simulation. The large area with various things drawn in it is the main display area. Within the display area robots, obstacles, and other features are visible. The solid round black circles are obstacles. The four robots are moving across the middle of the display area in roughly a diamond formation. More details about the type of mission displayed in the figure are explained in the next section. The command interface in the lower right part of Figure 10 allows the operator to control the execution of the mission. The steps of the mission are displayed as they execute.

Specifically, several software projects were finished: The configuration editor *CfgEdit* was rewritten to support the recursive construction implicit in the *Societal Agent* architecture. It now serves as the focal point for the *MissionLab* integrated development

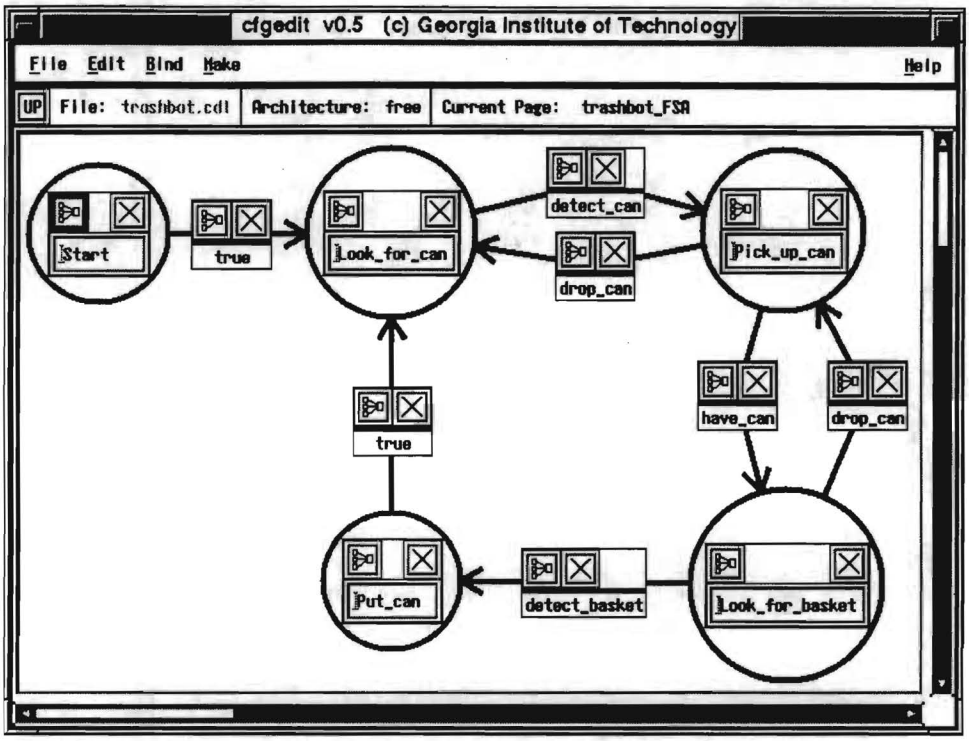


Figure 9: FSA for a trash collecting robot

environment and spawns other parts of the toolset as the user requires them. It allows graphical construction and visualization of the configuration description language (CDL) descriptions which can then be compiled into the selected output format. A conversion to the ARPA UGV standard IPT communications package was completed. The robot executables now communicate to the simulation server and operator console using IPT. Three internal compilers have been written. Two CDL compilers generate either the MRPL specifications used in the UGV program or a CNL (configuration network language) description of the input configuration. The CNL compiler can then translate this second type of output into C++ code using the AuRA architecture. These AuRA-style executables can then be run within the *MissionLab* operator workstation, freely mixing simulated and real robots.

4.3 Technology Transfer for FY 95

For ARPA UGV Demo C, Georgia Tech demonstrated the *MissionLab* mission specification and configuration software as part of a Technology Demo. This demo involved collaboration of Georgia Tech and the University of Texas at Arlington (UTA) in a joint demonstration. The *MissionLab* system has been adopted and extended by UTA to help verify their sensing algorithms for this joint tech demo. Additional requests to use *MissionLab* have come from the Naval Research Laboratories and

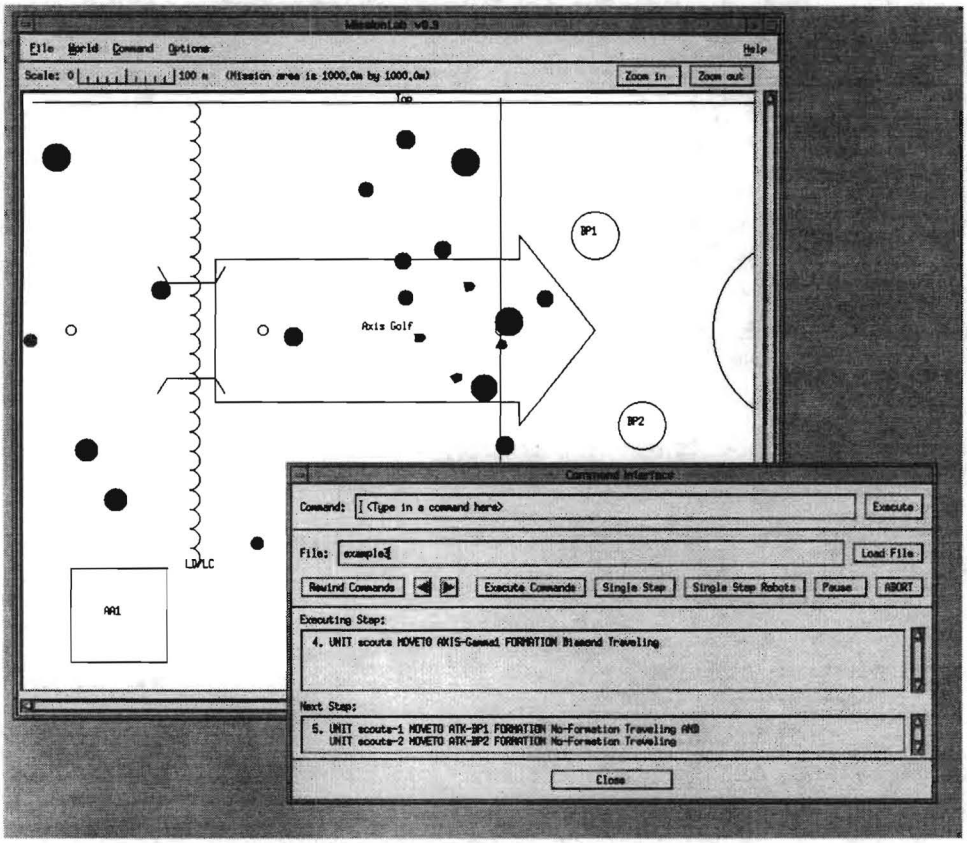


Figure 10: Example scenario in *MissionLab*

a joint US/Mexico research effort. The system is available through the Internet (http://www.cc.gatech.edu/ai/robot-lab/research/MissionLab) in both source and SPARC executable form.

4.4 Goals for FY 96

The *MissionLab* system is undergoing rapid evolution as the developer prepares to conduct usability studies on the system in support of his Ph.D. Dissertation. A goal of the studies relevant to this venue is to show that the toolset allows operators unable to specify missions using conventional means (such as soldiers) to conduct complex multiagent missions with only minimal training. It is intended that the evaluations will include testing by military personnel and military students here at Georgia Tech to allow direct verification.

Another release of *MissionLab* is planned for early 1996 to incorporate changes leading up to the usability studies. This will include advanced mission specification techniques, robot software configuration tools, and reusable control software libraries, as well as advanced team teleautonomous control concepts. Initial versions of many of these components were part of the 7/95 release of MissionLab. The next release will improve and complete the existing components, adding many new capabilities and simplifying the duties of the operator while making the system more powerful and robust.

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] Balch, T. and Arkin, R.C., "Motor Schema-based Formation Control for Multiagent Robot Teams", *1995 International Conference on Multiagent Systems*, San Francisco, CA, pp. 10-16, 1995.

[3] MacKenzie, D., Cameron, J., Arkin, R., "Specification and Execution of Multiagent Missions", *Proc. 1995 Int. Conf. on Intelligent Robotics and Systems IROS '95*, Pittsburg, PA, Vol. 3, pp. 51-58, 1995.

5. Publications to Date Resulting from this Research

- FORMATION BEHAVIORS

1. Balch, T. and Arkin, R.C., 1995. "Motor Schema-based Formation Control for Multiagent Robot Teams", *1995 International Conference on Multiagent Systems*, San Francisco, CA, pp. 10-16.

- TELEAUTONOMOUS CONTROL

1. Arkin, R.C. and Ali, K., 1994. "Integration of reactive and telerobotic control in multi-agent robotic systems", *Proc. Third International Conference on Simulation of Adaptive Behavior, (SAB94) [From Animals to Animats]*, Brighton, England, Aug. 1994, pp. 473-478.
2. Ali, Khaled S., 1994 "Telop: Teleoperation of multi-agent reactive robotic systems", *Working paper, contact author.*
3. Ali, K.S. and Arkin, R.C., 1995. "Multiagent Teleautonomous Behavioral Control", Submitted to *Robotica*, 1995.

- MISSION SPECIFICATION

1. MacKenzie, D. and Arkin, R.C., 1993. "Formal specification for behavior-based mobile robots", *Mobile Robots VIII*, Boston, MA, Nov. 1993, pp. 94-104.
2. MacKenzie, Douglas C., 1994. "A Design Methodology for the Configuration of Behavior-Based Mobile Robots", *Ph.D. Thesis Proposal*, 1994.
3. Cameron, Jonathan M. and MacKenzie, Douglas C., 1994. "Specifying complex military scenarios", *Working paper, contact authors.*
4. MacKenzie, D. and Arkin, R.C., 1995. "Specification and Execution of Multiagent Missions", *Proc. 1995 conference on Intelligent Robots and Systems (IROS'95)*, August 1995, Pittsburgh, PA, Vol. 3, pp. 51-58.
5. Douglas C. MacKenzie, Jonathan M. Cameron, and Ronald C. Arkin, 1995. "Specification and Execution of Multiagent Missions", Submitted to *Autonomous Robots.*

- INTER-ROBOT COMMUNICATION

1. Balch, T. and Arkin, R.C., 1994. "Communication in reactive multiagent robotic systems", to appear in *Autonomous Robots*, Vol. 1, No. 1.
2. Arkin, R.C. and Balch, T., 1995., "Communication and Coordination in Reactive Robotic Teams", to appear in *Coordination Theory and Collaboration Technology*, ed. G. Olson, J.B. Smith, and T. Malone, 1995.

3. Arkin, R.C. and Balch, T., 1995, "AuRA: Principles and Practice", submitted to *Journal of Experimental and Theoretical Artificial Intelligence*, 1995.

C-36-X25
#3

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	970327	931115 - 970331

4. TITLE AND SUBTITLE
Flexible Reactive Control for Multi-agent Robotic Systems in Hostile Environments

5. FUNDING NUMBERS
Contract No. N00014-94-1-0215

6. AUTHOR(S)
Ronald Arkin, Doug MacKenzie, Tucker Balch and Khaled Ali

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Dr. Ronald Arkin
Georgia Institute of Technology
College of Computing
801 Atlantic Drive
Atlanta, GA 30332-0280

8. PERFORMING ORGANIZATION REPORT NUMBER
C-36-X25

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Office of Naval Research, Ballston Tower One
800 North Quincy Street
Arlington, VA 22217-5660

Ms. Teresa McMullen, Scientific Officer Code: 333

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES COR:

12a. DISTRIBUTION/AVAILABILITY STATEMENT
May not be released by other than sponsoring organization without approval of Office of Naval Research

12b. DISTRIBUTION CODE

13. Abstract: This document constitutes the Final Report for the ONR/DARPA Grant #N00014-94-1-0215 entitled Flexible Reactive Control for Multi-agent Robotic Systems in Hostile Environments. This project is supported by DARPA's Real-time Panning and Control Program and has a customer DARPA's UGV DemoII program. This report reflects the project's accomplishments within the context of an overall three year research program. The goals of the research were to produce intelligent, flexible, reactive behaviors and methods for specifying and communicating information between multiagent teams.

14. SUBJECT TERMS
Formation Controls, Teleautonomous Control of Multi-agent Teams

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unlimited

FINAL REPORT

Flexible Reactive Control for Multi-Agent Robotic Systems in Hostile Environments

ONR/DARPA Grant #N00014-94-1-0215

Prepared by: Ronald C. Arkin (P.I.)
Khaled Ali, Tucker Balch, Darrin Bentivegna,
Zhong Chen, and Doug MacKenzie
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332
email: arkin@cc.gatech.edu
Fax: (404) 853-9376
Phone: (404) 894-8209

Contents

1. Introduction

This document constitutes the Final Report for ONR/DARPA Grant #N00014-94-1-0215 entitled *Flexible Reactive Control for Multi-Agent Robotic Systems in Hostile Environments.* This project wass supported by DARPA's Real-time Planning and Control Program and had as a customer DARPA's UGV Demo II program. This report reflects this project's accomplishments within the context of an overall three year research program.

The goals of this research were to produce intelligent, flexible, reactive behaviors and methods for specifying and communicating information between multiagent teams. In particular we have studied three closely related subjects:

- Formation Control - to allow teams of robotic agents to move in a coordinated manner through a potentially hostile environment without interfering with other's active navigational behaviors.
- Teleautonomous Control of Multi-agent Teams - to allow a massive reduction in cognitive workload for the control of a group of robotic vehicles by permitting commands to be specified at the team level rather than at the individual agent level.
- Team Mission Specification Methods - to provide robust and flexible mission specification for reactive team military scenarios.

2. Formation Control

We have developed a behavior-based approach to robot formation-keeping. Since behavior-based systems integrate several goal oriented behaviors simultaneously, systems using this technique are able to navigate to waypoints, avoid hazards and keep formation at the same time. The initial target for this work is a team of robotic vehicles to be fielded as a scout unit by the U.S. Army. Formation is important in this and other military applications where sensor assets are limited. Formations allow individual team members to concentrate their sensors across a portion of the environment, while their partners cover the rest. Air Force fighter pilots for instance, direct their visual and radar search responsibilities depending on their position in a formation [7]. The approach is potentially applicable in many other domains such as search, agricultural coverage tasks, security patrols and so on.

Behaviors for four formations and two formation reference types were implemented and evaluated. The behaviors were demonstrated successfully in the laboratory on holonomic robots, and outdoors on 4-wheel-drive HUMMERs. In the course of these evaluations, the approach was implemented on two different reactive robotic architectures, AuRA and the UGV Demo II Architecture. The AuRA implementation is

conceptually simpler and applicable to holonomic robots, while the UGV implementation addresses the additional complexity of non-holonomic vehicle control. Separate experiments in simulation evaluated the utility of the various formation types and references in turns and across obstacle fields.

Each robot's position in formation depends on a unique identification number (ID). This is important in applications where one or more of the agents is dissimilar. In Army scout platoons for instance, the leader is not usually at the front of the formation, but in the middle, or to one side.

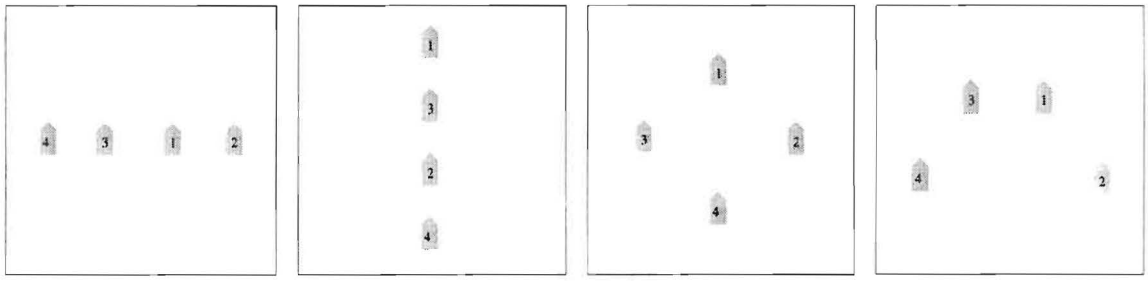


Figure 1: Formations for four robots (from left to right: line, column, diamond, wedge)

The formation behaviors were implemented as *motor schemas*, in the AuRA architecture [1], and as steering and speed behaviors in the UGV Demo II architecture [6]. In both cases, the individual behaviors run as concurrent asynchronous processes with each behavior representing a high-level behavioral intention of the agent. Perceptions are directly translated into a response vector in AuRA, or as turning or speed votes on the UGV. Readers are referred to [1] and [6] for more information on schema-based reactive control and the UGV Demo II architecture.

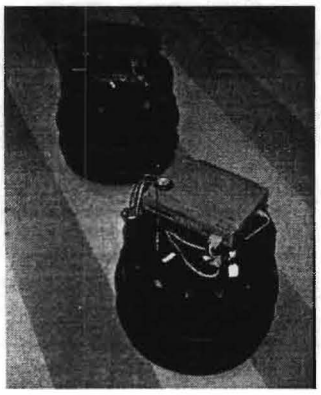


Figure 2: Shannon and Sally, two Nomadics Technologies Nomad 150 mobile robots. The formation behaviors demonstrated on Denning MRV-3s and DARPA's Unmanned Ground Vehicles are now under evaluation on these new robots.

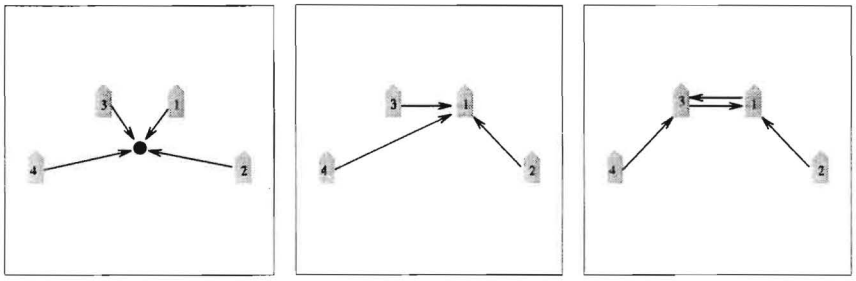


Figure 3: Formation position determined by various reference techniques (from left to right: unit-center, leader, neighbor)

Initial results of research at Georgia Tech for simulated robot teams appeared in [3]. Since then the approach has been demonstrated on on two types of mobile robots (Denning Mobile Robotics MRV-3s, and DARPA's Unmanned Ground Vehicles) and two robotic architectures (AuRA and the UGV Demo II Architecture). The system has also been ported to Nomadics Technologies Nomad 150 robots (Figure 2).

Several formations for a team of four robots are considered (Figure 1):

- *line* - where the robots travel line-abreast.
- *column* - where the robots travel one after the other.
- *diamond* - where the robots travel in a diamond.
- *wedge* - where the robots travel in a "V".

These formation types are used by U.S. mechanized scout platoons on the battlefield [2]. For each formation, each robot has a specific position (based on its ID). Figure 1 shows the formations and robots' positions within them.

Formation maintenance is accomplished in two steps: first, a perceptual process, **detect-formation-position**, determines the robot's proper position in formation based on current environmental data; second, the motor process **maintain-formation**, generates motor commands to direct the robot toward the correct location. In the case of motor schema control, the command is a movement vector towards the correct location. For the UGV Demo II Architecture separate "votes" are cast for steering and speed corrections towards the formation position. Motor commands for each architecture are covered in more detail below.

Each robot computes its proper position in the formation for each movement step. Three techniques for formation position determination have been identified:

- **Unit-center-referenced**: a unit-center is computed by averaging the x and y positions of all the robots involved in the formation. Each robot determines its own formation position relative to that center.

- **Leader-referenced**: each robot determines its formation position in relation to the lead robot (Robot 1). The leader does not attempt to maintain formation; the other robots are responsible for formation maintenance.
- **Neighbor-referenced**: each robot maintains a position relative to one other predetermined robot.

These relationships are depicted in Figure 3. Arrows show how the formation positions are determined. Each arrow points *from* a robot *to* the associated reference. The perceptual schema **detect-formation-position** uses one of these references to determine the position for the robot. Spacing between robots is determined by the *desired spacing* parameter of **detect-formation-position**.

Since AuRA and the UGV Demo II architectures utilize different formulations for perceptual and motor processes, they are examined separately below.

2.1 Motor Schema-based Formation Control

Several motor schemas, **move-to-goal**, **avoid-static-obstacle**, **avoid-robot** and **maintain-formation** implement the overall behavior for a robot to move to a goal location while avoiding obstacles, collisions with other robots and remaining in formation. An additional background schema, **noise**, serves as a form of reactive "grease", dealing with some of the problems endemic to purely reactive navigational methods [1]. Each schema generates a vector representing the desired behavioral response (direction and magnitude of movement) given the current sensory stimuli provided by the environment. A gain value is used to indicate the relative importance of the individual behaviors. The high-level combined behavior is generated by multiplying the outputs of each primitive behavior by its gain, then summing and normalizing the results.

Once the desired formation position is known, the **maintain-formation** motor schema generates a movement vector towards it. The vector is always in the direction of the desired formation position, but the magnitude depends on how far the robot is away from it. Figure 4 illustrates three zones, defined by distance from the desired position, used for magnitude computation. The radii of these zones are parameters of the **maintain-formation** schema. In the example, Robot 3 attempts to maintain a position to the left of and abeam Robot 1. Robot 3 is in the controlled zone, so a moderate force towards the desired position (forward and right) is generated by **maintain-formation**. The magnitude of the vector is computed as follows:

- **Ballistic zone**: the magnitude is set at its maximum, which equates to the schema's gain value.
- **Controlled zone**: the magnitude varies linearly from a maximum at the farthest edge of the zone to zero at the inner edge.
- **Dead zone**: in the dead zone vector magnitude is always zero.

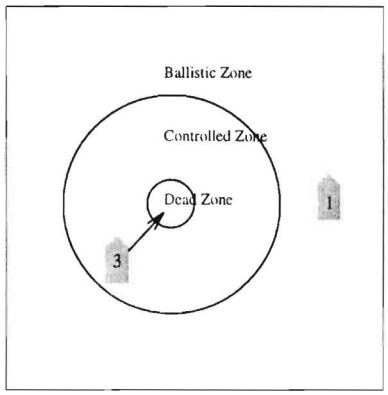


Figure 4: Zones for the computation of **maintain-formation** magnitude

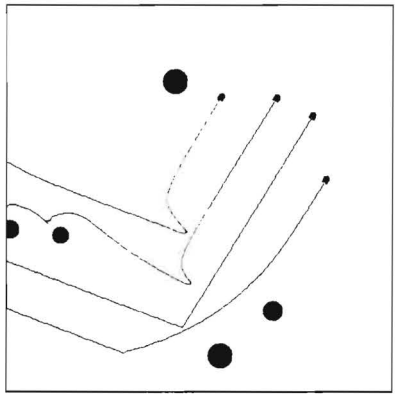


Figure 5: Typical simulation run showing four robots in a leader-referenced wedge formation executing a 90 degree left turn.

2.2 Motor Schema Results in Simulation

Experiments in simulation evaluated the utility of the various formation types and references in turns and across obstacle fields. For 90 degree turns, the diamond formation performs best when the unit-center-reference for formation position is used, while wedge and line formations work best when the leader-reference is used. For travel across an obstacle field, the column formation works best for both unit-center- and leader-referenced formations. In most cases, unit-center-referenced formations perform better than leader-referenced formations.

2.3 Motor Schema Results on Mobile Robots

MissionLab is designed so that at runtime a researcher may choose between a simulated run, or a run on physical robots. So far, the system can command Denning MRV-3, MRV-2 and DRV robots. Extensions to MissionLab provide control of No-

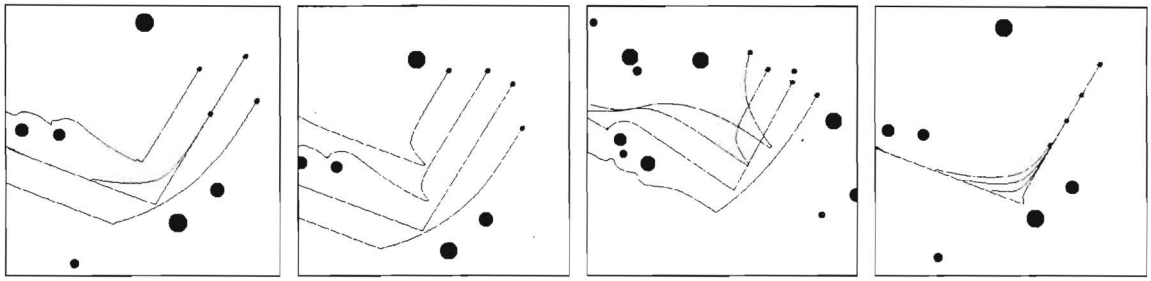


Figure 6: Four robots in leader-referenced *diamond*, *wedge*, *line* and *column* formations.

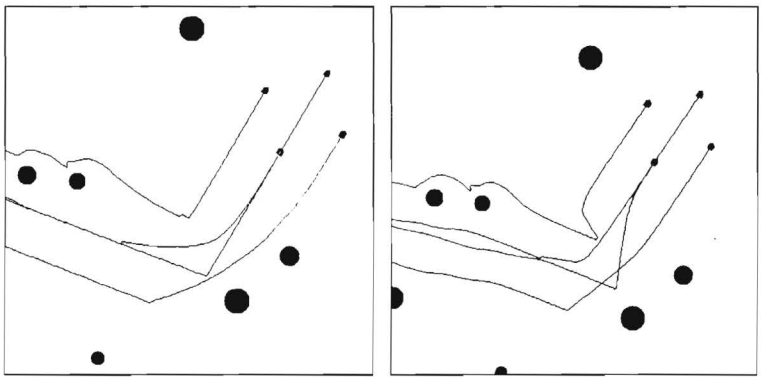


Figure 7: Comparison of leader-referenced (left) and unit-center-referenced (right) diamond formations.

madics Research Nomad-150 robots and a HUMMER 4-wheel drive vehicle instrumented for robotic use at Georgia Tech. Formation experiments on the latter two systems are in progress.

The behaviors have been tested on Denning MRV-3 robots, Ren and Stimpy, in conjunction with the teleoperation experiments covered in Section 3..

2.4 Formation Control for the UGV Demo II Architecture

UGV Demo II is an DARPA-funded project aimed at fielding a robotic scout platoon for the Army. Each Unmanned Ground Vehicle (UGV) is a 4-wheel-drive "Hummer" equipped with position, vision and hazard sensors, control computers and actuation devices for steering and speed control (Figure 8). Four UGVs were built by Lockheed-Martin, and up to three have been operated simultaneously in formation. This section shows how formation behaviors were adapted for use on these robots.

Motor behaviors in the UGV architecture are coordinated by a speed arbiter and a turn arbiter. This approach differs from the motor schema method where each behavior generates both direction and magnitude. In the UGV architecture each arbiter runs

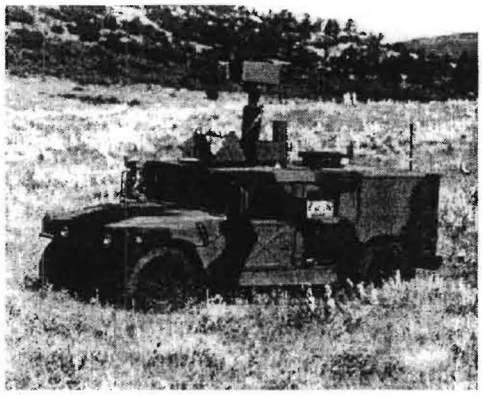


Figure 8: One of DARPA's UGVs for Demo II Program.

concurrently and accepts "votes" from the various active motor behaviors. For turning, behaviors vote for one of 30 discrete egocentric steering angles ; the angle with the most votes wins. A behavior may actually cast several votes for separate headings at once, where the votes are spread about a central angle with a Gaussian distribution. For speed, the lowest speed vote wins. Details on the mathematical formation of the arbitration process are available in [6].

As in the case of motor schema-based robots, the UGVs must simultaneously navigate to a goal position, avoid collisions with hazards and remain in formation. This is accomplished by concurrent activation of independent behaviors for each. Here we will deal only with the behaviors for formation.

For UGVs, formations and formation positions were determined in the same way as described for the **detect-formation-position** perceptual schema. But the non-holonomic constraints on UGV movement call for a revision of the formation motor behavior. Of significance in the non-holonomic case is that the robot's heading during and after formation corrections significantly impact its ability to remain in position. Not only should the vehicle be in the right location, but its heading should be aligned with the axis of the formation. If it is very far off heading, the robot will quickly fall out of position either laterally, fore-aft or both. A technique used by pilots for aircraft formation [7] is well suited for this task: positioning is decomposed into fore-aft and side-side adjustments. Fore-aft corrections are made by adjusting speed only, while lateral corrections are made by adjusting heading only. Each correction is applied independently. A consequence of the approach is that when a robot is ahead of its position it will not attempt to turn around, but just slow down. The following heuristics summarize the approach:

For speed selection:

- If the robot is in formation, the best speed for maintaining that formation is the current speed.

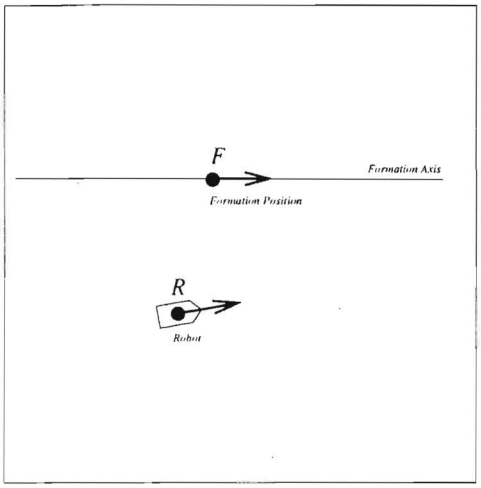


Figure 9: Illustration of terms used in describing formation behaviors for UGVs. In this diagram the robot is behind and to the right of its position in formation.

- If the vehicle is behind its position, it should speed up.
- If the vehicle is in front of its position, it should slow down.
- The selected change in speed should depend on how far out of position the robot is.
- Since the speed arbiter selects the lowest speed vote of all the active behaviors for output to the vehicle, formation control may only be possible by slowing down.

For steering:

- If the robot is in formation, the best heading for position maintenance is the formation axis.
- If the robot is out of position laterally and the formation is moving, it should turn towards the formation axis with an angle that depends on how far out of position it is.
- If the robot is out of position and the formation has stopped moving, the robot should head directly towards its position.

2.4.1 UGV Behaviors for Formation

Two separate behaviors, **maintain-formation-speed** and **maintain-formation-steer** run concurrently to keep the vehicle in position. Each determines an appropriate value at each movement step and votes accordingly. The votes, along with those from

other behaviors are tallied and acted upon by the speed and steering arbiters. The discussion will now focus on an individual robot and how the speed and steer behaviors determine their outputs. To formalize the approach, the following formation terms are introduced (see Figure 9):

- R_{pos}, R_{dir} the robot's present position and heading.
- R_{mag}, the robot's present speed.
- F_{pos}, the robot's proper position in formation.
- F_{dir}, the direction of the formation's movement.
- Formation Axis, a ray through F_{pos} in the F_{dir} direction.

The **maintain-formation-speed** behavior first determines the magnitude of the required speed correction, then casts its vote by adding the correction to the current speed. A gain value is used to adjust the rate of correction. The speed correction, *DeltaSpeed*, varies from -1.0 (slow down) to 1.0 (speed up). The magnitude depends on how far fore or aft the robot is of its desired position. Three zones, perpendicular to the formation axis and defined by distance fore or aft of F_{pos} determine *DeltaSpeed* (Figure 10). The size of these zones are parameters of the formation behavior. *DeltaSpeed* is set negative if the robot is in front of F_{pos} and positive otherwise. The magnitude is computed as follows:

- **Ballistic zone** : 1.0
- **Controlled zone** : the magnitude varies linearly from a maximum of 1.0 at the farthest edge of the zone to zero at the inner edge.
- **Dead zone** : in the dead zone the magnitude is always zero.

Finally, **maintain-formation-speed** casts its vote for the vehicle speed as follows:
$SpeedVote = R_{mag} + DeltaSpeed \times SpeedGain$

The **maintain-formation-steer** behavior follows a similar procedure to determine an egocentric steering direction, (the angle for the front wheels with respect to the vehicle body[1]). The calculation proceeds in three steps. First, the magnitude of correction is determined based on how far laterally the robot is from its formation position. The maximum correction is for the robot to head directly *towards* the formation axis, the minimum is for the robot to head directly along the formation axis. The magnitude of *HeadingCorrection* is determined as follows (Figure 10):

[1]In the actual implementation, votes are cast for a *turn radius*. For clarity we use the steering angle of the wheels here.

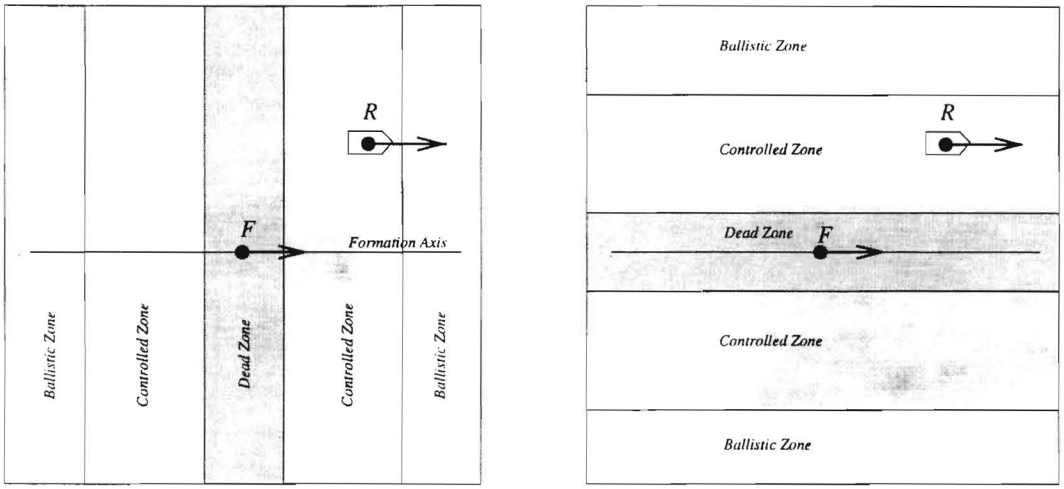


Figure 10: Zones centered on F, the desired formation position. The zones on the left are used for computing speed, corrections, while those on the right are for heading corrections.

- **Ballistic zone**: 90^o, i.e. head directly towards the axis.
- **Controlled zone**: the turn varies linearly from a maximum of 90^o at the farthest edge of the zone to 0^o at the inner edge.
- **Dead zone**: 0^o, i.e. head parallel to the axis.

The sign is set according whether the robot is left or right of the formation axis. If the robot is left of the axis, calling for a right turn the sign is positive, it is set negative otherwise. The *DesiredHeading* can now be determined with reference to the formation axis:

$$DesiredHeading = F_{dir} - DeltaHeading$$

As the robot moves forward, this heading will simultaneously bring it to and properly align it with the formation axis. In the special case where the formation has stopped moving, *DesiredHeading* is instead set to take the robot directly to its position:

$$DesiredHeading = F_{pos} - R_{pos}$$

Next, *DesiredHeading* is translated into an egocentric angle for the vehicle's front wheels:

$$SteerVote = DesiredHeading - R_{dir}$$

Positive angles indicate a right turn and negative ones a left turn. If the result is either greater than 180° or less than −180°, 360° is added or subtracted to bring the result within those bounds. Finally the angle is clipped to the physical limits of the vehicle.

2.5 The Formation Expert

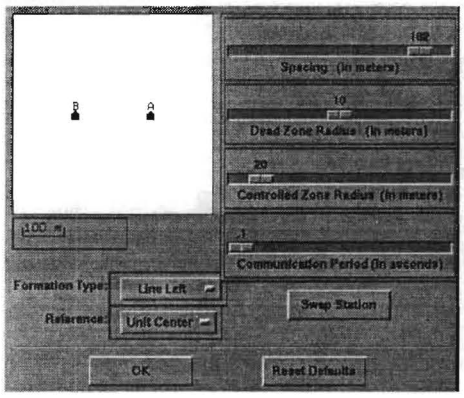


Figure 11: The Formation Expert GUI.

To aid UGV mission planners in selecting appropriate formations for particular situations, we developed an expert system called the Formation Expert. The Formation Expert automatically analyzes a mission plan then suggests parameter settings for formation behaviors based on that context. A user may adjust these recommended parameters with a graphical user interface (GUI). The displayed diagram of the formation is adjusted to reflect changes the user makes as he moves slider bars or pushes buttons (Figure 11).

In order to make its recommendations, the Formation Expert consults a *rule base*, which specifies conditions under which particular formations are appropriate. The rule base is an easy to understand text file that can be revised by a user to reflect new situations or better formations for certain situations. Also, since the Formation Expert is generic it may be easily adapted for use in other UGV domains as well.

2.6 Results for UGV Demo II Mobile Robots

The unit-center referenced approach was used exclusively on the HUMMERs because the UGV architecture only provides for a robot to slow down to keep formation. It was felt that since the leader would never slow down to keep formation and a trailer could never speed up if it fell behind, a leader-referenced approach would fail. Final integration with mobile robots was completed by Lockheed-Martin in Denver.

Figure 12: Two DARPA UGVs in formation (from left to right: line, wedge, column)

Formation played a key role in the success of UGV Demo C in the Summer of 1995. At that demonstration two HUMMERs ran through a series of tests including a sequence of formations (Figure 12). The HUMMERs followed an approximately one-half mile course across open terrain while smoothly shifting from column to wedge to line formations.

The behaviors were extended by Lockheed-Martin for use in three robot HUMMER formations. The three robot formations have run satisfactorily but videotape of the tests is not yet available. Performance in these tests was limited by a communications system that induced up to 7 seconds of latency in robot to robot position reports. This problem points to the utility of using a passive approach for locating team members, versus the explicit exchange of location based on GPS readings.

3. Team Teleautonomy

This research concerned the development and implementation of methods to allow a human operator to control a team of robots. Our approach provides a mechanism to significantly reduce the human operator's cognitive and perceptual load by allowing the reactive system to deal with each robot's local control concerns. Two principal mechanisms to achieve this are by allowing the operator to act either as a constituent behavior of the society or to allow him/her to supervise the societal behavioral sets and gains, acting only as needed based upon observable progress towards task completion.

3.1 Summary of Results for FY 94 and FY 95

Two forms of teleautonomous control of teams of mobile robots were developed and implemented. In each of these forms, the operator is allowed to control whole societies of agents; not one robot at a time, but rather controlling global behavior for the entire multiagent system. The end product is a simple way for a commander to control large

numbers of constituent elements without concern for low-level details (which each of the agents is capable of handling by themselves).

The first method for telerobotic control allows the human operator to give directional information to the robot team. He controls the output of a **teleautonomy** behavior by using an on-screen "joystick". The **teleautonomy** behavior then produces a vector output in the direction that the joystick is depressed and with a magnitude relative to the amount that the joystick is depressed. This vector is sent to each of the robots and is then summed and normalized with the vector outputs from the other active behaviors on each robot. The robot then executes the resultant vector. In this method of teleautonomous control, the operator acts as one of the robots' behaviors.

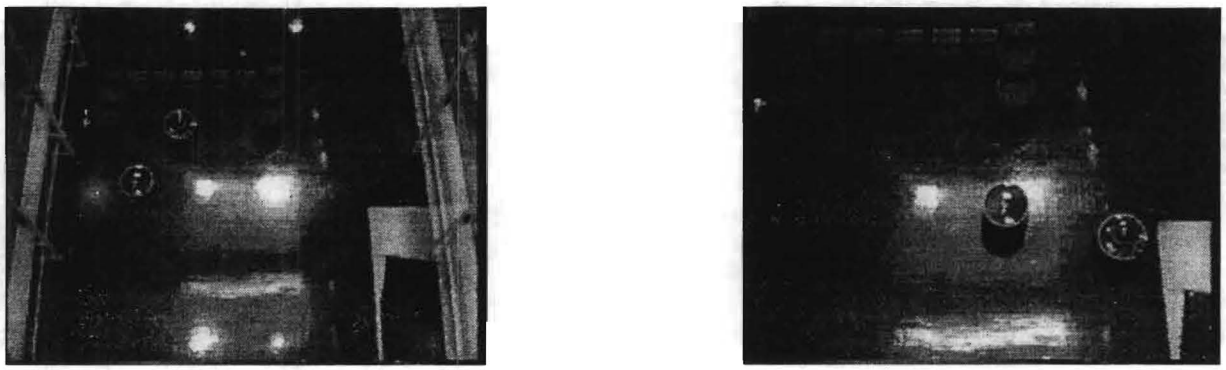
In the second method, the operator interactively changes the overall behavior of the robot team by adjusting the parameters of the reactive behaviors. The human operator can manipulate the behavioral parameters either individually or in terms of abstract groupings such as personality traits. Making parameter changes in terms of personality traits allows a user, with no knowledge about the underlying behaviors and their parameters, to effectively modify the robots' behavior. In our current system, the abstract parameters include *Aggressiveness* and *Wanderlust*. The value of an abstract parameter controls the values of several individual low-level parameters. The operator uses slider bars to modify the value of an abstract personality trait. In this method of telerobotic control, the operator acts as a behavioral supervisor.

Both methods for telerobotic control have been integrated with the *MissionLab* system and work both in simulation and on real robots. Additionally, both teleautonomy methods have been integrated with the DARPA UGV Demo II architecture using the STXmcu mission control system for use on teams of HMMWVs. The first method was demonstrated at a technical demo during Demo C of the UGV project in the summer of 1995.

Experiments was conducted to test the usefulness of the **teleautonomy** behavior for certain tasks. Some of the experiments were conducted in simulation and some were conducted on real robots.

The simulation experiments tested the first method of control, the operator as a behavior approach. The tasks included foraging, grazing, and herding. For the foraging task, if teleoperation is used wisely, it can significantly lower the total number of steps required to complete the task by greatly reducing the time spent in the *forage* state (*i.e.*, the number of steps that the robots spend looking for attractors). For the grazing task, teleoperation was not significantly better than no teleoperation. For the herding task, the **teleautonomy** behavior was discovered to be an acceptable tool, but possible improvements were determined.

The experiments on real robots tested the use of both the **teleautonomy** behavior and the abstract parameters. The tasks included directing the robots out of a box canyon and squeezing the robots through a small space. The telerobotic interface was tested on a pair of Denning MRV-2 mobile robots. A Sun Sparcstation 5 served as the



(a) (b)

Figure 13: Box canyon task: (a) The robots are stuck in the box canyon. (b) The robots are being maneuvered out of the box canyon using the teleautonomy behavior.
The camera was located 4 floors above the robots, giving a birds-eye view of the action. The robots have circles of white tape on top of them to make them more visible.

base station, running the telerobotic interface through *MissionLab*.

For the first task, the human operator was able to use the on-screen joystick to steer the robots out of a box canyon. After the robots were no longer in danger of falling back into the box canyon, the operator released the joystick, and the robots continued to their destination autonomously. Figure 13 shows video stills of the robots during this experiment.

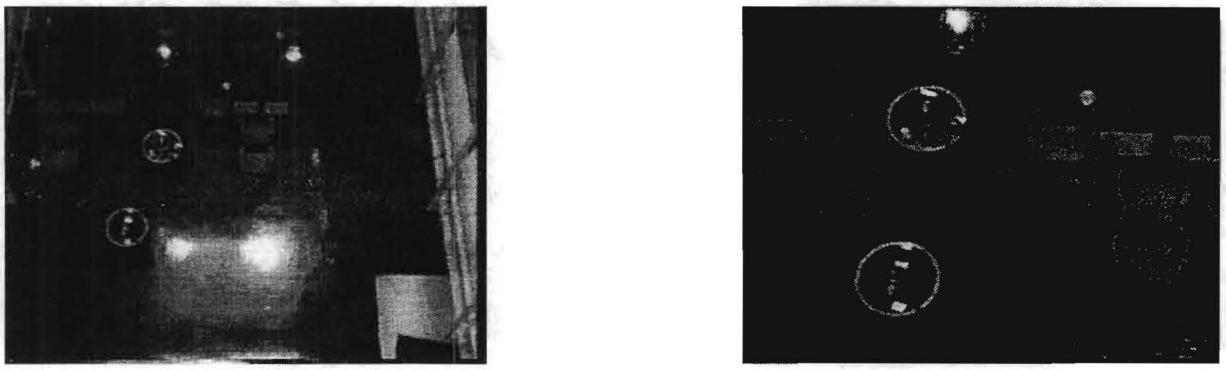
For the second task, the operator used the behavioral parameter control to cause the robots to squeeze through a small gap that would normally be too small for the robots to traverse, due to the repulsion from their **avoid-static-obstacle** behavior. By increasing the abstract parameter *Aggressiveness*, the operator was able to squeeze the robots through the small space. Figure 14 shows video stills of the robots during this experiment.

A set of usability tests were conducted on the operator's interface. These tests yielded useful information for making the interface more helpful and usable. Changes were made to the interface based on suggestions derived from these usability tests.

3.2 Research Accomplishments for FY 96

A Hummer has been actuated and controlled by means of teleautonomy. The low-level control software for the Hummer has been integrated with the *MissionLab* toolset. This allows a human operator to control the movement of the Hummer through the **teleautonomy** behavior using the on-screen "joystick" described in the Team Teleautonomy section. The joystick is shown in Figure 15.

The operator is located outside of the Hummer and commands a compass direction



(a) (b)

Figure 14: Squeezing task: (a) The robots are stuck in the box canyon with a gap. (b) The robots are being squeezed through the gap in the box canyon by making them more aggressive.
The camera was located 4 floors above the robots, giving a birds-eye view of the action.

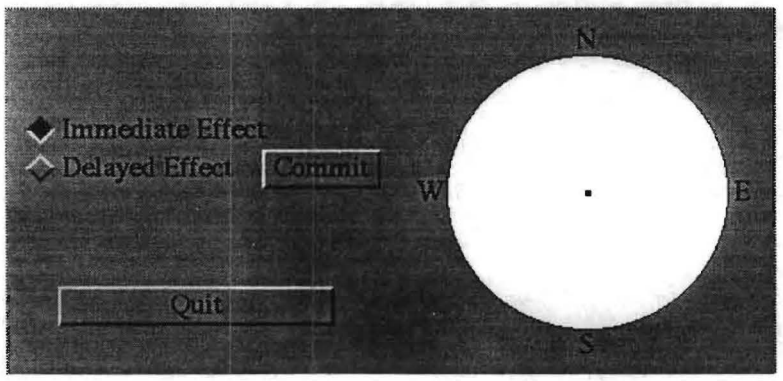


Figure 15: The on-screen "joystick" allows the human operator to control the robots heading in terms of compass directions.

and a speed in miles per hour by manipulating the joystick. This information is transmitted by means of a wireless communication network to the robot control software running on-board the Hummer. The on-board processing software then repeatedly reads the current heading and speed of the vehicle and determines how to control the steering wheel, brake, and throttle to cause the Hummer to move at the desired velocity. This allows a human operator to exert a form of supervisory control over the robot, where he sets a desired velocity and then does not need to send any more commands as the robot autonomously determines how to achieve and maintain this velocity.

Two experiments have been conducted to test the teleautonomous control of the Hummer. The first experiment tested the capability to provide the operator with control over the steering of the robot. The second tested steering and speed control by a human operator.

At the time of the first experiment, only steering control had been implemented. The operator sat inside the Hummer in the back seat and controlled the heading of the vehicle in terms of compass directions through the joystick. An emergency driver sat in the driver's seat and controlled the throttle and brake. The emergency driver did not touch the steering wheel, but was prepared to take control of the steering if necessary. The vehicle was steered by computer control around a crowded parking lot. Part of the test included steering the vehicle through a tight space with only a few feet of clearance on both sides of the robot. The experiment showed that our steering control worked, and that the method of controlling the robot's heading by specifying compass directions is effective. However, we also realized that this method of specifying compass directions would probably work better if done from outside the vehicle.

In the second experiment, the human operator was remotely located outside of the robot vehicle. This time, the operator controlled both direction and speed. An emergency driver was located in the driver's seat of the Hummer, but he did not touch the steering wheel, brake, or throttle. The vehicle was driven by computer control around a large empty parking lot. The operator successfully caused the robot to execute many turns and circles, driving both uphill and downhill. Figure 16 shows the vehicle during the experiment. The experiment successfully demonstrated steering, brake, and throttle control of both the direction and speed of the robot, and that controlling the heading of the robot by specifying a compass direction is easier to accomplish when the human operator is located outside of the robot vehicle.

4. MissionLab

MissionLab is powerful set of software tools for developing and testing behaviors for single robots and robot teams. Code generated by *MissionLab* can directly control commercial robots built by Denning Mobile Robotics and Nomadic Technologies, as well as an experimental 4-wheel drive Hummer developed at Georgia Tech. A primary strength of *MissionLab* is its support of both simulated and real robots. A developer

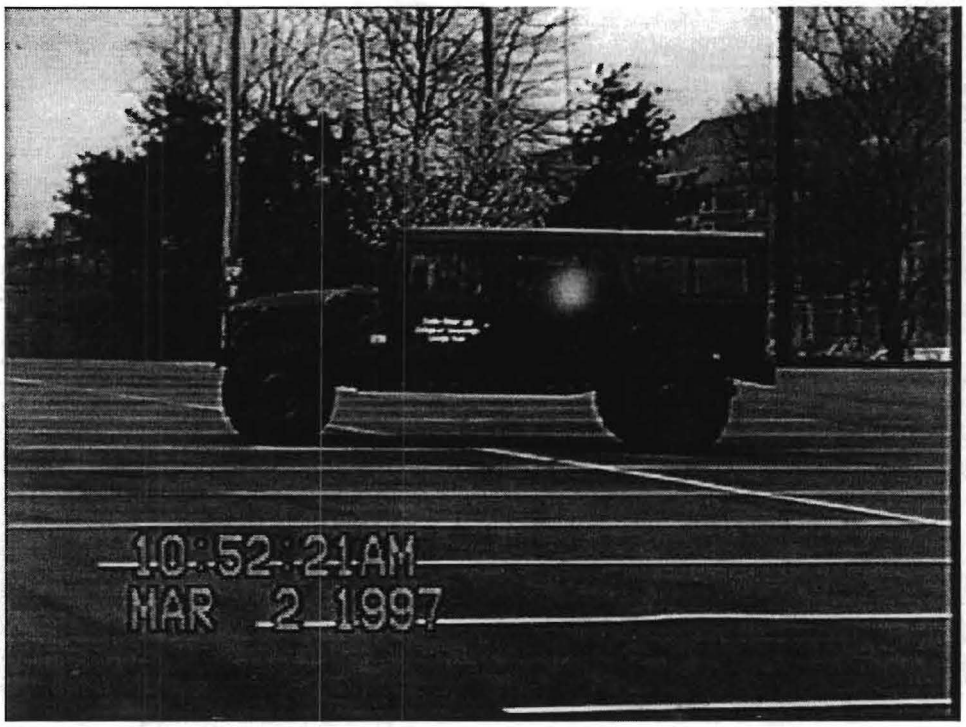


Figure 16: The automated Hummer under teleautonomous control. The operator was controlling the direction and speed of travel from another vehicle at the edge of the parking lot.

can experiment with behaviors in simulation and then run those same configurations on mobile robots.

MissionLab has a distributed architecture. The main user's console can run on one computer while multiple robot control executables are distributed across a network, potentially onboard the actual robots they control.

The core of the *MissionLab* toolset is composed of five primary components:

- **cfgedit**: The Configuration Editor is a graphical tool for building robot behaviors. The designer can build complex control structures with the point and click of a mouse. cfgedit generates source code which, when compiled, can directly control a simulated or real robot. Details on cfgedit are provided in a separate manual.

- **cnl**: cfgedit generates a control program in the Configuration Network Language (CNL) which is compiled by the program cnl. The compiled program (or robot executable) may now directly control a robot. The cnl compiler is automatically invoked by cfgedit when needed. In general, users will not need to be concerned with cnl unless they want to develop their own primitive behaviors.

- **mlab**: Once the robot executables are created, they can be tested in mlab. mlab is primarily a console-like program from which a developer monitors the progress of experimental runs of his robot executables. When mlab is used for simulation (as opposed to controlling mobile robots), it serves as a sensor and actuator simulator from the point of view of the robot executable. On mobile robots, the actual sensors are used instead.

- **CMDL**: The Command Description Language (CMDL) may optionally be used for describing simple sequential robot missions. CMDL files are read by mlab at runtime and offer a mechanism for providing high-level input to robot behaviors developed in cnl.

- **ODL**: The Overlay Description Language provides descriptions of the environment (especially useful in simulation) to mlab. Obstacles, boundaries and so on may be described in ODL.

The *Societal Agent* architecture was developed to capture the recursive composition of configurations. Specifying a reactive behavioral configuration for use by a multiagent team executing a mission requires both a careful choice of the behavior set and the creation of a temporal chain of behaviors which executes the mission. This difficult task is simplified by applying an object-oriented approach to the design of the mission using a methodology called *temporal sequencing*. Temporal sequencing partitions the mission into discrete operating states with perceptual triggers causing transitions between those states. Several smaller independent configurations (assemblages) can

then be created which each implement one state. Assemblages consist of groups of basic behaviors and coordination mechanisms that allow the group to be treated as a single, coherent behavior. Upon instantiation, the assemblage is parameterized based upon the requirements of these specific mission requirements. These assemblages can be re-parameterized and used in other states within this mission or archived as high level primitives for use in subsequent projects.

The *MissionLab* system, an implementation of the *Societal Agent* architecture, supports graphical construction of configurations using a visual configuration editor. This editor, *CfgEdit*, supports the recursive construction of reusable components at all levels, from primitive motor behaviors to societies of cooperating robots by allowing creation of coordinated assemblages of components which are then treated as atomic higher-level components available for later reuse. The Configuration Editor allows deferring commitment (binding) to a particular robot architecture or specific vehicles until the configuration has been developed. This explicit binding step simplifies developing a configuration which may be deployed on one of several vehicles which may each require use of a specific architecture. The process of retargeting a configuration to a different vehicle when the available vehicles or the system requirements change is similarly eased. The capability exists to generate either MRPL code for the DARPA UGV architecture or C++ code targeted for the Autonomous Robot Architecture (AuRA) which is executable within the *MissionLab* system. The AuRA executables drive both simulated robots and several types of Denning vehicles (DRV-1, MRV-2, MRV-3). The architecture binding process determines which compiler will be used to generate the final executable code, as well as which libraries of behavior primitives will be available for placement within the editor.

The mission scenario language and corresponding interpreter permit the specification of complex multiagent missions in a structured, relatively user-friendly, language. The mission coordination operator has the expressive power of a finite state machine but allows the user to specify the sequence of steps making up the mission using a domain-specific language with high-level primitives and mnemonic names. At run time, the mission coordination operator communicates with the operator console to allow the mission to be entered interactively or predefined missions to be executed from saved files.

Using these tools, various multiagent missions have been demonstrated in simulation and on our Denning robots. The *MissionLab* toolset was demonstrated as part of UGV demo C and has been made publicly available in source code form via anonymous FTP or WWW access.

4.1 Research Accomplishments for FY 94 and FY 9595

1995 saw the completion and initial release of the *MissionLab* toolset. The *MissionLab* robot software development system provides support for users in the various

stages of mission development (*e.g.*, behavior implementation, assemblage construction, and mission specification). The primitive behavior implementor must be familiar with the particular robot architecture in use and a suitable programming language such as C++. The assemblage constructor uses a library of behaviors to build skill assemblages using the graphical configuration editor. This allows visual placement and connection of behaviors without requiring programming knowledge. However, the construction of *useful* assemblages still requires knowledge of behavior-based robot control. Specifying a configuration for the robot team consists of selecting which of the available skills are useful for the targeted environments and missions. Specification of the mission sequence can occur at run-time using a domain-specific structured language. Military terminology and nomenclature are used in *MissionLab* to facilitate specification of missions by military users unfamiliar with robot control techniques. The overall philosophy, however, is by no means restricted to this application domain.

Consider specification of a configuration implementing a janitorial task for a mobile robot. Specifically, the robot should wander around looking for empty soda cans, pick them up, wander around looking for a recycling basket, and then place the can into the basket. Figure 17 is a schematic representation of an FSA for such a robotic trash collector constructed using *CfgEdit*. The circles represent the possible operating states with the label indicating the assemblage agent active during that state. The arcs are labeled with the perceptual triggers causing the transition where relevant. Powering up in the *start* state, the robot begins to wander looking for a suitable soda can, operating in the *Look_for_can* state. When a can is perceived, the *Pick_up_can* state is activated and if the can is successfully acquired, a transition to the *Look_for_basket* state occurs. Loss of the can in either of these states causes the FSA to fall back to the previous state and attempt recovery. When a recycling basket is located, the *Put_can* state becomes active and the can is placed in the basket. A transition back to the *Look_for_can* state repeats the process.

Figure 18 shows *MissionLab* executing a simulation. The large area with various things drawn in it is the main display area. Within the display area robots, obstacles, and other features are visible. The solid round black circles are obstacles. The four robots are moving across the middle of the display area in roughly a diamond formation. More details about the type of mission displayed in the figure are explained in the next section. The command interface in the lower right part of Figure 18 allows the operator to control the execution of the mission. The steps of the mission are displayed as they execute.

Specifically, several software projects were finished: The configuration editor *CfgEdit* was rewritten to support the recursive construction implicit in the *Societal Agent* architecture. It serves as the focal point for the *MissionLab* integrated development environment and spawns other parts of the toolset as the user requires them. It allows graphical construction and visualization of the configuration description language (CDL) descriptions which can then be compiled into the selected output format. A conversion to the DARPA UGV standard IPT communications package was completed.

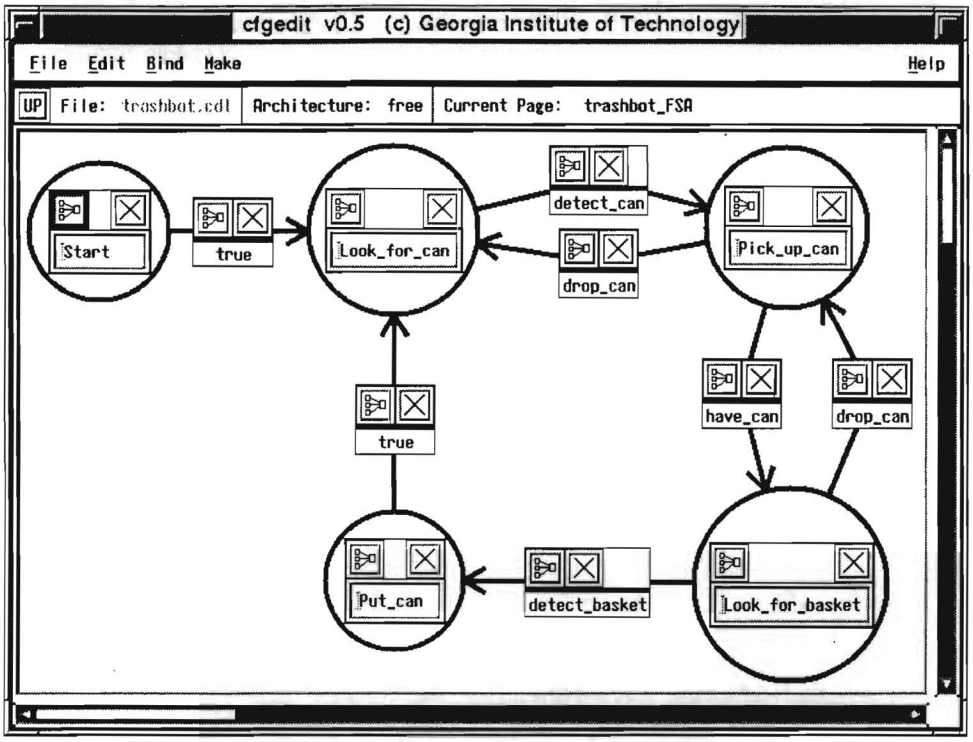


Figure 17: FSA for a trash collecting robot

The robot executables communicate to the simulation server and operator console using IPT. Three internal compilers have been written. Two CDL compilers generate either the MRPL specifications used in the UGV program or a CNL (configuration network language) description of the input configuration. The CNL compiler can then translate this second type of output into C++ code using the AuRA architecture. These AuRA-style executables can then be run within the *MissionLab* operator workstation, freely mixing simulated and real robots.

For DARPA UGV Demo C, Georgia Tech demonstrated the *MissionLab* mission specification and configuration software as part of a Technology Demo. This demo involved collaboration of Georgia Tech and the University of Texas at Arlington (UTA) in a joint demonstration. The *MissionLab* system has been adopted and extended by UTA to help verify their sensing algorithms for this joint tech demo. The system is available through the Internet (http://www.cc.gatech.edu/ai/robot-lab/research/MissionLab).

4.2 Research Accomplishments for FY 96

Significant accomplishments regarding MissionLab in 1996 include usability studies highlighting its utility in robot behavior design, improvements in the distribution and installation of the software, the addition of three dimensional views, hardware drivers for the control of new robot types, and porting to the Linux operating system. A

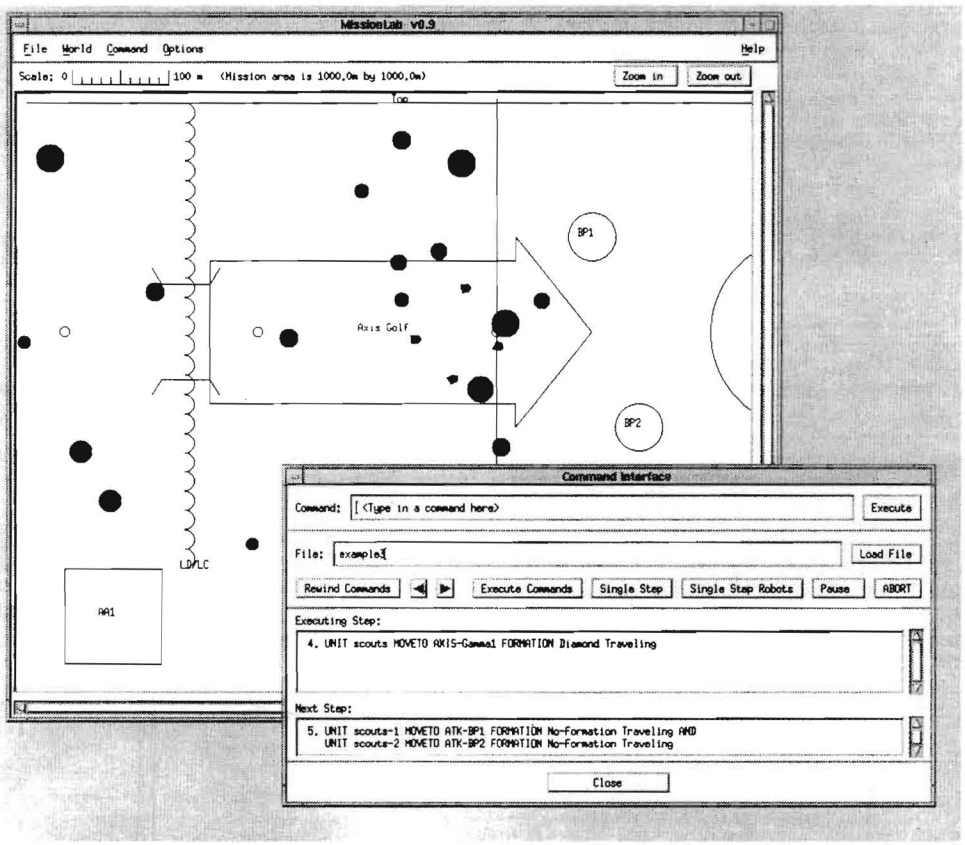


Figure 18: Example scenario in *MissionLab*

usability study completed this year as part of a Ph.D Dissertation has demonstrated the advantages of MissionLab as a robot design tool [4].

Previously, MissionLab was only available for computers running the SunOS operating system. This year, the entire MissionLab toolset has been ported to the Linux operating system as well. This significantly expands its applicability since portable computers running Linux may cost as little as $\frac{1}{4}$ the cost of other Unix-based computers. It is economically feasible to outfit robots with individual onboard control using low-cost Linux machines. This, in concert with the distributed capabilities of MissionLab make it easy to conduct multi-robot experiments over a distributed wireless network (Figure 19). In addition to compiling on Linux machines, the overall ease of installation on SunOS and Linux is now greatly improved.

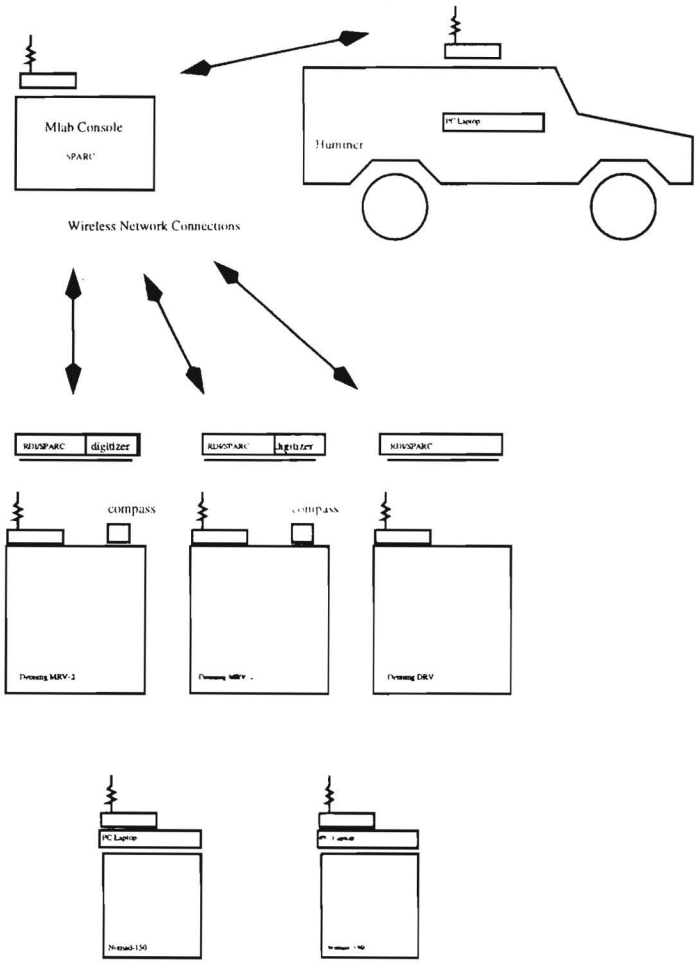


Figure 19: MissionLab provides for distributed control of multirobot missions. A single console computer (upper left) can be used to command a team of robots over a wireless network. The remote Unix machines are installed on the robots they control.

Three dimensional views can be invoked in mlab to give the user a more realistic

representation of the mission scenario. 3D views of the layout include top view, side view and front view. The robot in the views is shown as a six-legged robot. A snapshot of the 3D views is shown in Figure 20. The 3D views are generated using `sphigs`

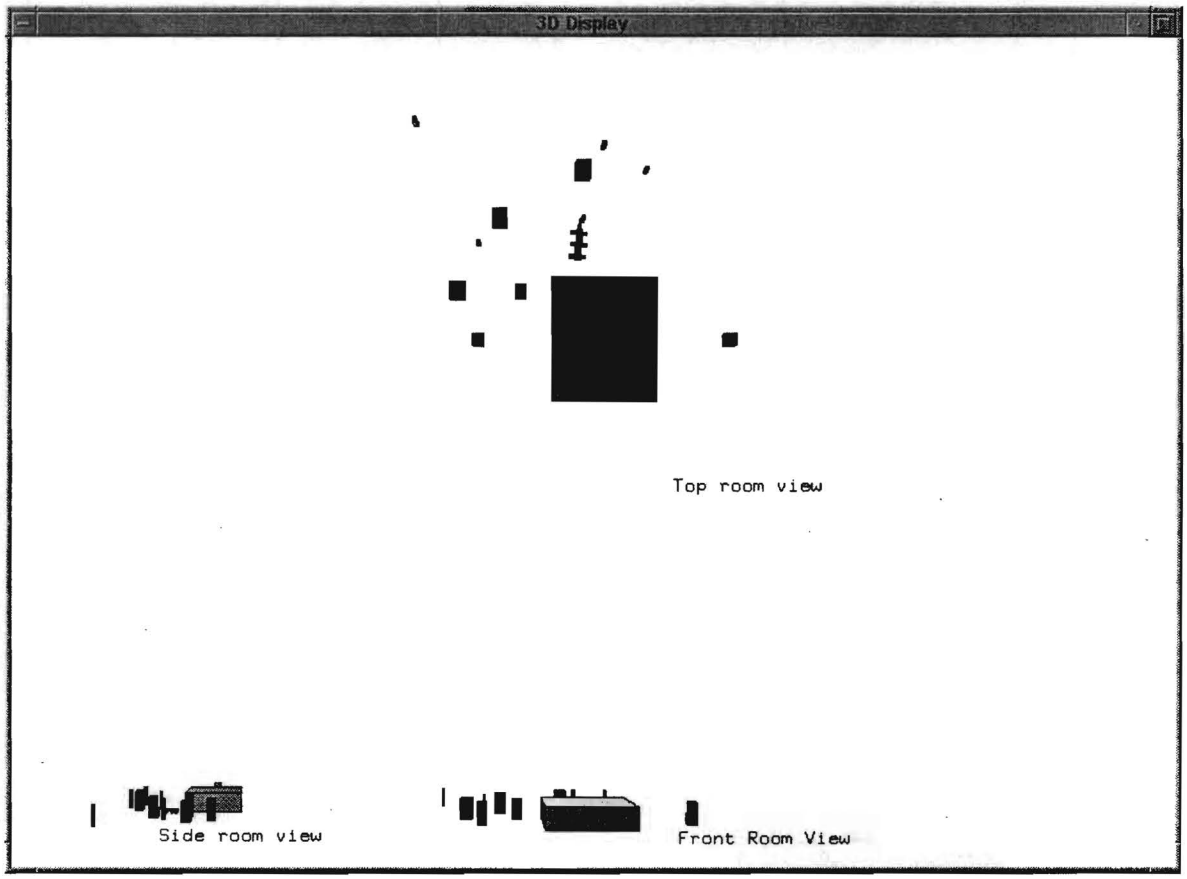


Figure 20: An example of the 3D view now available in MissionLab.

package.

In this version, all objects have the same height, and they are located on the same z plane. Basically, the 3D views contain the same information as the 2D view except for some 3D enhancement. In the future, we will change the overlay definition to accommodate true 3D descriptions, including the height of the object and the location of the object in the z direction. Robot movement vectors will also be extended to 3D to enable the robot to navigate in a 3D world. Also, robot representation descriptions will be added to the overlay file to allow the user to define the shape of the robot. Currently a fixed 6 legged robot is used to represent all robots. Some viewpoint control mechanism will also be added to view the 3D world from different angles.

Hardware drivers were added to MissionLab this year to enable support for control of Nomadic Technologies Nomad 150 robots and a robotic Hummer developed at

Georgia Tech. This is in addition to support for Denning MRV-3, MRV-4 and DRV robots already in place.

5. Experimental Testbeds

5.1 Nomad Testbed

MissionLab has been extended to provide control of Nomadics Technologies Nomad 150 robots (Figure 2). A new software library provides an interface between standard MissionLab movement commands and control messages sent over a serial line to the hardware. The library is similar to the MissionLab libraries which provide control of Denning robots and Georgia Tech's Hummer.

Each Nomad is equipped with an on-board PC laptop running Linux (a Unix-compatible operating system). Behaviors for the robots are compiled into a robot executable developed on a workstation using cnl or cfgedit. After development and testing, the executable is downloaded to the robots with ftp over a wireless network. The network supports typical services like ftp and telnet, but more importantly it enables the MissionLab console program to manage a team of robots remotely from a single workstation (Figure 19).

The Nomad 150s are being equipped with color vision capabilities. Real-time vision-processing is provided by an on-board computer (Cognachrome from Newtonlabs) able to track multiple objects identified by color at 30 updates per second. The vision computers will enable robots to track one another and salient features in their environment. This provides for tasks including formation based on visual references and foraging for colored objects. MissionLab control programs communicate with the vision computer over a serial line using an additional interface library.

5.2 Hummer Testbed

Various equipment was installed on a Hummer to provide autonomous control. Safety for personnel and equipment were the highest priority in the design of the Hummer control system. Care was taken throughout the installation processes to ensure that we could operate the vehicle safely at all times. A high level C++ software library was writen to control the actuators. Two programs were also wriiten: one to test the actuators and the other to operate the vehicle via a serial port. This section describes the equipment installed in the Hummer, the software library used to control the installed equipment, and some challenges that arose during the vehicle automating process.

5.2.1 Hardware Description

POWER

The following voltages are needed for the operation of the installed components; +12VDC, -12VDC, 5VDC, 28VDC and 110VAC. Electrical power to operate the installed equipment is obtained from the vehicle's battery. An inverter/charger provides 110VAC to operate a monitor and other devices that need AC power. A low power LCD display can replace the monitor. The needed DC voltages are produced using DC/DC converters. The converters produce reliable outputs for input voltages between 11-14VDC. The inverter/charger, when plugged into a 110VAC outlet, supplies 110VAC and 13.6VDC to the vehicle. The 14.5VDC can be used to charge the vehicle's battery.

ACTUATORS

Three actuators control the Hummer, one on the brake, one on the throttle, and one on the steering wheel. Each actuator is unique and provided its own set of challenges during installation. At the core of each actuator is a DC motor that turns in different directions when given a different polarity voltage, and at different speeds for different voltages. There is a clutch on each motor so an operator can quickly decouple the actuated mechanism from the actuator.

Brake Actuator: The brake actuator was purchased as a complete unit from Red Zone Robotics. This unit includes the positioning motor, motor position encoder, brake position potentiometer, reduction gears, and a clutch. The brake actuator is mounted on the floor of the vehicle directly below the brake pedal. When the clutch is engaged, the encoder gives a direct indication of the brake pedal position. The potentiometer output is not currently being used. The potentiometer output could be used to detect failures such as the operator has stepped on the brake or the brake clutch has failed.

Steering Actuator: The steering actuator was assembled using off-the-shelf components. A motor, reduction gears, position encoder and a clutch are used for the driving mechanism. A bracket was fabricated that allowed the driving mechanism to be mounted directly under the steering wheel shaft and as close to it as possible. A sprocket was installed on the factory steering wheel and the drive mechanism connected to the steering wheel with a nylon belt. When the clutch is engaged, the encoder gives a direct indication of the position of the front wheels. As a safety measure, we have designed the belt to slip if the operator grabs the steering wheel. This slippage may be a problem if the vehicle is operated in rough terrain and excessive force is needed to turn the wheels. Slippage will result in a mismatch in the actual and measured position of the front wheels. If slippage occurs, the vehicle will steer slightly off the desired heading.

Throttle Actuator: The factory installed cruise control servo is used to operate the throttle. The use of this servo made installation easy but implementation very difficult. This servo does not have a positioning encoder installed on it. This caused

many problems which are covered in the Electronics and Software sections below. The throttle actuator, unlike the other two actuators, does not need to have a constant voltage applied to it to maintain a desired throttle position. Once the throttle position has been set, when voltage is removed from the servo motor, the throttle will remain in that position.

ELECTRONICS

At the heart of the actuator control is a PC104 80486-based computer board. This board has support for a VGA monitor, four serial lines, a parallel line, an Ethernet connection, and an LCD monitor. A hard drive is connected to the computer and contains the development environment but it is not necessary for the operation of the actuator software. The computer may be started with a boot disk and the programs are small enough to fit on one floppy disk.

PC104 compliant embedded controllers manage the actuators. These devices handle much of the low-level control work necessary for the positioning devices. The user supplies the values for the proportional, integration, integration limit, differentiation, velocity and acceleration parameters. Once the parameters are set, a user can give a desired position to the board and the board performs the required trajectory.

The embedded controllers work very well for the brake and the steering wheel. These actuators have encoders on them that show the position of their respective motor. The encoders enable the embedded controller to operate independently. Due to the lack of encoder position for the throttle, its control presents some difficulties tht are covered further in the software section. The motor and clutch for the throttle actuator are connected the same as the other actuators. Since there is not an encoder on this actuator, the encoder inputs on the embedded controller are not connected to anything.

A PC104 compliant board with A/D converters and counters is installed at the top of the PC104 stack. This board allows for the measurement of speed. Distance pulses are generated by a magnetic pickup coil located on the transmission output shaft. This is the factory installed sensor used in various places throughout the vehicle to compute speed. The raw output signal is very noisy and could not be used as a direct connection to the counter. The factory cruise control module circuitry is used to filter the sensor output and produce a square wave proportional to the distance traveled.

5.2.2 Software Description

We wrote the software is Borland C++ to take advantage of data abstraction. The software lends itself easily to modifications and added functionality. The Controller is the major class. The Brake, Steer and Throttle class inherit from the Controller class and add methods as necessary for their unique functions. There are also a Counter class and a Speed Class. Each of these classes is covered below.

Controller Class

The Controller class provides the interface to the embedded controllers. This class provides methods for setup, initialization, and positioning devices connected to the embedded controllers.

Steer and Brake Class

The Steer and Brake class inherit from the Controller class. The Brake class provides methods for increasing and decreasing the brake position and for releasing the brake. The Steer class provides methods for moving the steering wheel. The steering is controlled by a method that moves the steering wheel to position the front wheels to the desired position given the desired angle, movement velocity and acceleration. The input angle is the desired angle of the front wheels and can be between -32 and 32 degrees. Negative angles correspond to right turns. To ensure excessive force is not applied to the steering wheel, the Steer class will not attempt to turn the wheel unless the vehicle's velocity is greater than 1MPH. In order for a Steer object to know the current ground speed, it is connected to a Speed object.

Throttle Class

The Throttle class inherits from the Controller class. Due to the throttle actuator's lack of an encoder, this class operates a bit differently from the Steer and Brake classes. The vehicle's Throttle Position Indicator (TPI) signal is run to the A/D converter. Currently the software does not use this input for throttle position but it can be incorporated into the throttle control if more functionality is desired. A voltage may be applied to the throttle servo by sending a command to the embedded controller that will attempt to move the servo at a steady velocity or to a desired position. Since the controller does not have feedback as to the position of the throttle, the voltage will continue to be applied until another command is given to stop the motor. The stop command must be given a precise time after the move command to position the throttle to a desired position.

We first explored two methods to apply the stop command a given time after the move command is given. One method was to put in a delay for how much time to apply the voltage. During the delay time the software was not addressing other vehicle functions, such as the speed of the vehicle. The other method was to put a command into the loop that will continue to check if the desired time has expired and then apply the stop command. This method presented a major safety concern. If something in the software failed between sending the move command and before the stop command is given, the throttle could quickly be fully applied.

The goal was to be able to send a command to the embedded controller that applied the voltage for the desired amount of time. To accomplish this goal we used the positioning error abilities of the embedded controller. A user may define a maximum position error in the controller that, when exceeded, will remove power from the controlling servo motor. The throttle does not need to have a voltage applied to it to maintain its position. With the right combination of position error, acceleration, proportionality gain and desired position values, a voltage may be applied to the brake

for a given amount of time with just one command. After the command is sent, further monitoring will not needed. The use of the throttle to control the speed of the vehicle is covered in the Speed Class below.

Counter Class

The Counter class provides methods to interface to the installed A/D/counter board. Methods may be easily added to this class to control and access all items connected to the A/D board. Currently, the major function of this board is to read the counts from the ground distance sensor. There is another counter on this board that may be used to measure engine speed or other devices that provide a pulse. The A/D inputs are configured to measure voltages between 0 and 10 volts. This board can be used to measure any voltage between the configured voltage and therefore provides many capabilities to add future functionality to the entire configuration such as a temperature sensor in the Motor Controller Box.

Speed Class

The Speed class provides methods to control and read the current ground speed of the vehicle. The method for controlling the speed is designed to be placed in a loop and called at least four times per second. In order for a Speed object to control the vehicle's speed, it must be connected to a Brake, Throttle, and Counter object. The Speed class will read the speed via the counter object. If the vehicle speed is less than the desired speed, the Speed class will ensure the brake is released and apply the throttle. If the vehicle's speed is greater than the desired speed, the Speed class will decrease the amount of throttle applied. If the vehicle does not begin to slow, the Speed class will slowly apply the brake until the desired speed is obtained. If stop command or a desired velocity of zero is sent to the Speed class, it will apply the brake to bring the vehicle to a safe and steady stop. If the stop command is used, it must also be in a loop and continually sent to the Speed object.

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] Army. *Field Manual No 7-7J.* Department of the Army, Washington, D.C., 1986.

[3] Balch, T. and Arkin, R.C., "Motor Schema-based Formation Control for Multiagent Robot Teams", *1995 International Conference on Multiagent Systems*, San Francisco, CA, pp. 10-16, 1995.

[4] MacKenzie, D., "A Design Methodology for the Specification of Behavior-based Robotic Systems", *Ph.D. Dissertation*, College of Computing, Georgia Institute of Technology, Atlanta, GA, 1997.

[5] MacKenzie, D., Cameron, J., Arkin, R., "Specification and Execution of Multiagent Missions", *Proc. 1995 Int. Conf. on Intelligent Robotics and Systems IROS '95*, Pittsburg, PA, Vol. 3, pp. 51-58, 1995.

[6] Rosenblatt, J., "DAMN: A Distributed Architecture for Mobile Navigation", *Working Notes AAAI 1995 Spring Symposium on Lessons Learned for Implemented Software Architectures for Physical Agents*, Palo Alto, CA, March 1995.

[7] U.S. Air Force. *Air Combat Command Manual 3-3.* k Department of the Air Force, Washington, D.C., 1992.

6. Publications Resulting from this Research

- FORMATION BEHAVIORS

1. Balch, T. and Arkin, R.C., 1995. "Motor Schema-based Formation Control for Multiagent Robot Teams", *1995 International Conference on Multiagent Systems*, San Francisco, CA, pp. 10-16.
2. Arkin, R.C. and Balch, T., 1997. "Cooperative Multiagent Robotic Systems", invited chapter to appear in *AI-based Mobile Robots: Case Studies of Successful Robot Systems*, ed., D. Kortenkamp, R.P. Bonasso, and R. Murphy, MIT Press.

- TELEAUTONOMOUS CONTROL

1. Arkin, R.C. and Ali, K., 1994. "Integration of reactive and telerobotic control in multi-agent robotic systems", *Proc. Third International Conference on Simulation of Adaptive Behavior, (SAB94) [From Animals to Animats]*, Brighton, England, Aug. 1994. pp. 473-478.
2. Ali, Khaled S., 1994 "Telop: Teleoperation of multi-agent reactive robotic systems", *Working paper, contact author.*
3. Ali, K.S. and Arkin, R.C., 1997. "Multiagent Teleautonomous Behavioral Control", to appear *Robotica*. 1997.

- MISSION SPECIFICATION

1. MacKenzie, D. and Arkin, R.C., 1993. "Formal specification for behavior-based mobile robots", *Mobile Robots VIII*, Boston, MA, Nov. 1993, pp. 94-104.
2. Cameron, J. and MacKenzie, D., 1994. "Specifying complex military scenarios", *Working paper, contact authors.*
3. MacKenzie, D. and Arkin, R.C., 1995. "Specification and Execution of Multiagent Missions", *Proc. 1995 conference on Intelligent Robots and Systems (IROS'95)*, August 1995. Pittsburgh, PA, Vol. 3, pp. 51-58.
4. MacKenzie, D., Arkin, R. and Cameron, J., 1997. "Specification and Execution of Multiagent Missions", *Autonomous Robots*, Vol. 4, No. 1, Jan. 1997.
5. MacKenzie, D., 1997. "A Design Methodology for the Configuration of Behavior-Based Mobile Robots", *Ph.D. Thesis*, College of Computing, Georgai Tech, 1997.
6. MacKenzie, D. and Arkin, R.C., 1997. "Evaluating the Usability and Utility of Robot Programming Toolsets", submitted to *International Journal of Robotics Research.*

- INTER-ROBOT COMMUNICATION

1. Balch, T. and Arkin, R.C., 1994. "Communication in reactive multiagent robotic systems", *Autonomous Robots*, Vol. 1, No. 1.
2. Arkin, R.C. and Balch, T., 1997., "Communication and Coordination in Reactive Robotic Teams", to appear in *Coordination Theory and Collaboration Technology*, ed. G. Olson, J.B. Smith, and T. Malone, 1997.
3. Arkin, R.C. and Balch, T., 1995, "AuRA: Principles and Practice", submitted to *Journal of Experimental and Theoretical Artificial Intelligence*. 1995.