Evaluating the Usability of Robot Programming Toolsets*

Douglas C. MacKenzie
Mobile Intelligence Corp., and Georgia Tech
doug@mobile-intelligence.com

Ronald C. Arkin
College of Computing
Georgia Tech
arkin@cc.gatech.edu

October 14, 1997

Abstract

The days of specifying missions for mobile robots using traditional programming languages such as C++ and LISP are coming to an end. The need to support operators lacking programming skills coupled with the increasing diversity of robot run-time operating systems is moving the field towards high-level robot programming toolsets which allow graphical mission specification. This paper explores the issues of evaluating such toolsets as to their usability. This article first examines how usability criteria are established and performance target values chosen. The methods by which suitable experiments are created to gather data relevant to the usability criteria are then presented. Finally, methods to analyze the data gathered to establish values for the usability criteria are discussed. The *MissionLab* toolset is used as a concrete example throughout the article to ground the discussions, but the methods and techniques are generalizable to many such systems.

1 Introduction

The field of mobile robotics has matured to the point where it is time to move robots out of laboratories and into the hands of users. However, before this transition can occur better methods for tasking robots are required. Currently, highly skilled robotic experts hand-craft robot missions using traditional programming languages, such as C++ and LISP. This unnecessarily excludes people who are not fluent computer programmers from functioning as robot end-users.

Robot programming toolsets intend to improve this situation by providing an integrated development environment for specifying, evaluating, and deploying robot missions. Such a toolset should allow novice users to specify robot missions using a visual programming paradigm. Here, a visual editor allows users to graphically create missions (complex sets of tasks) by selecting reusable high-level constructs and related perceptual activities from menus. Integrated support for evaluating solutions via simulation and finally deploying them on robots must also be available.

*This research was funded under ONR/ARPA Grant # N0001494-1-0215. The Mobile Robot Laboratory is supported by additional grants from the U.S. Army and NSF. The experimental procedures and benchmark tasks were developed in cooperation with Erica Sadun, Darrin Bentivegna conducted the *MissionLab* experiments, and many others have contributed to the *MissionLab* system.

The specification of the components, connections, and structure of the control system for a group of robots will be called the **configuration**. A configuration consists of a collection of agents, inter-agent communication links, and a data-flow graph describing the structure of the configuration created from the agents and channels. There are two types of agents: atomic and assemblages. The atomic agents are parameterized instances of primitive behaviors while assemblages are coordinated societies of agents which function as a new cohesive agent. The configuration references the low level primitives, but does not describe their implementations, since that is generally hardware dependent. A configuration generally is a solution for a particular robot mission.

This article explores the issues surrounding just how one begins to evaluate the usability of robot programming toolsets used to create and maintain configurations. It is necessary to consider usability early in the develop cycle[4, 10, 14, 15, 26, 33], but when the application is available, it must be evaluated as to its usability by the target audience. There are four popular procedures to evaluate the usability of software packages[23] in the Human-Computer Interfaces literature. *Heuristic evaluation*[25, 32] asks interface specialists to study the package and look for aspects that, based on their experience, will be confusing for users. A process called *Guidelines*[31] has developers rate their system based on a list of good interface design principles. In *Cognitive walkthroughs*, developers perform software walkthroughs to evaluate the actions required by the toolset based on a cognitive model of how users will expect the interface to work. *Usability testing*[14, 23] attempts to study and measure how representative users interact with the system while performing realistic tasks. The peculiarities of applying Usability testing to a robot programming toolset are the focus of this article.

The desired characteristics of a Robot Programming Toolset are presented in Section 2. The *MissionLab* system, an exemplar toolset used to ground these discussions, is presented in Section 3. Section 4 presents specific techniques which can be used to establish usability criteria for toolsets, with Section 5 documenting the usability criteria established for *MissionLab*. Designing experiments to generate values for usability criteria is discussed in Section 6 while two specific experiments created to evaluate *MissionLab* are presented in Section 7. The evaluation of experimental data is discussed in Section 8 with the results for the *MissionLab* experiments analyzed in Section 9. The summary in Section 10 concludes the article.

2 Robot Programming Toolset Requirements

Behavior-based robotic systems are becoming both more prevalent and more competent[5, 22, 8, 27, 13, 9, 16, 1]. However, operators lacking programming skills are generally forced to use canned configurations hand-crafted by experienced roboticists. This inability of ordinary people to specify tasks for robots inhibits the acceptance of robots into everyday life. Even expert roboticists are often unable to share solutions since there is no common language for configuration descriptions. Indeed, a configuration commonly requires significant rework before it can be deployed on a different robot, even one with similar capabilities.

A robot programming toolset should attack these issues head-on. Current methods for specifying mobile robot missions using traditional programming languages such as C++ or LISP must be replaced with visual programming interfaces to support novice users. The configurations created must remain architecture- and robot-independent until explicitly bound to the target robots, easing the transformation from one system implementation to another. This independence coupled with

support for multiple code generators ensures that a wide variety of robots can be supported from a single high-level toolset. Finally, integrated simulation and run-time support are required to ease the process of evaluation and deployment.

Toolsets should clearly separate development tasks. Skilled developers are then able to create libraries of high-level control abstractions tailored to a particular target task domain. Robot commanders can then select, parameterize, and combine components from these libraries to perform a variety of missions, without requiring detailed robotics knowledge.

MissionLab[6], presented in the next section, is an example of a robot programming toolset which meets these goals (Another example targeted to industrial robotics is Onika[29, 11]). *MissionLab* uses the *assemblage*[18] abstraction to permit the recursive construction of new coherent behaviors from coordinated groups of other behaviors. This allows developers to build libraries of increasingly higher-level abstractions which are directly tailored to their end-users' needs. *MissionLab*'s support for the graphical construction of state-transition diagrams allows the use of *temporal sequencing*[3] that partitions a mission into a set of discrete operating states, with assemblages implementing each state.

3 Example: The *MissionLab* Robot Programming Toolset

The *MissionLab* toolset has been created at Georgia Tech as an integrated development environment for behavior-based mobile robots. It provides operators with a graphical configuration editor which allows developing and visualizing multi-agent robot missions. An integrated simulator allows preliminary evaluation of mission configurations before they are deployed. *MissionLab* also permits mixing simulated and real robots within a single mission to allow evaluating the benefits of additional hardware.

This section provides an overview of the *MissionLab* toolset to ground the usability evaluations which will follow. In-depth descriptions of *MissionLab* can be found in the *MissionLab* user's manual[6].

3.1 The Societal Agent theory

The theoretical basis of *MissionLab* is the **Societal Agent** theory[20, 21], which describes the recursive composition of agents in both natural and man-made organizations. Minsky proposes an agent-based structure of human intelligence in "The Society of Mind"[24]. The **Societal Agent** theory broadens this agent metaphor by proposing that coordinated societies of physical agents can be viewed as coherent agents in their own right.

This provides insight into the recursive composition of societal agents. Consider a herd of buffalo moving across a plain. Each individual animal has a variety of motor behaviors active, such as herding, obstacle avoidance, and eating. Each of these motor behaviors can be represented as an agent. Each buffalo is an agent constructed from its own individual motor behaviors. Within the herd, a cow and her calves group together and form a `cow_with_calf` agent. The herd itself is an aggregate of all the societal subgroups of which it is constituted. As a whole, the herd has both speed and direction and constitutes the top-level recursively constructed agent.

Examples are also common in human circles, with military organizations being the most prominent. For example, squads of soldiers train, live, and fight together with the intent to form a cohesive

squad agent which is interchangeable with other similarly performing *squad* agents. There are certain well documented commands and actions which each squad must be capable of carrying out, independent of their particular individual subparts. This allows the lieutenant commanding the platoon to plan at the squad level, and ignore the details and idiosyncrasies of individual soldiers. Similarly, the company commander will abstract the platoons into coherent objects, since platoons also constitute coherent agents.

3.2 The Configuration Description Language

The Configuration Description Language (CDL) captures the **Societal Agent** theory in a recursive composition language tailored for representing behavior-based robot configurations. CDL represents only the mission *configuration*, not the robot- and architecture-dependent implementations of the behavioral primitives. CDL encourages creation of generic mission descriptions by partitioning hardware specific information from the bulk of the configuration, supported by an explicit binding step.

CDL specifies how primitives are instantiated and coordinated, not how they are implemented. This is necessary to allow configurations to remain independent of implementation details. Each primitive must have a CDL prototype which specifies how it is invoked. An important facet of CDL is its support for the construction of reusable assemblages. This allows building libraries of high-level primitives for later reuse. Assemblages are defined using the `defAgent` keyword and can be used interchangeably with primitives. The syntax and semantics of CDL is formally defined in [20, 21] and the interested reader should look there for in-depth treatments.

3.3 The *MissionLab* Toolset

The *MissionLab* toolset has been developed based upon Configuration Description Language. Figure 1 shows a block diagram of the *MissionLab* system. The user interface centers around the graphical designer (Configuration Editor - *CfgEdit*). From here the user can develop configurations, bind them to specific robots, and generate executables. The CDL compiler generates architecture-specific code based according to the user's intentions. Built-in support for the AuRA [2] architecture allows deploying and monitoring configurations on the multiagent simulator and/or robots, all from within *MissionLab*.

CfgEdit is used to create and maintain configurations. It supports the recursive construction of reusable components at all levels: from primitive motor behaviors to entire societies of cooperating robots. *CfgEdit* supports this recursive design process by facilitating the creation of coordinated assemblages of components which are then treated as higher-level components available for later reuse. It allows deferring commitment (binding) to a particular robot architecture or specific vehicles until the abstract mission has been developed. This explicit binding step simplifies development of a configuration which may be deployed on different robotic vehicles with each perhaps requiring use of a different behavioral architecture. The process of retargeting a configuration when hardware requirements change is thus eased.

MissionLab currently possesses the ability to generate code for either the ARPA Unmanned Ground Vehicle (UGV) architecture[13, 12, 27, 28] or for the AuRA architecture[1, 19, 2]. The AuRA executables drive both simulated robots, several types of Denning robots (DRV-1, MRV-2, MRV-3), and a robotic HUMMER all-terrain vehicle. The binding process determines which compiler

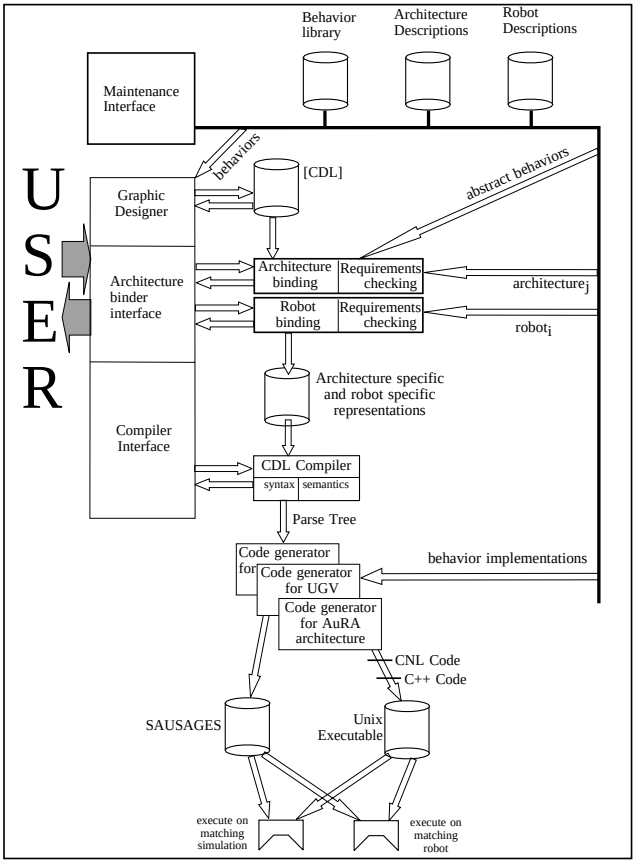

Figure 1: Block diagram of the *MissionLab* System.

is used to generate the executable code and which libraries of behavior primitives are available for user placement within the graphical editor. The *MissionLab* system[30] is available in both source and binary form at `http://www.cc.gatech.edu/ai/robot-lab/research/MissionLab`.

3.4 Illustrative *MissionLab* session

Figure 2 shows a screen snapshot of CfgEdit with a military mission to survey a minefield loaded. This display is where end-users will normally interact with the system. Missions are constructed by adding states and transitions to the workspace. Once added, these objects are further specified by choosing appropriate behavioral and perceptual actions to carry out their task. The selection process uses popup menus showing available library components, each with a short description. Figure 3 shows a screen snapshot of the selection popup used to choose a new behavior for a mission state.

Once an appropriate behavior is selected, it must be parameterized for the specific mission. Figure 4 shows a screen snapshot of the parameter modification popup for the `AwayFrom` perceptual

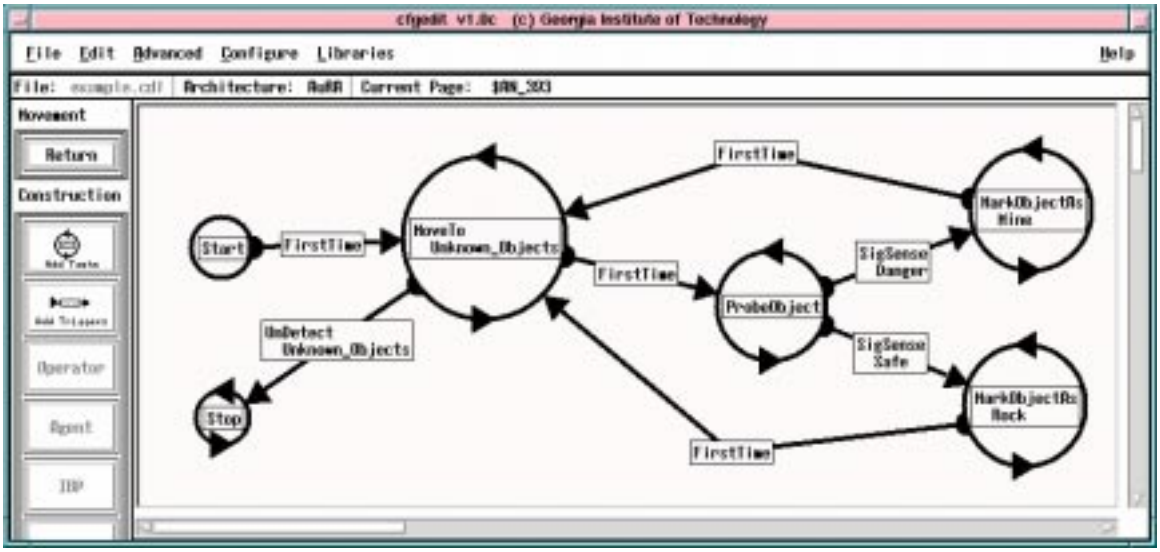

Figure 2: CfgEdit displaying an example Mission (encoded as a finite state diagram) to survey a minefield.

Figure 3: Example list of behaviors available for states.

trigger, which specifies when a behavioral transition should occur at a certain distance from a detected object. Notice the use of radio-buttons for selecting which classes of objects the perceptual process is sensitive to and the slider-bar for setting the trigger distance.

The high-level abstractions available to the mission commander are assemblages constructed previously for other missions (using the editor) that were subsequently archived to the component library. Figure 5 shows the components of the `MoveTo` behavioral assemblage. It is built as a cooperatively coordinated assemblage of three other assemblages: `Noise`, `AvoidObstacles`, and

Figure 4: Example list of parameters for `AwayFrom` perceptual trigger.

`MoveToGoal`. The recursive construction of components at all levels allows designers to focus on building high-level components, tailored to their specific target domain, yet draw upon previous solutions.

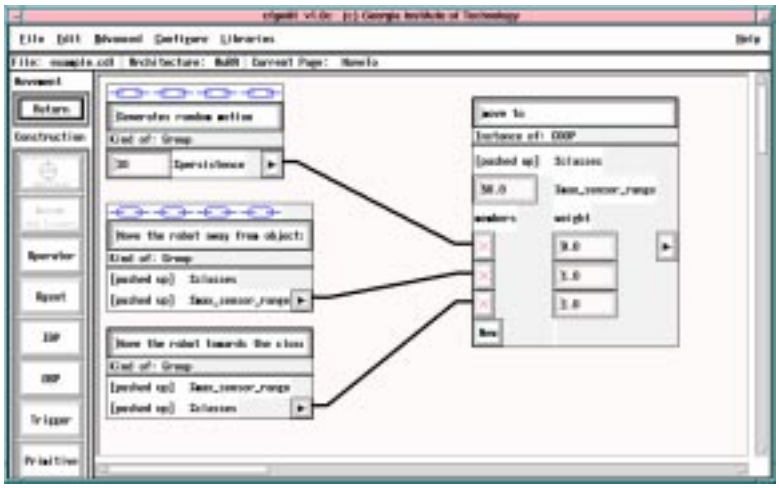

Figure 5: Example `MoveTo` assemblage.

4 Establishing Usability Criteria

Usability criteria are necessary and useful during two stages of a product's life cycle. First, they should serve as design goals to focus development efforts. Secondly, when the end product is available, usability experiments can measure values for these metrics, providing statistical data for determining the success and degree of completion of the development effort. Table 1 depicts an example technique for presenting the usability metrics.

Table 1: An example usability criteria specification table for some indeterminate task (After [14], page 223). Notice that `Usability Attributes` are vague high-level concepts while the `Values to be Measured` are concrete performance metrics. The `Current Level` shows the average user performance on existing systems. The `Worst Acceptable Level`, the `Target Level`, and the `Best Possible Level` are predictions of the average performance of users on the new system.

Example Usability Specification Table					
Usability Attribute	Value to be Measured	Current Level	Worst Acceptable Level	Target Level	Best Possible Level
Novice performance	Time to perform action A	Hours	30 minutes	20 minutes	5 minutes
Novice performance	Time to perform action B	20 minutes	5 minutes	1 minutes	15 seconds

Notice that each line in the table lists a unique tuple combining an attribute and measurable value $\langle$`usability attribute`, `value to be measured`$\rangle$ and specifies target values for that feature. Using a table such as this, the designer can focus his/her efforts on improving performance in areas that are important, instead of wasting time on improving insignificant aspects. This table also provides criteria to objectively determine when the development process is finished. Once a product achieves all of the minimum acceptable values, it can be considered satisfactory. We now define each of the columns appearing in Table 1.

- **Usability Attributes**

  The `Usability Attributes` are high level concepts that are deemed important to the customers, such as the performance of new users. The careful selection of attributes is necessary to ensure that all important facets of the human-computer interface are covered. For example, though a lot of attention is normally placed on improving the performance for users familiar with the system, all users begin as novices. Therefore, if the system is too painful for new users to learn, there will be no expert users to consider.

- **Value to be Measured**

  The `Value to be Measured` selects a particular aspect of the attribute for which we will specify performance figures. A particular attribute may have several relevant values which can be

used to measure aspects of it. For example, given an attribute such as "novice user performance" there are many values which can be measured to illuminate aspects of the attribute. A small subset includes "time to perform a benchmark task," "time to perform a particular action," and "number of errors while performing a benchmark task." The idea is to take a high-level concept like "novice user performance" and develop concrete metrics that can be experimentally verified and which provide insight into the attribute itself. Of course, a particular value may be relevant to multiple usability attributes.

- **Current Level**

  The `Current Level` represents the average performance achieved by the target class of participants using the current state of the art. In cases where users of the existing systems are unable to perform the proposed task, a value of `not possible` can be entered. It is important to list these values to set the threshold the new product must compete with. There is little hope for acceptance if a new product is worse than what the customers are currently using.

- **Worst Acceptable Level**

  The worst acceptable level sets the minimums for the design process. Any values which are, on average, below this threshold require further refinement before the product can be considered finished. These values are normally close to the current levels since customers won't switch to something clearly worse than what they currently have. These are the best estimates of the levels below which the customers will not use the product.

- **Best Possible Level**

  The upper bound on the level of performance that could reasonably be expected is called the `Best Possible Level`. This knowledge is useful to aid understanding of the significance of the performance values. The value should be set to the highest level that could reasonably be expected, on average, from users of the system. A useful method to determine these maximums is to base them on the performance of members of the development team using the system. It is unlikely that a user will ever be as familiar with the system as its designers and, therefore, their performance is likely to be less.

- **Target Level**

  The target levels define what the designers should be striving towards. These goals can be set based on market surveys, predicted customer needs, and other relevant information. Normally this value would be set last, after the `Best` and `Current` values are available to provide guidance. It is important that the designer has some input into these levels to ensure they are realistic and achievable with the available level of personnel and technology. It does little good to set targets that are out of reach.

5 Example: *MissionLab* Usability Criteria

In this section, using *MissionLab* as a concrete example, usability criteria are established that serve as a prelude to an actual usability evaluation of the toolset. (An example of analyzing Onika appears in [11]). Two primary objectives were identified as important for the evaluation of *MissionLab*:

1. Show that it is significantly faster to create robot configurations by using the *MissionLab* toolset than by writing corresponding C code.

2. Show that the *MissionLab* toolset is well suited to the configuration design task.

Given these objectives, the following usability criteria were developed, using the procedures described in the previous section, to rate the usability of the *MissionLab* configuration editor for specifying robot missions:

1. `Time to add a mission step`
   The time required to add a new step to a mission is an important determiner in how long it takes to construct missions from task descriptions.

2. `Time to specialize a step`
   The time required to change the behavior of a step in a mission sequence is a predictor of the time required to modify existing configurations.

3. `Time to parameterize a step`
   The time required to change the parameters used by a mission step also impacts usability of the toolset.

4. `Time to add a mission transition`
   The time required to create a new transition between two operating states in a mission sequence gives an indication of how easily the user is able to manipulate the configurations.

5. `Time to specialize a transition`
   The time required to change the perceptual events causing a particular transition in a mission sequence is a predictor of the time required to modify existing configurations.

6. `Time to parameterize a transition`
   The time required to change the parameters used by a perceptual activity also impacts the usability of the toolset.

7. `Number of compiles required to create a simple configuration`
   The number of edit/compilation cycles required to create benchmark configurations measures the level of understanding of the users.

8. `Time to create a simple configuration`
   The time required to create benchmark configurations serves as a yardstick metric, giving a handle on the overall performance of the test participants using the toolset.

9. `Ability to create a configuration`
   A binary metric which catalogs the ability of participants to successfully create configurations using the toolset.

Table 2 lists the usability criteria using the tabular form developed earlier. The `Current Level` values are *a priori* estimates based on participants using a traditional programming language. These predictions can be re-evaluated using the data gathered from experiment 2 (presented in Section 7.2).

Table 2: The *MissionLab* usability criteria specification table.

***MissionLab* Usability Specification Table**						
	Usability Attribute	Value to be Measured	Current Level	Worst Acceptable Level	Target Level	Best Possible Level
1.	Novice user performance	Time to add a mission step	1 Min	30 sec	10 sec	1 sec
2.	Novice user performance	Time to specialize a step	2 min	1 min	30 sec	3 sec
3.	Novice user performance	Time to parameterize a step	1 min	1 min	30 sec	2 sec
4.	Novice user performance	Time to add a mission transition	1 min	30 sec	10 sec	2 sec
5.	Novice user performance	Time to specialize a transition	2 min	1 min	30 sec	3 sec
6.	Novice user performance	Time to parameterize a transition	1 min	1 min	30 sec	2 sec
7.	Novice user performance	Number of compiles to create a configuration	4	5	2	1
8.	Novice user performance	Time to create a simple configuration	20 min	20 min	15 min	5 min
9.	Non-programmer performance	Ability to create configurations	No	Yes	Yes	Yes

The `Worst Acceptable Levels` were picked arbitrarily by the designer as estimates of the performance levels below which experienced programmers will avoid using the system. These levels are intended to be slightly lower than the performance of programmers using the C language. The system will be acceptable if experienced programmers suffer only a mild drop in productivity, since the system will also empower non-programmers, as reflected in Attribute 9. For this class of novice roboticists we are looking for a clear improvement, from not being able to specify missions, to the successful construction of robot configurations. The `Best Possible Levels` were determined based on the performance of the developer. These values are likely unapproachable by all but very experienced users. The `Target Levels` reflect the design goals of the project. These numbers were selected as targets for the development effort to provide a clear benefit to users over traditional programming languages.

6 Designing Usability Experiments

Once metrics have been specified and the various values selected, it is necessary to determine how data can be gathered to allow measuring the levels for the metrics. This is not an easy task and

requires careful planning and execution to prevent bias and noise from swamping the underlying data.

Objective methods for data gathering generally involve test subjects using the system under controlled conditions[17]. Commonly, the software is instrumented to gather keystroke and timing information that will allow determining how the user performed certain tasks. The experiments are best if administered by a third party observer to remove bias and to keep the developers from interjecting knowledge not commonly available. This observer is responsible for logging interesting events in a journal of the experiment. The sessions are also videotaped to provide a method for closer and repeated examination of interesting details (and as a permanent record in case of disagreements with participants). Although these sterile test environments clearly impact participant performance, they do allow objective comparisons between competing techniques.

It is important to note that before conducting experiments such as these involving human subjects, it is necessary to gain approval at most institutions from an oversight organization. At Georgia Tech this is the *Human Subjects Board*. These experiments were approved for this project, by that board, contingent on participants reading and signing the informed consent form reproduced in Figure 6.

Gathering the data using objective methods is clearly preferable, but not always possible. Certain attributes (*i.e.*, initial impression, user comfort, *etc.*) are by nature subjective and best gathered via questionnaires and informal discussions. Of course, the questions must be carefully crafted to minimize sampling bias. The Questionnaire for User Interface Satisfaction (QUIS)[7] has been developed at the University of Maryland as a general purpose user interface evaluation tool and has undergone extensive testing and validation. The QUIS test can provide a starting point to creating a customized test to extract the desired information.

7 *MissionLab* Usability Experiments

We now present two usability experiments which were developed to allow establishing values for the usability attributes in Table 2. In Experiment 1 the participants construct a series of configurations to achieve written mission specifications using the graphical configuration editor. Experiment 2 repeats the process for the subset of subjects in Experiment 1 comfortable using a traditional programming language. Since participants conduct Experiment 2 using conventional text editors, it is necessary to exercise care in the experimental procedures to ensure that as many of the usability attributes as possible are being measured accurately. Participants were asked *a priori* if they were fluent in the `C` programming language. Of those answering yes, half were randomly assigned to complete Experiment 1 first and the remainder completed Experiment 2 first. This was intended to allow measuring the learning effect which aided the second experiment performed.

The remainder of this section presents the development of the experiments; the procedures to be followed in carrying them out, the nature and type of data generated, and the evaluation methods followed in analyzing the data.

INFORMED CONSENT FORM

THIS STUDY IS ABOUT THE MISSIONLAB SYSTEM. YOU HAVE BEEN ASKED TO PARTICIPATE IN OUR EVALUATION OF THE MISSIONLAB ROBOT COMMAND AND CONTROL ENVIRONMENT. THIS EVALUATION IS BEING CONDUCTED BY DOUGLAS MACKENZIE (DOUG@CC.GATECH.EDU) AS PART OF HIS DISSERTATION RESEARCH UNDER DR. RON ARKIN (ARKIN@CC.GATECH.EDU). ANY QUESTIONS ABOUT THE EVALUATION MAY BE ADDRESSED TO MR. MACKENZIE OR DR. ARKIN. THIS EVALUATION CONSISTS OF SEVERAL SESSIONS--A TUTORIAL SESSION AND TESTING SESSIONS--WHICH MAY BE CONDUCTED ON SEPARATE DAYS. THESE EXPERIMENTS WILL BE COMPLETED OVER THE NEXT THREE TO SIX MONTHS.

YOU WILL BE ASKED TO PERFORM VARIOUS TASKS WITH THIS SYSTEM. THESE TASKS INVOLVE THE EXPLORATION OF MISSIONLAB DESIGN AND TESTING ENVIRONMENT. WE ARE EVALUATING THE SYSTEM TO MAKE IT AS USABLE AS POSSIBLE BUT WE ARE IN NO WAY EVALUATING YOU. THE PURPOSE OF THIS STUDY IS TO TEST MISSIONLAB, NOT TO TEST YOU. THERE IS NO WAY FOR YOU TO FAIL. WE WILL UNOBTRUSIVELY GATHER STATISTICS RELATED TO TASK COMPLETION TIMES TO EVALUATE THE MERITS OF THE ROBOT TASKING TOOLS. SESSIONS WILL LAST APPROXIMATELY THREE HOURS AND INVOLVE MINIMAL RISKS.

YOU WILL BE ASKED TO COMPLETE QUESTIONNAIRES AT THE START AND END OF EACH SESSION. SHORT INTERVIEWS WILL ALSO BE CONDUCTED AFTER YOU FILL OUT THE QUESTIONNAIRES TO BETTER UNDERSTAND YOUR ANSWERS. DURING THE SESSION YOU WILL BE ASKED TO "SPEAK YOUR THOUGHTS ALOUD", SO YOUR IMPRESSIONS OF THE SYSTEM CAN BE BETTER UNDERSTOOD.

YOU ARE A VOLUNTEER. YOU WILL NOT BE PAID FOR THESE SESSIONS ALTHOUGH WE WILL REIMBURSE ANY SPECIAL EXPENSES DIRECTLY RELATED TO THE TESTING WITH PREAPPROVAL BY DR. ARKIN. WE WILL BE USING A SUBJECT POOL OF APPROXIMATELY TWENTY FIVE TO FIFTY PEOPLE.

As a participant you have certain rights which are listed below.

1. You may withdraw from the session at any time for any reason. Taking part in this study is completely voluntary. If you do not take part, you will have no penalty. If you have any questions about your rights as a research volunteer, call or write:

   Debbie Bell, IRB Administrator, 404-894-6906
   Office of the Provost, Georgia Tech
   Atlanta, GA 30332-0325

2. At the conclusion of the session you may see your data. If you decide to withdraw your data, please inform the evaluator immediately. No attempt will be made to maintain confidentiality.

3. You are requested not to discuss this session with other people who might be in the group from which other participants are drawn during the next few months.

4. Reports of injury or reaction should be made to Dr. Ron Arkin (PI) at 404-894-8209. Neither Georgia Tech nor the principle investigator has made provision for payment of costs associated with any injury resulting from participation in this study. If you have any questions about this research, call or write Dr. Arkin (room 375 MaRC, Georgia Tech). Campus mailing address is College of Computing, mail code 0280.

Thank you. Your time and effort while participating in this study are greatly appreciated. Your signature below indicates that you have read this form in its entirety and that you voluntarily agree to participate.

I have read and understood the information above. The researchers have answered all my questions to my satisfication. They gave me a copy of this form. I consent to take part in this study.

Subject's Signature:_____ Date:_____
Subject's Name:_____
Subject's E-mail:_____

Investigator's signature:_____ Date:_____

Figure 6: This consent form was approved by the Georgia Tech oversight board for use in the usability experiments (Patterned after [14], page 300).

7.1 Experiment 1: CfgEdit Mission Specification

7.1.1 Objective

Determine the performance of novice and expert users specifying benchmark robot missions using the Configuration Editor.

There are two target audiences for *MissionLab*: Non-programmers who are able to use the toolset, and expert programmers who can successfully utilize both *MissionLab* and traditional programming languages. Test participants are drawn from both participant pools for this experiment. This allows testing both the hypothesis that skilled programmers can utilize the *MissionLab* system with little drop in productivity after minimal training, and that there exists a group of people who can create a *MissionLab* configuration but are unable to construct the corresponding code directly.

To evaluate this research project's military relevance, an attempt was made to include a number of U.S. Army Reserve Office Training Corps (ROTC) students as test participants. This allows testing the claim that many will be able to modify a *MissionLab* configuration but unable to manipulate corresponding configurations written in traditional programming languages. If a significant number of the ROTC participants are able to use the *MissionLab* toolset, it will explicitly show the impact of this research for the military community.

7.1.2 Experimental Setup

An independent third party observer conducts and monitors the experiments to ensure impartiality.

Test Environment

1. A small quiet room where participants can be observed unobtrusively.
2. Videotape equipment to record the session.
3. An X Window-based workstation (SUN SPARC 10).

Desired Test participants

The broadest spectrum of people possible should be run through this experiment. How these test participants are chosen as well as their numbers have the largest impact on the significance of the test data. Ideally, a random sample of potential users large enough to ensure statistical significance should be used as subjects.

Unfortunately, the number of test subjects necessary to ensure statistical significance is dependent on the expected variance in the data to be gathered. Therefore, as a first step, the test pool suggested below will provide a starting point to estimate the experimental parameters. The data gathered from this group cannot be assured to be statistically significant *a priori* but, even without those assurances, it should provide insight into whether the claims are supported at all by experimental evidence. This initial data will also be conducive to refining these experiments and provide guidance for a better selection of the subject pool for similar studies undertaken by other researchers.

The time requirement for each participant is 2 hours. The actual number and skill sets of the participants is generally governed by the breakdown of people volunteering to participate. As an initial guideline, the desired test participants are as follows:

1. $3-6$ ROTC students

2. 3 – 6 CS students familiar with C
3. 3 – 6 individuals familiar with the MissionLab toolset
4. 3 – 6 participants with random skill levels

Software

1. GNU C compiler version 2.7 or newer
2. MissionLab Toolset version 1.0 with logging enabled

Tasks

1. Deploy a robot to move to a flag, return to home base, and stop.
2. Deploy a robot to retrieve mines one by one, returning each to the Explosive Ordinance Disposal (EOD) area. When all the mines are safely collected, the robot should return home and stop.
3. Deploy a robot to retrieve a specified flag while avoiding surveillance. Allow the robot to move only when the mission commander signals it is safe.
4. Deploy a robot to explore a mine field. Each possible mine must be probed. If it is dangerous mark it as a mine; if it is safe, mark it as a rock. The robot should return home when all unknown objects are marked.
5. Deploy a robot for sentry duty. Chase and terminate any enemy robots, then return to guarding home base.

Programming Model

1. All of the configurations created by the participants are executed in simulation for this evaluation. Since we are gathering metrics concerning the mission development process and not concentrating on their execution, it is felt that little would be gained by imposing the additional complexity required to deploy each configuration on real robots. This also allows the simulated hardware to be idealized to reduce complexity.
2. The simulated robots possess a complete and perfect sensor model, allowing determination of the identity, color, and relative location of all objects within range in their environment with a single sensor reading.
3. The environmental objects are partitioned into four physical classes: Fixed, movable, containers, and robots. Each object can be any color although, for these experiments a classification based on color is created and enforced. Mines are orange, enemy robots are red, flags are purple, EOD areas (containers where mines can be placed) are green, rocks are black, trees and shrubs are dark green, home base is a white rectangle, and unknown objects (either a mine or a rock) are brown.

4. When a mine is being carried by a robot or residing within one of the EOD areas it is not visible to any of the robot's sensors. This includes the robot carrying the object and any other robots operating within the environment.

5. To simplify the control software, the robots in this study are idealized holonomic vehicles. This means that they can move in any direction and need not deal with turning radius issues. The system does not simulate vehicle dynamics; only the maximum robot velocity is restricted.

These idealizations and simplifications result in a straightforward programming model presented to the test participants. It becomes easier to explain and for them to understand the requirements without detracting from the validity of the mission configuration usability experiments themselves. Since the modifications apply equally to each participant, any resulting bias is eliminated in the comparisons.

7.1.3 Experimental Procedure

The participants were given oral and written specifications for a series of five tasks, one at a time, and instructed to create robot configurations which fulfilled those mission requirements. The same uninvolved third party was used as the instructor for all of the experiments. All interactions between the instructor and the participant were scripted to ensure consistency. If any questions were asked by participants after they had begun a task, that session was marked as incomplete.

1. Participants read and signed the required informed consent form.

2. Participants were given a tutorial introduction to the *MissionLab* graphical configuration editor. This provided an introductory overview of the *MissionLab* toolset and helped the participants become familiar with using the system. The tutorial script had the participants construct a simple configuration in cooperation with the person monitoring the experiments. The task was to cause a robot to move around picking up mines. The observer assisted the participants in completing this task, using it to demonstrate usage of the toolset.

3. Repeated for each of the 5 tasks:

   (a) Gave the participants the next task description and asked them to construct a configuration which achieved it.

   (b) At this point, the observer left the room and only offered assistance when the test participants asked for aid. This policy allowed the test participants to decide for themselves when they reached a stumbling block, and kept the observer from interjecting help when it may not have been required. This also allowed all help to be logged and attempted to prevent bias from creeping into the experiments from unequal amounts of help being given to certain participants.

   (c) The test participants used the configuration editor to construct a configuration which performed the desired task, compiling and testing their solutions using the *MissionLab* compilation facilities and simulation system.

   (d) When the user-created configuration correctly completed the task, or if the participants were not finished within 20 minutes, the testing observer re-entered the room. If the

participant believed they had completed the task, they then demonstrated their solution in both of the test worlds provided. The experiment was only marked as successful if their solution performed correctly in both test cases. At this point any questions were answered and, if the participants' solutions were incomplete or incorrect, they were corrected and missing portions explained before the next task was introduced.

4. After completing as many of the tasks as possible within 2 hours, the session concluded with a survey.

7.1.4 Nature and Type of Data Generated

Metrics measuring the performance of each participant are gathered by instrumenting the *MissionLab* system, by the experiment observer, via video tape, and through participant surveys. The results are used to determine how the *MissionLab* system performs against the results gathered in Experiment 2, and to evaluate its usability in general. For both Experiment 1 and Experiment 2, at least the following data values were generated for each subject completing one of the 5 tasks:

1. Time expended creating each configuration until first compile.
2. Log of the durations of compilations.
3. Log of the length of intervals between compilations.
4. Number of compilations before running the simulator.
5. Number of compilations after first simulation until each task is completed.
6. Time required to finish each task. If the participant fails to complete the task, the observer estimates their progress towards a solution (0, $\frac{1}{4}$, $\frac{1}{2}$, $\frac{3}{4}$).

7.2 Experiment 2: Mission Specification using C

7.2.1 Objective

Determine the performance of participants on tasks similar to Experiment 1 when using a traditional programming language.

This experiment is intended to provide data allowing a direct comparison to the data gathered in Experiment 1. Ideally, the same subject pool should perform both this experiment and Experiment 1 in a random order. There should also be at least a one day break between the two experiments. Of course, participants who are not programmers will be unable to perform this experiment. Given the goal of duplicating as closely as possible conditions in Experiment 1, the procedures and tasks are the same as in Experiment 1 except for differences noted below.

7.2.2 Experimental Setup

Same as Experiment 1, with the following exceptions:

Desired Test Participants

Same as Experiment 1, except for the additional restriction that they need to be fluent in the C programming language.

Software

1. GNU C compiler version 2.7 or newer
2. Current versions of `vi` and `emacs` editors
3. MissionLab simulation system Version 1.0

7.2.3 Experimental Procedure

Same as Experiment 1, except that the tutorial also presents a library of behaviors and perceptual triggers that can be called from a traditional programming language. Instead of presenting the graphical editor, the mechanics of editing, compiling, and running programs are presented.

The participants are given the exact same task descriptions as Experiment 1 and asked to construct configurations by hand to achieve them. Test participants are allowed to use their favorite text editor to construct the configurations, and they evaluate their solutions using the same *MissionLab* simulation system as in Experiment 1.

An equivalent set of motor behaviors and perceptual activities (triggers) are provided as callable functions. A sample C program which causes the robot to simply wander about is given to the participants as a base on which to create their solutions. Effectively, the subject's job is to construct by hand the mission states and transitions that CfgEdit generates from the graphical descriptions. This is intended to give programmers every advantage in reproducing the *MissionLab* capabilities. Starting them out with less support would force them to take far longer to create a solution.

7.2.4 Nature and Type of Data Generated

Metrics measuring the performance of each participant are gathered by instrumenting the build and run scripts, by the experiment observer, via videotape, and through participant surveys. There are no event logging capabilities available during the editing process as in Experiment 1. Therefore, the data gathered during this experiment needs to center on logging when they start and stop editing, compiling, and running their configurations. As a minimum, the following data values are generated:

1. Time expended creating each configuration until first compile.
2. Log of durations of compilations.
3. Log of intervals between compilations.
4. Number of compilations before running the simulator.
5. Number of compilations after first simulation until each task is completed.
6. Time required to complete each task. If the participant fails to complete the task, the observer will estimate their progress towards a solution (0, $\frac{1}{4}$, $\frac{1}{2}$, $\frac{3}{4}$).

8 Evaluating Usability Experimental Results

The purpose of usability experiments is to establish values for certain usability criteria. Although raw data is gathered in several forms, the most detailed information is found in event logs generated by the software while participants perform the experiments. A careful design of the event logging facilities is necessary to ensure that the desired facets are properly represented in the logs. A simple format represents each entry as having a time stamp, the keyword `start`, `end`, or `event` followed by information to identify the event or action which took place. This type of log file is easy to generate by instrumenting the software. It is also straightforward to generate a parsing tool to compute statistics from the event logs.

If we are interested in establishing the duration of `Action01` based on the event logs, the parse tool would extract a list of the durations of the `Action01` events. These raw durations can then be correlated with the classes of users, visualized to look for correlations, and meaningful statistics such as mode, mean, and standard deviation can be computed. Using these statistics, the measured values for the usability criteria can then be determined.

9 Results of *MissionLab* Usability Experiments

9.1 Experiment 1 Results and Evaluation

Usability experiment 1 was conducted using the *MissionLab* toolset in order to establish values for the various usability metrics for novice and expert users. The task involved specifying benchmark robot missions using the graphical editor. The tests were conducted in the Georgia Tech Usability Lab. The lab is outfitted with one-way mirrors and video cameras which allow monitoring and recording the experiments from outside the room. A third party conducted the experiments to ensure consistency and impartiality. All experiments were videotaped, and logging data was gathered both by the proctor and automatically through the *MissionLab* software.

Twelve people participated in this experiment and are identified with a numeric code ranging from 1 to 12. The skill set of the participants was as follows:

- 1 ROTC student:
  Participant 12.
- 3 people familiar with the MissionLab toolset:
  Participants 2, 5, and 6.
- 4 people with no programming experience:
  Participants 1, 3, 7, 10, and 12 (Note that 12 is also the ROTC student).
- 4 people with programming skills, but no MissionLab experience:
  Participants 4, 8, 9, and 11.

This experiment required participants to construct solutions, similar to the one shown in Figure 7, for each of the five tasks described earlier. Figure 8 shows an annotated portion of an event log generated automatically by the *MissionLab* system while a user constructed a configuration. The logs can be used to reconstruct the number and duration of many types of events occurring during the experiments. Events include adding states and transitions, selecting new agents for tasks and

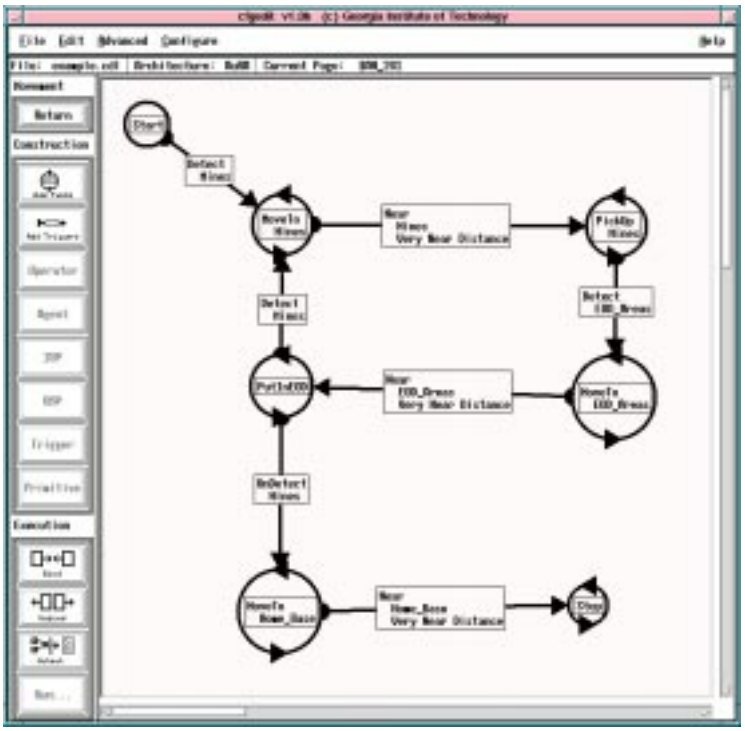

Figure 7: A representative task solution for Experiment 1.

triggers, parameterizing those agents, and compilation and execution of the configurations. For example, the time to specialize a step (Modify Agent) occurs in Figure 8 from time 58.3 to time 61.8 in the log. This interval started when the user clicked the right mouse button on state 2 to choose a new task and ended when the user selected `MoveTo` from the popup menu of tasks.

A statistical analysis of the variance in the measured parameters is necessary to determine the significance of the data gathered. Computing this variance allows researchers to understand to what extent the data is predictive for future research. Comparisons between Experiment 1 and Experiment 2 should be made as paired samples when the same person performs both experiments since the performance of different subjects likely has more variability than for the same individual. The detailed analysis of these experiments can be found in [20].

As a concrete example, consider how the `Time to specialize a step` value was determined. When a new step is added to a mission it defaults to the `Stop` behavior. Usually this is not the desired behavior and the step must be specialized by selecting the correct behavior from a popup menu. The `Time to specialize a step` attribute measures the time it takes for a user to complete this specialization task. This action is measured from the time the user clicks the middle mouse button on the state until the left mouse button is clicked on the `OK` button in the popup window. These points are denoted in the event logs with the `StartModify Agent` and `EndModify Agent` events.

```
        // Information to identify event file //
  0.0: start Session
  0.1: status StartTime "827354547.5"
  0.2: status Task "4"
  0.3: status Subject "0"
        // A new state was added to workspace //
 16.2: start PlaceState "State1"
 16.8: end PlaceState
        // A state was moved to a new location //
 19.9: start Move
 20.7: end Move
        // A transition was added to connect two states //
 21.7: start AddTransition Trans1
 22.8: status FirstState
 23.6: end AddTransition
        // State2 was changed to the MoveTo behavior //
 58.3: StartModify Agent State2 "Stop"
 61.8: EndModify Agent "MoveTo"
        // Unknown objects targeted for MoveTo //
 64.3: StartModify Parms State2 "MoveTo   None"
 67.1: EndModify Parms "MoveTo   Unknown objects"
        // Transition 1 was changed to Detect trigger //
276.1: StartModify Agent Trans1 "FirstTime"
280.5: EndModify Agent "Detect"
        // Transition 1 was changed to detect Mines //
340.9: StartModify Parms Trans1 "Detect   None"
343.9: EndModify Parms "Detect   Mines"
        // Configuration compiled successfully //
538.4: event StartMake
602.6: event GoodMake
605.7: event EndMake
        // The Configuration was executed //
607.2: start Run
678.7: end Run
824.2: end Session
```

Figure 8: An annotated portion of a *MissionLab* event log. Comments are enclosed in // // brackets. The numbers are the time the event occurred (in seconds) after the start of the experiment.

Figure 9 graphically shows the length of time taken by each participant to specialize steps. The graphs for each participant are stacked on top of each other for ease of comparison. Figure 10 is a histogram showing the distribution of this data. The horizontal resolution of this graph is 1 second. The peak in this graph marks the mode at 3 seconds. This suggests that expert users require about 3 seconds to choose a new behavior for a step. The measured value for the `Time to specialize a step` attribute for novice users is computed as the average of the 260 data points. This works out

to 6.15 seconds with a very high standard deviation of 5.86 seconds. The 30 second target value was easily surpassed by these novice users. The estimated time for a programmer to modify a C file to invoke a different behavior was 2 minutes, showing the benefits *MissionLab* users gain.

The long right-hand tail on the distribution graph as well as the variability in Figure 9 appear to show a consistent difference in performance between novice and expert users on completing this action. Looking at Figure 9, Participants 5 and 6 did quite well on this task and generated times consistently in the 5 second range. Compare those records with Participants 7, 10, 11, and 12 who exhibit far greater variability and numerous times in the 20 and 30 second ranges. These long periods are instances where the users were confused about which behavior to select. This suggests that the method used to present the behavior choices is confusing and may require reworking to be useful to people unfamiliar with robotics.

Values for the remaining usability criteria relevant to experiment 1 were established similarly. The actual values for the usability criteria were estimated using the average durations measured during the experiment. Figure 11 presents these results in tabular form. Notice that all times were less than 25% of the target values, and many show far better improvements. This demonstrates that the system is quite easy for novices to use to construct and evaluate robot missions.

9.2 Experiment 2 Results and Evaluation

Experiment 2 was used to provide a direct comparison of the performance of participants using the graphical editor versus the traditional programming language C. Participants from the same subject pool performed both this experiment and Experiment 1 in a random order. There was a several day break between the two experiments to attempt to minimize the benefits associated with repeating the same tasks. Of course, participants who were not programmers were unable to perform this experiment. The primary goal was to duplicate conditions in Experiment 1 as closely as possible except for the use of the C programming language.

Each of the people who volunteered for Experiment 1 were asked if they were fluent in the C programming language. Those who were able to program in C were asked if they would be able to take part in two sessions. Three participants (4,5 and 6) could program in C but were unable to participate in more than one session and only completed Experiment 1. Five of the participants (1,3,7,10 and 12) were unable to program in C and therefore didn't complete Experiment 2.

This left four participants (2,8,9, and 11) who completed both Experiment 1 and Experiment 2. The numeric codes assigned to these participants match those from Experiment 1. Two of the participants were randomly selected to complete Experiment 2 before Experiment 1 and the others did Experiment 1 first.

A library of C functions which reproduced the behaviors available in the graphical editor was created and provided to the participants. A stub program and scripts to build and execute the configurations required the participants only to create a suitable state machine to complete the missions. The standard UNIX text editors `vi` and `emacs` were available for the participants' use. The same *MissionLab* simulation system was used to evaluate their solutions as in Experiment 1.

Due to the use of standard UNIX tools, the ability to automatically log editing events was lost in this experiment. The videotape taken of the experiments was shot over the shoulder of the test participants and not of sufficient quality to recreate their edit session. However, by instrumenting the build and run scripts, useful information was still gathered. Figure 12 shows an annotated event log from this experiment. The comments are enclosed in `// //` brackets. The start of the experiment

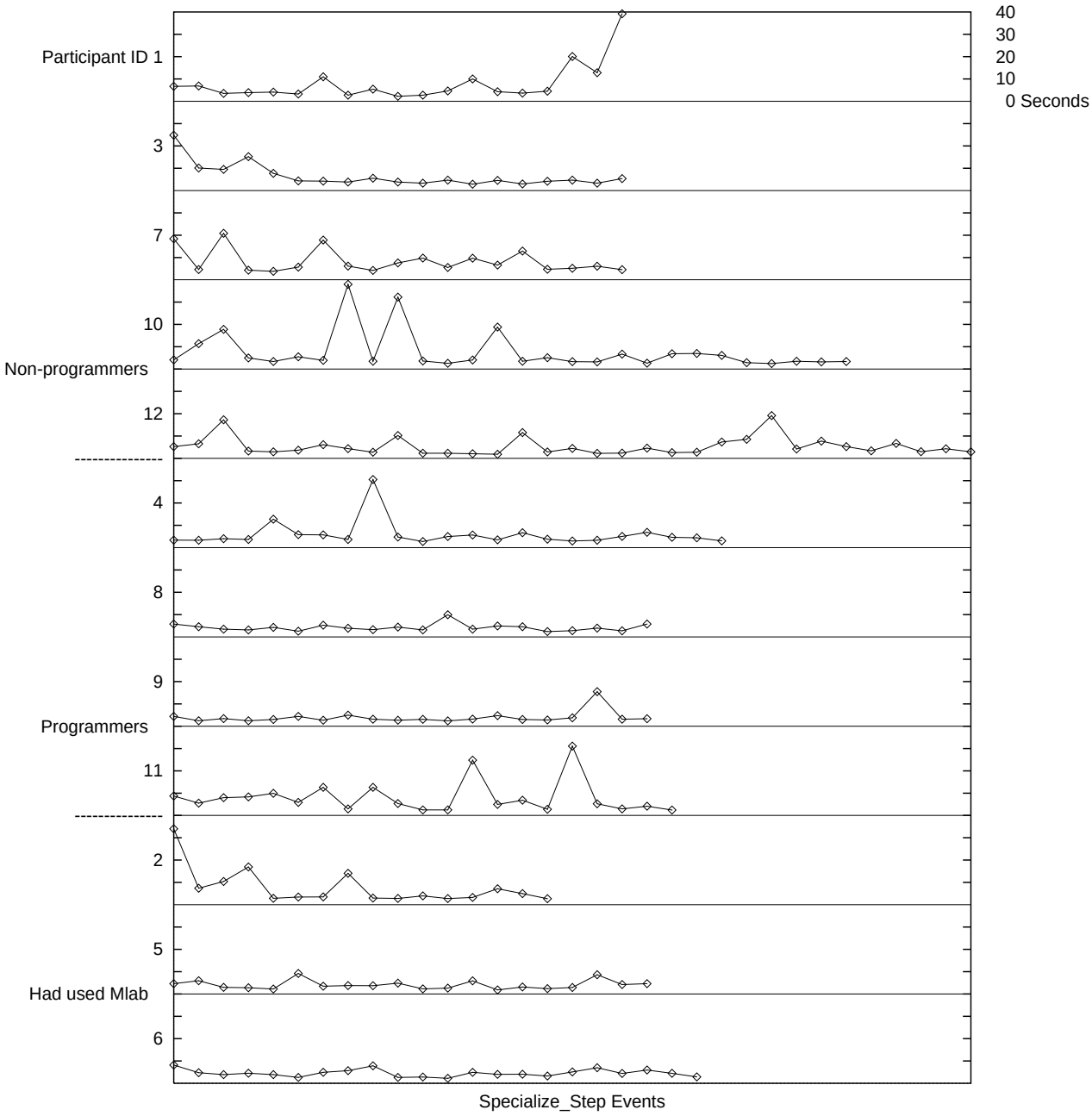

Figure 9: **Time to specialize a step**
The vertical scale is 40 seconds. The number of actions varied based on how users proceeded in the development.

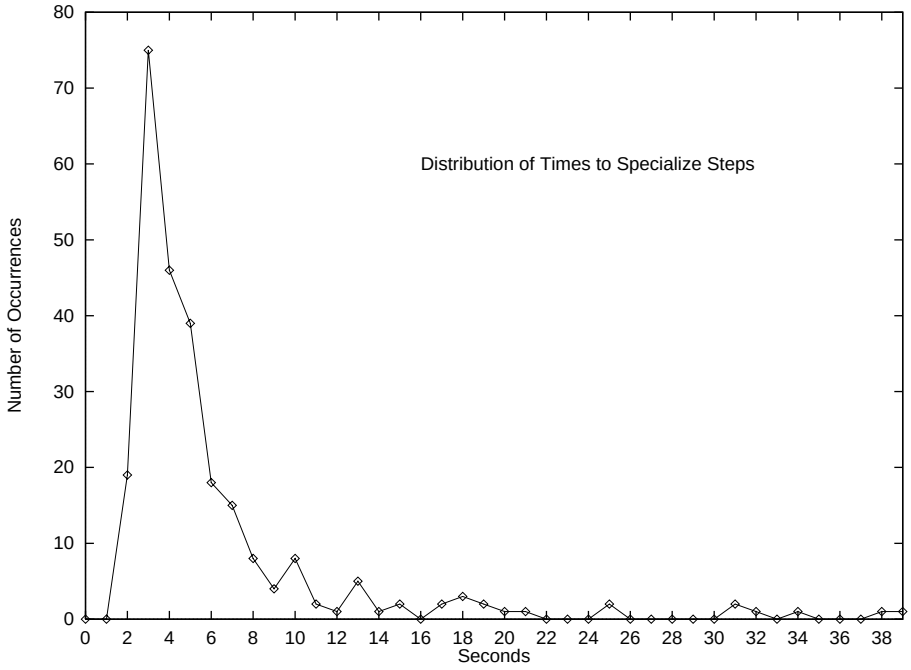

Figure 10: Distribution of the time required to specialize mission steps. The Mode occurs at 3 seconds, suggesting that users will be able choose new behaviors for steps in about 3 seconds after they have gained experience with the toolset. The large right-hand tail suggests that some users are having difficulty choosing behaviors. Steps to simplify this process warrant attention in future studies.

Novice User Performance			
Value to be Measured	*Target Level*	*Measured Value*	*Standard Deviation*
Time to add a mission step	10 sec	2.2 sec	1.9 sec
Time to specialize a step	30 sec	6.2 sec	5.9 sec
Time to parameterize a step	30 sec	4.1 sec	2.1 sec
Time to add a mission transition	10 sec	2.6 sec	1.5 sec
Time to specialize a transition	30 sec	4.9 sec	5.0 sec
Time to parameterize a transition	30 sec	4.0 sec	2.9 sec
Number of compiles to create configuration	2	2.0	1.7 sec
Time to create a simple configuration	15 min	7.4 min	2.4 min

Figure 11: Experiment 1 established values for the usability criteria

is logged, along with the task number. The start and end times for each compile are also recorded. This allows counting the number of compilations as well as computing the time the participant spent editing the configuration. The start and end time for each execution of the configuration in the simulation system is also logged.

Figure 13 shows a representative solution for a task in Experiment 2. Each participant constructed a `robot_command` function which called the library of behaviors and perceptual processes to complete the mission. This support library exactly matched those available in the graphical configuration editor.

9.2.1 Time to create a simple configuration

Figure 14 presents edit times for the participants using the C programming language. Figure 15 graphs this data and also the corresponding time spend editing in Experiment 1 for the four people who participated in both experiments to allow a closer comparison. There are 12 instances where the time taken using the graphical user interface (GUI) is less that when using C. There is one clear case of the GUI taking longer and 4 examples where the times are quite similar. These results show using the GUI speeds the development process.

Notice that only Subjects 2 and 11 were able to do better using C than with the GUI (on only one of the 5 tasks each). This is interesting since Subjects 2 and 11 performed the GUI portion (Experiment 1) before the C session (Experiment 2). It appears that there is a speed-up from performing the same experiments again using the other modality. However, even these participants performed the tasks faster using the GUI for the other tasks.

The experiment was structured to allow a direct comparison between the graphical configuration editor and the C programming language. The results, summarized below, clearly demonstrate the

```
// Started the experiment //
Wed 10:56:37 AM, Mar 20 1996
         Starting task 3
// 1st build of the solution //
Wed 11:03:28 AM, Mar 20 1996
         Start make
Wed 11:03:36 AM, Mar 20 1996
         End make
// 1st build of the solution //
Wed 11:04:07 AM, Mar 20 1996
         Start make
Wed 11:04:11 AM, Mar 20 1996
         End make
// 2nd build of the solution //
Wed 11:05:08 AM, Mar 20 1996
         Start make
Wed 11:05:23 AM, Mar 20 1996
         End make
// 1st run to check correctness //
Wed 11:05:24 AM, Mar 20 1996
         Start run
Wed 11:05:54 AM, Mar 20 1996
         End run
// 2nd run to check correctness //
Wed 11:06:20 AM, Mar 20 1996
         Start run
Wed 11:06:57 AM, Mar 20 1996
         End run
```

Figure 12: An annotated portion of an event log from Experiment 2. Comments are enclosed in `// //` brackets.

advantages of using the graphical editor over hand-crafting solutions in C. For the 4 people who completed both Experiment 1 and Experiment 2:

- In 12 instances participants completed a task faster using the *MissionLab* configuration editor than they completed the same task using `C`.
- In only one instance did a participant complete a task faster using `C` than using the configuration editor. Note: This occurred on Task 5 and the participant had previously completed the GUI portion.
- In 4 cases times were similar.
- In general, the times required to generate solutions using the configuration editor were more consistent.
- The average time required by the 4 participants for each task was 12.4 minutes using `C` and 5.9 minutes using the configuration editor.

```
Vector robot_command()
{
  static int status = 0;
  if(SigSense(SAFE))
  {
    switch (status)
    {
      case 0:   /*  At start  */
        status = 1;
        return MoveTo(flags);
      case 1:  /*  On way to flag  */
        if(Near(flags,0.1))
        {
          status = 2;
          return Stop();
        }
        return MoveTo(flags);
      case 2:  /*  At flag  */
        status = 3;
        return Stop();
      case 3:
        if (Near(home_base,0.1))
          return Stop();
        return MoveTo(home_base);
    }
  }
  else return Stop();
}
```

Figure 13: A representative task solution

- The average number of compilations was 4 using `C` and only 2 using the configuration editor.

9.3 Summary of Experimental Results

Values for various usability criteria for the graphical editor were established using event logging data gathered in Experiment 1. Table 3 is a reproduction of Table 2 with the measured values column added. This presents the usability criteria in tabular form for ease of comparison. Notice the measured values are all far superior to the expected values. However, the large amount of variance in the time users spent picking new behaviors and perceptual activities points out some remaining difficulty in that area. A popup window currently presents an alphabetical list of choices, each with a short description. More effort is needed in both naming, describing, and visualizing the behaviors and perceptual activities in order to make their usage more apparent.

Edit time using C					
Participant	*Task 1*	*Task 2*	*Task 3*	*Task 4*	*Task 5*
2[1]	547	1383	951	600	630
8[2]	834	1342	526	766	—
9[2]	635	626	413	659	538
11[1]	1057	725	407	1269	241

1. Performed the GUI tasks first.
2. Performed the C tasks first.

Figure 14: Total edit time (in seconds) when using the C programming language. A dash (—) indicates there wasn't time to work on that task.

10 Conclusions

In order to ensure the acceptance of these products by end-users, usability methods must of necessity be introduced to robotics. This article has described methods and techniques by which robot programming toolsets can be analyzed along these lines. Choosing target values for usability criteria provides designers with metrics useful to determine where to focus their efforts and a yardstick to establish when development is complete.

The means by which usability experiments can be designed and administered to generate and analyze data relevant for this evaluation process has been provided. It is important that such experiments be well designed and impartially administered in order to minimize bias and variability in the data.

Before robots can move into common use among non-programmers it is necessary that they become easier to task. Mission specification must be straightforward and routine. This article has presented a foundational methodology for the evaluation of robotic toolsets, hopefully, leading others in the field to consider the needs of target consumers of this new technology in a new light.

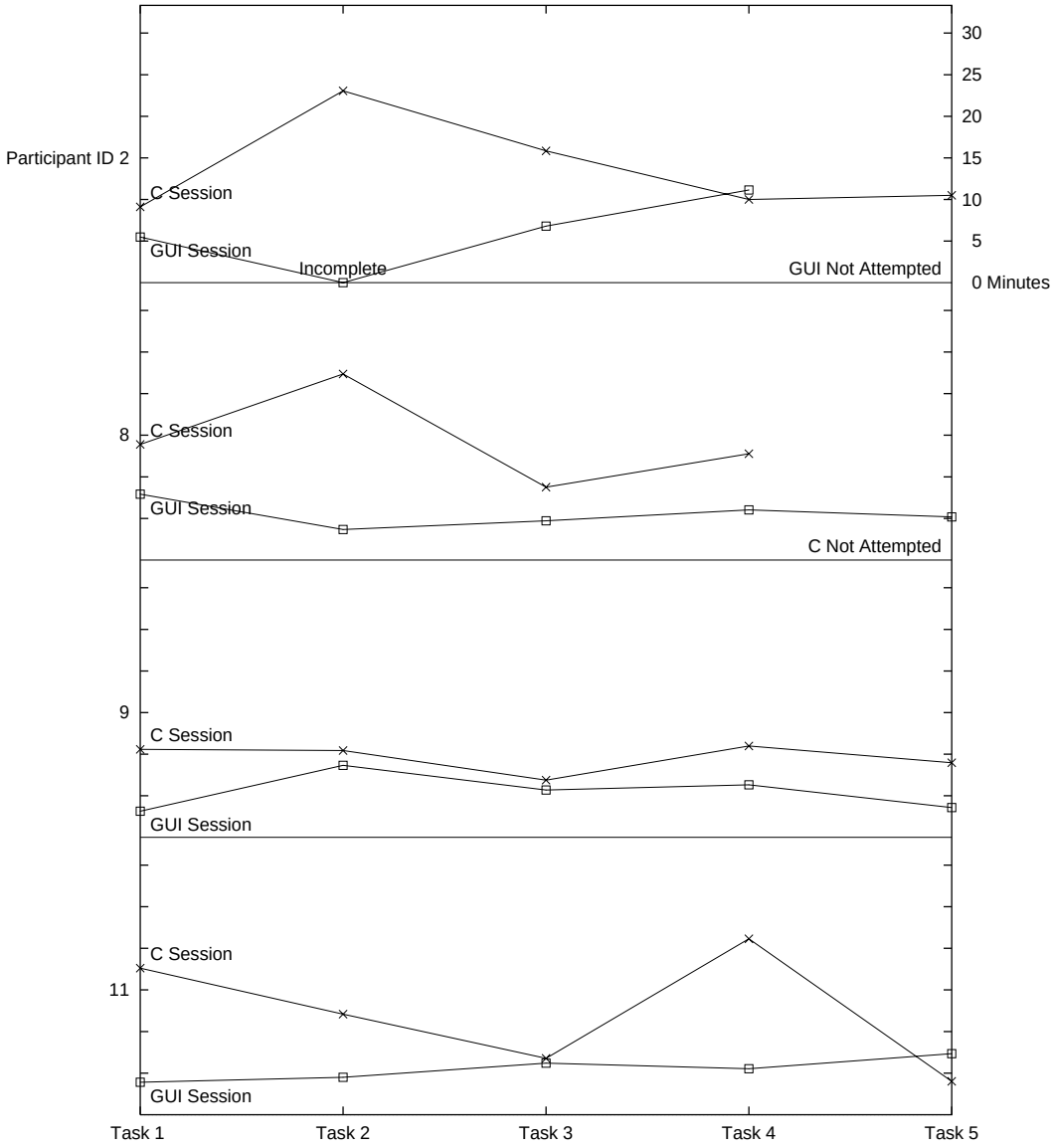

Figure 15: Graph of the time spent editing in both the GUI and C sessions. GUI sessions are marked with □ and C sessions with ×. The vertical axis represents time required to complete the tasks.

Table 3: The *MissionLab* usability criteria with the experimentally measured values included.

***MissionLab* Usability Criteria**						
Usability Attribute	Value to be Measured	Current Level	Worst Acceptable Level	Target Level	Best Possible Level	Measured Value
1. Novice user performance	Time to add a mission step	1 Min	30 sec	10 sec	1 sec	2.2 sec
2. Novice user performance	Time to specialize a step	2 min	1 min	30 sec	3 sec	6.2 sec
3. Novice user performance	Time to parameterize a step	1 min	1 min	30 sec	2 sec	4.1 sec
4. Novice user performance	Time to add a mission transition	1 min	30 sec	10 sec	2 sec	2.6 sec
5. Novice user performance	Time to specialize a transition	2 min	1 min	30 sec	3 sec	4.9 sec
6. Novice user performance	Time to parameterize a transition	1 min	1 min	30 sec	2 sec	4.0 sec
7. Novice user performance	Number of compiles to create a configuration	4	5	2	1	2.0
8. Novice user performance	Time to create a simple configuration	20 min	20 min	15 min	5 min	7.4 min
9. Non-programmer performance	Ability to create configurations	No	Yes	Yes	Yes	Yes
10. User acceptance	General feeling after use	N/A	medium	good	great	good

References

[1] R.C. Arkin. Motor schema-based mobile robot navigation. *The International Journal of Robotics Research*, 8(4):92–112, August 1989.

[2] R.C. Arkin and T.R. Balch. AuRA: Principles and practice in review. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(2):175–189, 1997.

[3] R.C. Arkin and D.C. MacKenzie. Temporal coordination of perceptual algorithms for mobile robot navigation. *IEEE Transactions on Robotics and Automation*, 10(3):276–286, June 1994.

[4] N.A. Aykin, J.P. Cunningham, and J.P. Rotella. Designing operations systems interfaces that are easy to use. *AT&T Technical Journal*, 73(4):14–21, July/August 1994.

[5] R.A. Brooks. A robot that walks: Emergent behaviors from a carefully evolved network. *Neural Computation*, 1(2):253–262, 1989. Also MIT AI Memo 1091.

[6] Jonathan M. Cameron and Douglas C. MacKenzie. *MissionLab User Manual.* College of Computing, Georgia Institute of Technology, Available via *http://www.cc.gatech.edu/ai/robot-lab/research/MissionLab/mlab_manual.ps.gz*, Version 1.0 edition, May 1996.

[7] John P. Chin, Virginia A. Diehl, and Kent L. Norman. Development of an instrument measuring user satisfaction of the human-computer interface. In E. Soloway et al., editors, *Proc. CHI'88, Human Factors in Computing Systems*, pages 213–218. ACM, 1988.

[8] J. Connell. A colony architecture for an artificial creature. AI Tech Report 1151, MIT, 1989.

[9] J. Firby. Adaptive execution in complex dynamic worlds. Computer Science Tech Report YALEU/CSD/RR 672, Yale, January 1989.

[10] Susan L. Fowler and Victor R. Stanwick. *The GUI Style Guide.* Academic Press, Cambridge, MA, 1995.

[11] Matthew W. Gertz, Roy A. Maxion, and Pradeep K. Khosla. Visual programming and hypermedia implementation within a distributed laboratory environment. *Intelligent Automation and Soft Computing*, 1(1):43–62, 1995.

[12] J. Gowdy. *SAUSAGES Users Manual.* Robotics Institute, Carnegie Mellon, version 1.0 edition, February 8 1991. SAUSAGES: A Framework for Plan Specification, Execution, and Monitoring.

[13] J. Gowdy. SAUSAGES: Between planning and action. Technical Report Draft, Robotics Institute, Carnegie Mellon, 1994.

[14] Deborah Hix and H. Rex Hartson. *Developing User Interfaces.* John Wiley and Sons, New York, 1993.

[15] William Horton. *The Icon Book.* John Wiley and Sons, New York, 1994.

[16] Jaeho Lee, Marcus J. Huber, Edmund H. Durfee, and Patrick G. Kenny. UM-PRS: An implementation of the procedure reasoning system for multirobot applications. In *Proceedings AIAA/NASA Conference on Intelligent Robots in Field, Factory, Service, and Space (CIRFFSS '94)*, 1994.

[17] Michelle A. Lund. Evaluating the user interface: The candid camera approach. In L. Borman et al., editors, *Proc. CHI'85, Human Factors in Computing Systems*, pages 107–113. ACM, 1985.

[18] Damian M. Lyons and M. A. Arbib. A formal model of computation for sensory-based robotics. *IEEE Journal of Robotics and Automation*, 5(3):280–293, June 1989.

[19] Douglas C. MacKenzie. *Configuration Network Language (CNL) User Manual.* College of Computing, Georgia Institute of Technology, Available via *http://www.cc.gatech.edu/ai/robot-lab/research/MissionLab/cnl_manual.ps.gz*, Version 1.5 edition, June 1996.

[20] Douglas C. MacKenzie. *A Design Methodology for the Configuration of Behavior-Based Mobile Robots.* Ph.D. dissertation, Georgia Institute of Technology, College of Computing, 1997. GIT-CS-97/01.

[21] Douglas C. MacKenzie, Ronald C. Arkin, and Jonathan M. Cameron. Multiagent mission specification and execution. *Autonomous Robots*, 4(1):29–52, 1997.

[22] M. J. Mataric. Designing emergent behaviors: From local interactions to collective intelligence. In *Proceedings From Animals to Animats, Second International Conference on Simulation of Adaptive Behavior (SAB92).* MIT Press, 1992.

[23] James R. Miller and Robin Jeffries. Usability evaluation: Science of trade-offs. *IEEE Software*, pages 97–102, September 1992.

[24] M. Minsky. *The Society of Mind.* Simon and Schuster, New York, 1986.

[25] Jakob Nielsen. Finding usability problems through heuristic evaluation. In *Proc. CHI '92: Striking a Balance - Conference on Human Factors in Computing Systems*, pages 373–380, Monterey, CA, 1992.

[26] R.E. Opaluch and Y.C. Tsao. Ten ways to improve usability engineering - designing user interfaces for ease of use. *AT&T Technical Journal*, 72(3):75–88, May/June 1993.

[27] Julio K. Rosenblatt. Damn: A distributed architecture for mobile navigation. In *AAAI Spring Symposium: Lessons Learned from Implementing Software Architectures for Physical Agents*, volume 1, pages 167–178, 1995.

[28] Julio K. Rosenblatt and D.W. Payton. A fine-grained alternative to the subsumption architecture for mobile robot control. In *IEEE INNS International Joint Conference on Neural Networks*, volume 2, pages 317–323, 1989.

[29] David B. Stewart and P.K. Khosla. Rapid development of robotic applications using component-based real-time software. In *Proc. Intelligent Robotics and Systems (IROS 95)*, volume 1, pages 465–470. IEEE/RSJ, IEEE Press, 1995.

[30] MissionLab Toolset. *Mobile Robot Laboratory.* College of Computing, Georgia Institute of Technology, Available via *http://www.cc.gatech.edu/ai/robot-lab/research/MissionLab*, Version 1.0 edition, May 1996.

[31] M. Turnell and J. de Queiroz. Guidelines - an approach in the evaluation of human-computer interfaces. In *Proc. IEEE International Conference on Systems, Man, and Cybernetics*, volume 3, pages 2090–2095, 1996.

[32] R.A. Virzi, J.L. Sokolov, and D. Karis. Usability problem identification using both low- and high-fidelity prototypes. In *Proc. CHI '96: Common Ground - Conference on Human Factors in Computing Systems*, pages 236–243, Vancouver, BC Canada, 1996.

[33] Susan Weinschenk and Sarah C. Yeo. *Guidelines for Enterprise-Wide GUI Design*. John Wiley and Sons, New York, 1995.

Douglas C. MacKenzie received the B.S. degree in Electrical Engineering in 1985 and the M.S. degree in Computer Science in 1989, both from Michigan Technological University. He was employed as a Software Engineer in Allen-Bradley's Programmable Controller Division in Cleveland, Ohio from 1988 until 1990. He attended the Georgia Institute of Technology from 1990 until graduating in 1997 with a Ph.D. in Computer Science. His research interests center around artificial intelligence as applied to mobile robotics and he has published over 13 related articles. Dr. MacKenzie is currently the founding president of Mobile Intelligence Corporation, which focuses on developing tools to reduce the complexity of the operator/robot interface. Dr. MacKenzie can be reached at `doug@mobile-intelligence.com` and many of his publications can be downloaded as technical reports from `www.mobile-intelligence.com`.

Ronald C. Arkin is Professor and Director of the Mobile Robot Laboratory in the College of Computing at the Georgia Institute of Technology. Dr. Arkin's research interests include reactive control and action-oriented perception for the navigation of mobile robots, robot survivability, multi-agent robotic systems, learning in autonomous systems, and cognitive models of perception. He has over 90 technical publications in these areas. Funding sources have included the National Science Foundation, DARPA, Savannah River Technology Center, U.S. Army, and the Office of Naval Research. Dr. Arkin has served as an Associate Editor for IEEE Expert, and is a Member of the Editorial Board of Autonomous Robots. He is a Senior Member of the IEEE, and a member of AAAI and ACM. Dr. Arkin can be reached at `arkin@cc.gatech.edu`.