Balancing Robotic Teleoperation and Autonomy in a Complex and Dynamic Environment

A thesis presented
by

Ryan Wegner

to
The Department of Computer Science
in partial fulfillment of the requirements
for the degree of
Master of Science
in the subject of

Computer Science

University of Manitoba
Winnipeg, Manitoba
July 2003

Abstract

While Artificial Intelligence has been working to produce truly autonomous problem solving agents for many years, the current abilities of such agents are extremely limited. In highly complex, dynamic situations such as disaster rescue, today's agents simply do not have the ability to perform successfully on their own: the environment is difficult to traverse and even to sense accurately, time is a significant factor, and the dynamic and unpredictable nature of the environment tends to preclude the ability to produce extensive plans for future activity. Because of these and other limitations, robotic agents for environments such as disaster rescue rely strongly on human teleoperation. This too has its limitations: humans become fatigued rapidly, suffer from cognitive overload when they obtain too much sensory information in a short time, and have difficulties in constructing a mental image of the space around a robot given information from its senses (situational awareness).

This thesis focuses on combining the limited abilities of an autonomous agent together with human control, in order to produce a teleautonomous system that supports blending the desires of a robot with the wishes of its human controller. The approach I present is intended to allow a human to control a number of robots, being interrupted only when the robots are truly in need, and with the ability to alter the autonomous abilities of the robots for particular contexts.

In order to examine the effectiveness of this approach, I develop a simulated domain for disaster rescue using a widely-employed robot simulation tool, and implement this control mode for a set of simulated Pioneer mobile robots. An evaluation of this control mode in comparison to autonomous and teleoperated agents is then presented.

Contents

1 Introduction **1**

1.1 Overview 1

1.2 Terminology 2

1.3 Urban Search and Rescue Domain 5

1.4 Balancing Autonomy and Teleoperation 8

1.5 Research Questions 13

1.6 Addressing Thesis Research Questions 15

1.7 Summary 16

2 Literature Review **17**

2.1 Overview 17

2.2 Agents 18

2.2.1 Planning Agents 18

2.2.2 Reactive Agents 21

2.2.3 Hybrid Approaches 29

2.3 Teleautonomy 31

2.4 User Interfaces 39

2.5 Summary 39

3 A Blending Approach to Mobile Robot Control 40

3.1 Overview 40

3.2 Autonomous Control 41

3.3 Teleoperation 45

3.4 Blending Teleoperation and Autonomy 47

3.4.1 Intervention Recognition 48

3.4.2 Mediator 51

3.5 User Interface 54

3.6 Summary 55

4 Implementation 56

4.1 Overview 56

4.2 Player/Stage 57

4.3 Simulated Agent 60

4.4 Implementation Language 62

4.5 Schemas 63

4.5.1 Perceptual Schemas 64

4.5.2 Motor Schemas 66

4.6 Autonomous Control System Implementation 69

4.7 Implementation of Teleautonomy 71

4.7.1 Joystick 72
4.7.2 Waypoint Manager 73
4.7.3 Autonomous Control Panel 75
4.7.4 Intervention Recognition Implementation 77
4.7.5 Mediator Implementation 79
4.8 User Interface 81
4.8.1 Vision 81
4.8.2 Map 83
4.9 Summary 87

5 Experimentation **88**

5.1 Overview 88
5.2 Purpose 89
5.3 Experimental Environment 89
5.3.1 Types of Objects 89
5.3.2 Generating and Evaluating Experimental Environments 91
5.3.3 Environment Evaluation 94
5.4 Methodology 96
5.5 Performance Evaluation 102
5.6 Results 104
5.6.1 Teleoperated Results 104
5.6.2 Autonomous Results 108

5.6.3 Blending 111

5.7 Analysis 116

5.8 Summary 123

6 Findings and Recommendations **124**

6.1 Overview 124

6.2 Findings and Analysis 125

6.3 Contributions 131

6.4 Future Work 132

6.5 Conclusion 135

List of Figures

1.1 Two example mobile robotic platforms. Left: The Trilobot mobile platform. Right: The Pioneer DX mobile platform. 3

1.2 Examples of the USAR Domain [Jacoff et al., 2001b] 6

1.3 The NIST standard test bed, Robocup 2003, Padua, Italy. 9

2.1 An example of potential fields being combined [Balch, 1998b]. Left: the potential field of a goal. Middle: the potential field of two obstacles. Right: the global potential field. 23

2.2 An example of Brooks' layered subsumption architecture [Brooks, 1986]. Higher level layers subsume lower level layers to control the system. . 24

3.1 Examples of victims in the NIST USAR standard test bed, Robocup 2003, Padua, Italy, July 2003 42

4.1 Example of communication in Player/Stage [Gerkey et al., 2003]. . . . 59

4.2 The joystick. 72

4.3 Example of an agent with several waypoints selected. 74

4.4 The autonomous control panel. 75

4.5 The motor schema panel for the noise motor schema. 77

4.6 Complete user interface. 82

4.7 The vision panel, currently a victim (larger) and a non-victim (smaller) are in the agent's field of view. 83

4.8 The map of the environment so far. 84

4.9 Screenshot of the user interface displaying an agent motor schema vectors as colored arrows. 86

5.1 Example of an environment generated by the environment generator. 92

5.2 Breakdown of trials 99

5.3 Average (n=5) environment coverage achieved by teleoperated agents in 5%, 10%, 15% and 20% obstacle coverage environments. 105

5.4 Average (n=5) number of victims identified by teleoperated agents in 5%, 10%, 15% and 20% obstacle coverage environments. 106

5.5 Average (n=5) number of interactions operators had with teleoperated agents in 5%, 10%, 15% and 20% obstacle coverage environments. . . 107

5.6 Average (n=5) environment coverage achieved by autonomous agents in 5%, 10%, 15% and 20% obstacle coverage environments. 109

5.7 Average (n=5) number of victims identified by autonomous agents in 5%, 10%, 15% and 20% obstacle coverage environments. 110

5.8 Average (n=5) time autonomous agents spent immobile in 5%, 10%, 15% and 20% obstacle coverage environments. 111
5.9 Average (n=5) environment coverage achieved by blending agents in 5%, 10%, 15% and 20% obstacle coverage environments. 112
5.10 Average (n=5) number of victims identified by blending agents in 5%, 10%, 15% and 20% obstacle coverage environments. 113
5.11 Average (n=5) time blending agents spent immobile in 5%, 10%, 15% and 20% obstacle coverage environments. 114
5.12 Number of interactions operators had with blending agents in 5%, 10%, 15% and 20% obstacle coverage environments. 114
5.13 Comparison of environment coverage in teleoperated, autonomous, and blending experiments. All results are averages over 5 trials. 118
5.14 Comparison of number of victims identified in teleoperated, autonomous, and blending experiments. All results are averages over 5 trials. . . . 119
5.15 Comparison of agent-operator interactions in teleoperated and blending experiments. All results are average over 5 trials. 121
5.16 Average time in milliseconds spent immobile by environment difficulty, for blending and autonomous agents. 122

Chapter 1

Introduction

1.1 Overview

The goal of this research is to provide an infrastructure to allow for a balance between autonomy and teleoperation for multiple mobile robotic agents operating in complex dynamic environments. Agents in these environments are limited in their ability to perform intelligently and in real time, causing them to occasionally fail to solve problems (e.g. getting hung up, not recognizing a goal) or becoming stuck in a local minimum. In any of these situations, a human operator or another agent may usefully intervene to assist the robot to overcome its current situational problem. On the other hand human operators become rapidly overloaded if they have to provide significant control for many robots or even a single agent for any length of time [Arkin and Ali, 1994]. Supporting a balance between the work of a remote operator and the autonomous abilities of a robotic agent can allow robots to be used in domains where

autonomous agents are currently not able to perform successfully, and also allows a single human operator to control a larger number of robotic agents. Specifically, this work explores recognizing situations within a domain that are problematic for an agent, the use of several different modes of control to support blends of operator control and autonomous decision making for different situations, and the adjustment of this blend in a context-dependent manner.

In this chapter I will begin by introducing some terminology that will be used throughout this work. Secondly, I will describe the urban search and rescue domain, an example of an extremely complex dynamic domain that will serve as the experimental setting for this research. Once the terminology is introduced and the domain has been described, I will discuss reasons for blending autonomy and teleoperation.

1.2 Terminology

This thesis deals with the topic of balancing autonomy in robotic control with remote control by a human operator in mobile robots situated in dynamic complex environments. Before the topic can be properly discussed, some terminology will be presented here that will be used throughout this work.

First, what is an *agent*? In this work the term "agent" refers to a mobile robot. The mobile robots referred to in this work are mechanical platforms containing actuators that enable the robot to move around and interact with a physical environment, and sensors that enable the robot to perceive information from the environment (see

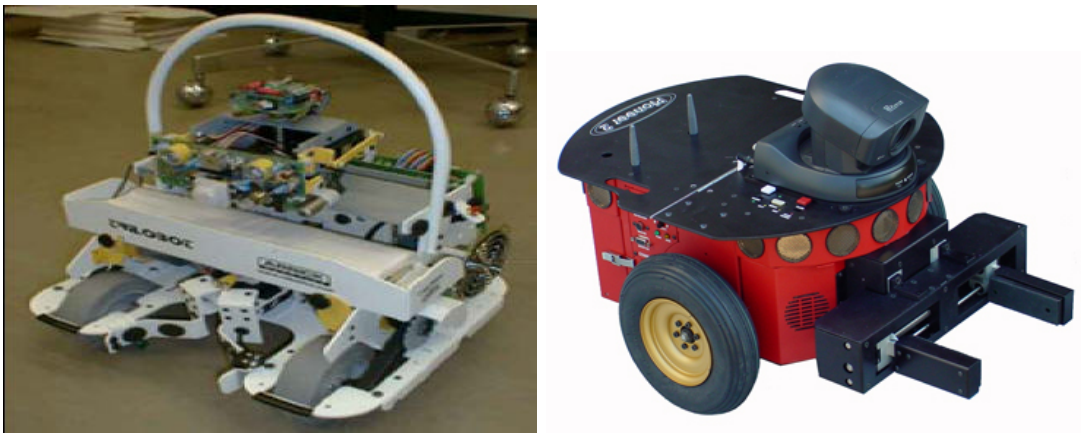

Figure 1.1: Two example mobile robotic platforms. Left: The Trilobot mobile platform. Right: The Pioneer DX mobile platform.

Figure 1.1). Actuators include equipment such as grippers used to pick up and put down objects in a physical environment and motors that are designed to move the robot throughout a physical environment. Sensors include equipment such as sonar, odometry and vision. While others employ a broader definition of the term "agent" that includes an existence purely in software [Russel and Norvig, 1995; Symeonidis et al., 2002; Bradshaw, 1997; Khoo et al., 2002], I am chiefly concerned with interesting problems requiring physical embodiment, and thus restrict the definition. While some of the ideas presented here may be useful for controlling software agents as well, that is not the focus of my investigation.

Teleoperation refers to a method of mobile robot control where an operator sends commands to an agent and the agent is responsible for executing those commands. In a purely teleoperated control system, commands are low-level instructions that the agent is expected to execute without the ability to question or interpret.

Autonomy refers to the extent to which agents are capable of making decisions and allowed to act on those decisions. The autonomy of an agent can be measured on a sliding scale [Anderson and Wurr, 2002]. Fully autonomous agents have no external operator and must accomplish tasks entirely on their own. Agents that have no autonomy, on the other hand, do not make decisions: they simply receive low-level commands from an external operator and execute them: they are purely teleoperated. Between these two points lies a blend of individual ability and following the commands of others. An agent may be able to execute some actions on its own (such as simple obstacle avoidance) but require commands from an outside agent to perform other tasks (e.g. be directed toward an exit). The level of abstraction in commands also grant an agent limited autonomy. The degree of teleoperation is related to the level of abstraction in commands recognized by the robot [Goldberg, 2000; Anderson and Wurr, 2002]. Commands containing a higher degree of abstraction, such as "find a pen" permit the robot some freedom as to how the command can be executed, since there are many series of actions that can result in finding a pen. Low levels of abstraction restrict the freedom of the robot (e.g. specifying a sequence of individual actions to find a pen, leaving no room for interpretation). The degree of teleoperation varies with the abilities of the agent. The more an agent can be trusted to perform adequately on its own, the less intervention on the part of an operator is required. Autonomy is also related to the number of participating agents and operators for a particular task. Participation in a group often requires agents to forfeit a degree of autonomy. In a multi-agent system, if agents do not have the necessary skills to

perform an entire task themselves, they must rely on other agents or operators to complete some of that task. When another agent completes a portion of a task, the other agents are forfeiting their autonomy in order to avoid having to embody an unreasonably large set of skills. Humans do this in the real world in any complex society. For example, I hire a mechanic if my car fails, I hire an electrician if the wiring in my house is faulty, and similarly employ other experts in our society who have the skills that I do not, since it is not reasonable for me to embody every single skill that is required to exist in a complex world. In such situations, agents can no longer decide to do whatever they would like: they must tailor some of their actions to suit the other agents, since they presumably embody skills those other agents do not have. Even when agents have the same skills, cooperation demands sacrifices for others, presumably in return for similar sacrifices on the part of others. There are many examples of autonomous agents in literature (e.g. [Baltes and Anderson, 2002; Michaud and Caron, 2002; Brumitt et al., 2002; Montemerlo et al., 2002]).

1.3 Urban Search and Rescue Domain

I have chosen to focus this work in a domain that is representative of many challenging issues in AI: *Urban Search and Rescue* (USAR). USAR is typically executed following a disaster that has occurred in a populated area, such as an earthquake, flood, fire, hurricane or terrorist attack [Casper, 2002; Casper and Murphy, 2002]. When disasters occur in populated areas, people often become trapped in collapsed

Figure 1.2: Examples of the USAR Domain [Jacoff et al., 2001b]

structures. Teams of rescue workers are dispatched to the collapsed structures tasked to locate and retrieve people who are trapped, referred to as *victims*.

Collapsed structures often contain a variety of *voids*. Voids are openings in the environment large enough for a rescue worker to move around in, either the remains of a room before the structure was compromised or an open area formed by the collapse of the structure. Since voids may contain human victims, all voids in a collapsed structure should be located and searched.

The USAR domain is complex and dynamic. The features of the USAR environment include debris scattered on the floor, broken mirrors and glass on the walls, poor lighting, and dark areas and surfaces (see Figure 1.2) [Murphy et al., 2000b]. The environment often contains elements dangerous to rescue workers such as toxic spills, extreme heat and unstable structures.

Robot assisted USAR involves deploying mobile robots in collapsed structures to locate victims. The danger posed by the USAR domain makes it ideal for the

application of mobile robot technology. The same factors that make the environment dangerous for rescue workers, however, also make it very difficult for the successful application of robotic technology. Most of the sensors that mobile robots and their operators rely on are much less effective in these domains.

Image processing is often hindered due to poor visibility. Reasons for poor visibility include a lack of lighting, due to electrical failure or the destruction of light sources. The presence of flames can cause intense flickering, distorting colors from darker to lighter shades and back. In addition, a coating of dust over objects leaves everything in the environment a similar color, making objects difficult to distinguish purely by color patterns [Casper, 2002; Murphy and Hershberger, 1996; Murphy et al., 2000b].

Sonar and laser range finders are affected by the materials that are typically found in collapsed structures. Highly reflective materials such as mirrors can distort lasers, causing noisy readings. Materials such as draperies can absorb sonar readings, making it difficult to tell how far objects are [Casper, 2002; Murphy and Hershberger, 1996; Murphy et al., 2000b]. In general, if mobile robots are not equipped with highly robust methods to deal with noisy data, they are likely to fail [Casper et al., 2000].

Robot mobility is also likely to be compromised by the unstable and often jagged debris that can cause a robot to become hung up and unable to move. The jagged debris may also return extremely noisy sonar and laser readings [Casper, 2002; Murphy and Hershberger, 1996; Murphy et al., 2000b].

While these factors make such a domain very difficult to the successful application

of robotic technology, the hazards associated with it make this environment typical of those to which we most want to apply mobile robotic technology. While the focus of much USAR work is on indoor domains, other such environments include open mines, landslides and other outdoor disasters, deep underwater environments, and outer space.

To promote research in robotic assisted USAR, the domain has been proposed as a challenge in recent mobile robot competitions [Casper et al., 2001]. These competitions have inspired the creation and assessment of deployable test beds for USAR [Jacoff et al., 2001a]. Figure 1.3 illustrates an example of such a test best developed by the *National Institute of Standards and Technology* (NIST) for USAR and used in the 2003 Robocup competition in Italy. This test bed mimics some of the important features of the USAR domain, including darkened chambers, curved walls, soft materials, hidden victims, ramps, various flooring material and debris. The test bed provides an extremely difficult domain for both autonomously and teleoperated robots. The NIST test bed was assessed by Murphy et al. [2000a], who found that although it was difficult, it still did not represent the degree of difficulty found in real world search and rescue scenarios.

1.4 Balancing Autonomy and Teleoperation

In a complex dynamic environment such as USAR, strong demands are placed on the abilities of an agent in comparison with the simple environments that characterize

Figure 1.3: The NIST standard test bed, Robocup 2003, Padua, Italy.

most laboratory research. When operating in such a domain, agents can fail at their tasks for a wide variety of reasons: misjudgment, sensor failures, sensor inaccuracies, unpredictable environmental issues, actuator failure or becoming immobile, etc. Artificial intelligence is currently far away from solving all the problems necessary for agents to participate fully autonomously in complex environments like those described above. The introduction of a human operator can help agents perform their tasks more effectively.

Given that a human operator is of use, why not leave the operator in control? First and foremost, the operators are limited as well: operators become fatigued, often after only a short time. In an emergency situation, agents must be able to operate continuously for long periods of time. Casper and Murphy [Casper, 2002; Casper and Murphy, 2002] describe the fatigue and stress that operators had to endure during their experiences working with robotic rescue at the World Trade Center. Most slept only three hours during the first three days, leading to many fatigue-

related errors in judgment. Such errors include missing important visual clues while navigating the robot (e.g. signs of potential victims, such as a watch on the ground) and poor navigation. Fatigue is of special importance when certain tasks are simple but repeated continually. Automating those tasks would offer a great reduction in the cognitive load on the operator.

How much information an operator can handle at any given moment is another issue. When an operator must process too much information at one time, he or she may suffer from cognitive overload [Arkin and Ali, 1994]. Cognitive overload refers to the limit of information processing an individual can handle efficiently. Automating some of the task can alleviate the amount of information the operator must handle: if an agent can perform 80% of the required task autonomously, the operator will only have to perform 20%, reducing the cognitive load of the operator significantly. Automating the task also allows an operator to teleoperate multiple agents. If there were no autonomy in the agents, then the operational ratio of each operator would likely be 1:1. If agents can perform the majority of their task autonomously, then the operator can focus his or her attention on the agents that require assistance.

Finally, we must consider how well an operator can accomplish certain tasks. Some tasks are performed more efficiently autonomously, even with today's AI technology. Arkin and Ali [1994] demonstrate the efficiency of autonomy given certain tasks in a study of the efficacy of teleoperation vs. autonomy in foraging, grazing (covering an area completely, as would occur when grazing or mowing grass) and herding (activities in teams of robots). They found that although teleoperation was advantageous, it

was only advantageous under specific conditions. For a significant proportion of the experiments, agents were more efficient performing autonomously. They account for the improved autonomous performance by the operator's frequent inability to reconstruct a representation of the physical world from the direct perception of the robot as accurately or efficiently as an autonomous agent. This human difficulty of situational awareness arises whenever a human controls a remote object, especially in three dimensional space. At the World Trade Center rescue operation, the operators would often not use all the available information channels, concentrating only on the CCD camera. Concentrating on the CCD camera led to the agents becoming stuck and the operators not being able to understand how they got stuck. In a specific example one agent got hung up on a metal rod not visible to the operator. Frustrated, the operator had to end the mission and retrieve the robot only to find out at that time how the agent was stuck [Casper, 2002]. Relying too heavily on certain perceptions at the expense of others is analogous to trying to park a car while wearing blinders. The lack of certain perceptions or the over-dependence on particular sensory information can hinder one's ability to perform the task as efficiently as required.

Since agents are unable to perform adequately in such unpredictable and dynamic environments, and human operators have similar difficulties, mobile robots must be designed to balance the *degree* of autonomous performance with the instructions and advice provided by a human controller. In the state of the art today, operator-supplied instructions are normally very low-level instructions (direct machine instructions for the robot or a level only slightly removed from this), and providing these instructions

to a robot that is otherwise autonomous is termed a *teleautonomous* relationship.

While allowing an agent some autonomous abilities can greatly reduce the problems introduced by human operators, keeping a human in the loop also simplifies building the agents themselves. Fully autonomous agents can often find themselves in situations that may not occur frequently enough to warrant the programming effort involved to equip the agent to handle these situations. While being difficult to automate, many of these problems are not difficult for humans to solve in theory, although in practice these problems can be difficult for humans to handle as well. Consider a robot assigned the task of searching through debris in a collapsed building. Suppose while working, the robot becomes stuck in a pile of debris. The robot may not be equipped with the cognitive abilities to recognize how it is stuck other than that specific wheels are not moving, and it may lack the ability to create a plan sophisticated enough to free itself in such an unstructured domain. Despite the difficulty of solving this problem from the robot's perspective, a human operator may be able to recognize the problem and come up with some strategy to free the agent. Another agent observing the stuck robot may also be able to provide a better sensory perspective and assist in solving the problem. The above example, though simple, conveys the need for methods to aid agents when they are limited in their abilities and cannot solve a problem because it was either unpredictable or beyond the capabilities of an autonomous agent.

Combining the best elements of both autonomous control and teleautonomy shows promise. Ali [1999] points out an important additional advantage not yet mentioned

here: teleautonomous agents can be more versatile than autonomous agents, since by receiving human assistance to deal with novelties they have not seen, robots can attempt to deal with environments that were not considered in their original programming.

The goal of this research is to design and build a control architecture that allows a flexible balance between teleoperation and autonomous performance in a complex, dynamic domain: that of USAR. Robotic agents operating under the proposed control architecture will attempt to recognize when they are in a situation they cannot handle, and signal the operator and/or other agents of this condition. At that point the operator can take control of such an agent and help it to solve its dilemma, and other agents may be able to offer their perceptions. The major design goal for this approach is to reduce the amount of work necessary for an operator to control multiple robots.

1.5 Research Questions

Now that the terminology has been introduced and the domain has been described, the basic research questions that this thesis will address can be presented.

Research Question 1

Will the addition of teleoperation to autonomous agents increase their overall performance?

If autonomous agents are designed to perform some task, we can observe those agents performing that task and attempt to improve their performance by introducing

external instruction in the form of teleoperation. By observing the performance of the agents working autonomously and then observing the agents working with a blend of autonomy and teleoperation, we can evaluate and compare their performance and conclude whether performance is increased by the introduction of teleoperation.

Research Question 2

Can the introduction of autonomy reduce the number of interactions required between the agent and the operator while maintaining a comparable overall performance?

We can use the number of interactions that an operator has with an agent to estimate the amount of cognitive load that the operator is enduring. High levels of interaction are correlated with high levels of cognitive load. If we introduce autonomy into previously teleoperated agents then a reduction in interactions between the operator and the agent indicates a reduction in cognitive load. If the same overall performance can be achieved by agents performing the task while maintaining a lower number of operator-agent interactions, then blending autonomy and teleoperation is beneficial to the operator.

Subsidiary Question

Can an infrastructure be designed to support a practical balance between autonomy and teleoperation in complex dynamic environments?

In answering the first two research questions the subsidiary question is also addressed. In this thesis, such infrastructure is designed and demonstrated. This infrastructure is discussed in Chapters 3 and 4.

1.6 Addressing Thesis Research Questions

A domain-independent solution to the problem of balancing autonomy and teleoperation is not only beyond the scope of a thesis, but well beyond the current state of the art of artificial intelligence. The breadth of knowledge required to handle every possible situation in every possible problem environment would be equivalent to human-level intelligence. Within USAR, there are clearly many problems to be solved. I have focused the efforts of this work on teleautonomous agents on a specific problem identified by the research community as extremely important to the USAR domain. Casper et al. [2000] indicate two problems whose solution would be of particular practical interest: teleautonomous performance of topological searches and teleautonomous stairwell searches. In USAR, robots are required to search various sized voids and create a map so that rescue workers can find victims trapped within them. Searching various sized openings and creating maps is commonly termed *topological search*. Stairwell searches are similar, with the added difficulty (in terms of mobility of a robot) of having to search up stairwells. These tasks are emphasized by Casper [2002] because they are frequently executed by USAR workers but are too difficult to perform autonomously and so still require an operator.

Of the two, this thesis focuses on topological search in USAR. The domain is also an important choice: these are dynamic and hazardous environments where sensors are not only prone to error, but where errors must be expected. As such, results in this domain should be transferable to simpler domains without this degree of

error expectation. Moreover, this domain can be handled using equipment currently available to me, rather than the expensive equipment necessary to traverse a stairwell.

1.7 Summary

In this chapter I introduced terminology that will be used throughout the rest of this work. The USAR domain was described in detail and I discussed the importance of striking a balance between autonomy and teleoperation.

The next chapter will discuss work related to balancing autonomy and teleoperation. Chapter 3 describes the control system that was designed for this work while Chapter 4 describes how it was implemented. Chapter 5 describes experiments performed to evaluate the control system described in Chapter 3 and discusses their results. Finally, Chapter 6 provides an analysis of the results and the conclusions that can be drawn from this research, along with some indication of useful future work in this area.

Chapter 2

Literature Review

2.1 Overview

This research contains elements from the fields of autonomous agent development, control theory, mobile robotics, and distributed artificial intelligence. Each of the fields named previously has been the topic of research for many years and all have been well studied. The goal of this chapter is to introduce important research in areas related to this thesis and to differentiate this research from similar work. Included in this literature review will be a brief history on mobile robot control and autonomous agents, followed by a look at work related to my research area.

2.2 Agents

All agents respond in some fashion to the world in which they are situated. The goal of agent design is to have the agent respond to its environment in a meaningful way. The agent must have some way of perceiving elements of the environment and interacting with those elements in a purposeful manner. How agents should respond to their environment is a widely studied problem. From researching this problem, many approaches to agent design have been developed. Most agent design approaches can be grouped into on of three categories: planning agents, reactive agents and hybrid agents.

2.2.1 Planning Agents

The discipline of artificial intelligence has focused for many years attempting to build intelligence using logic to reason over a symbolic representation of the environment [Brooks, 1991b], an internal world model. Central to many classical planning systems is the symbol system hypothesis [Newell and Simon, 1976], which states that intelligence operates on a system of symbols, where the symbols represent entities in the environment. Agents typically execute a sequence of three phases; sense, plan and act. In the first phase, sense, the agent uses its sensors to gather data about the world around it. The data is used to update the agent's internal world model, so that changes in the environment are reflected in the symbolic representation maintained by the agent. Following the sensing phase, the agent plans a course of action.

The agent analyzes the new internal world model and using some planning algorithm decides what sequence of actions will result in the agent achieving its current goal. Once a sequence of actions has been chosen, the execution phase commences. The execution phase takes as input the sequence of actions that the agent would like to execute and attempts to perform them [Arkin, 1998].

Symbolic reasoning has the advantage of enabling the agent to reason about its past and form elaborate plans for its future. Assuming the sense, plan and act cycle repeats frequently enough, the agent can adjust the plan as things in the environment change. An example of a planning agent is described in [Nilsson, 1984].

Several disadvantages for planning agents have been identified. One major disadvantage of planning agents lies in the difficulty of maintaining accurate internal world models [Brooks, 1991a]. The accuracy of a world model is affected by both the frequency that the world model is updated and the amount of noise in the sensor readings. Because the environments for which agents are designed are often very complex, very elaborate world models are required, and maintaining consistency between such complex world models and the real world is a daunting task [Agre and Chapman, 1991; Brooks, 1990; Chapman, 1989]. If the world model is not updated frequently enough, the world model will fall out of date or out of sync with the real world [Brooks, 1990]. If the planning algorithm is applied to a world model that is out of date, the plan produced by the planning algorithm may not be applicable to the real world, since the real world may have changed since the last time the wold model was updated [Brooks, 1990; Mataric, 1997]. Noisy sensory readings also

contribute to inaccurate world models that do not represent the environment close enough to be useful as a basis for planning. Since sensor reading are taken every cycle, and the world model is incrementally built from those readings, errors in the world model are cumulative, leading to a world model that becomes more inaccurate over time [Mataric, 1997; Agre and Chapman, 1991].

Another disadvantage is related to the reliance of many planning agents on the symbol system hypothesis. If the symbols representing entities in the world model are not sufficiently grounded to physical objects in the real world, they become meaningless [Coradeschi and Saffiotti, 2000; Harnad, 1990]. Additionally, the symbols are task dependent: different tasks require specific representations, making a planning agent only capable of solving the problem it was designed for [Brooks, 1990].

Planning is also very time consuming. Since the world being modelled is complex, the planning system has to be able to predict an exponential number of outcomes. This requires searching through a potentially exponential search space. Heuristics are useful for trimming the search space, but heuristics often sacrifice accuracy [Chapman, 1989]. In any case, planning takes a significant amount of time. Agents in most real world environments do not have significant time to ponder their actions, since rapid changes in the environment quickly make plans obsolete. Taking too much time to plan also highlights another disadvantage: obsolete plans require a rerunning of the planning algorithm to produce another plan, which may also be obsolete by the time it is ready to execute. Once a plan is prepared to execute the agent will attempt to complete plan execution until either the plan will obviously fail or a better course

of action arises. This requires a level of planning above the planning algorithm to determine when a plan is no longer sensible [Agre and Chapman, 1991]. The plan-replan cycle can lead to agents spending all their time planning, and never executing any actions in the environment. The sense, plan and act cycle in very complex systems inevitably ends up taking too long to allow the agent to interact with a sophisticated world in real-time.

2.2.2 Reactive Agents

Reactive agents address some of the downfalls of planning agents. Reactive agents react to the environment without searching or requiring world models [Mataric, 1997]. In general, reactive systems are based on stimulus-response relationships. Stimuli in the environment trigger an immediate response from the agent's actuators. Noisy sensor readings have little effect on reactive systems, since there is no world model and therefore errors in readings are not cumulative. Planning is also eliminated, since responses occur immediately in response to the stimuli requiring very little time [Mataric, 1997; Brooks, 1990]. Eliminating planning makes interactions between reactive agents and their environment much faster than planning agents. Reactive agents are therefore more effective at interacting with the environment in real time. Reactive systems are based on the assumption that the world is its own best model, since it is always up to date and always contains every detail there is to know [Brooks, 1990]. Reactive agents also address the difficulties with the symbol system hypothesis

using the *physical grounding hypothesis*. The physical grounding hypothesis is based on having a system composed of modules, where each module produces behavior and the combination of those modules produce more complex emergent behavior [Brooks, 1990; Mataric, 1997; Arkin, 1998]. Purely reactive agents are robust. Since every module of the reactive system produces responses based on stimuli, if stimuli does not occur in the environment, the associated responses are simply not elicited, having little effect on the emergent behavior of the agent.

The concept of potential fields underlines many reactive approaches. Potential fields use a mathematical function to transform sensor data from the robot into actions to be performed [Arkin, 1998]. A continuous mathematical function provides output for all stimuli across the environment, so that no matter where the robot is located it gets some instruction from the potential field. Many potential field controllers use a function based on the law of universal gravitation: as the robot gets further from an object, the force exercised by the object on the robot is reduced. The reduction in force approximates the square of the distance between the robot and the object [Krogh, 1984; Khatib, 1985]. Obstacles exercise a repulsive force, while goals exercise an attractive force. Every object produces its own potential field. Fields are combined into a global field using superposition [Arkin, 1998], which allows a number of individual potential fields to be combined into a global potential field (2.1. Following the gradient of the global field produces a smooth trajectory that a robot can follow to navigate the environment. Many reactive systems are based on the idea of potential fields.

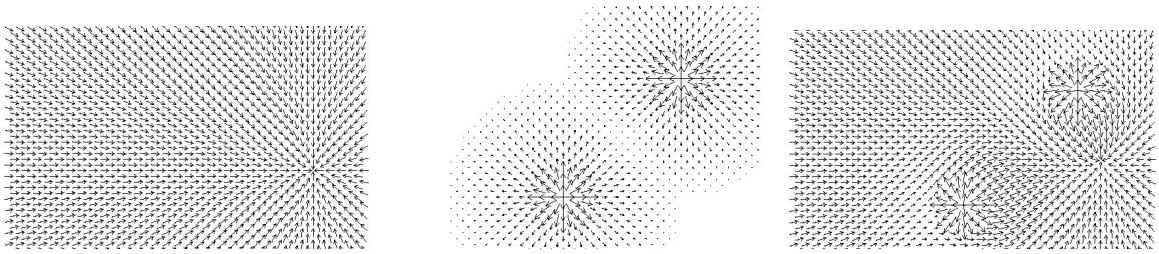

Figure 2.1: An example of potential fields being combined [Balch, 1998b]. Left: the potential field of a goal. Middle: the potential field of two obstacles. Right: the global potential field.

2.2.2.1 Subsumption Architecture

Central to many reactive approaches is the notion of a *behavior*. Arkin [1998] defines a behavior as a stimulus/response pair for a given environmental setting that is modulated by attention and determined by intention. Reactive approaches that are behavior-based take the concept of a behavior and add structuring to it. Behaviors are organized into larger packages that interact with each other in some way, and often some form of layering is added to mediate between competing behaviors and packages of behaviors

One of the earliest behavior-based approaches was the subsumption architecture, proposed by Brooks [1986] in response to the limitations of planning agents. Like many reactive systems, subsumption discourages the use of internal world models, allowing only very small pieces of state information to be stored in the behavior

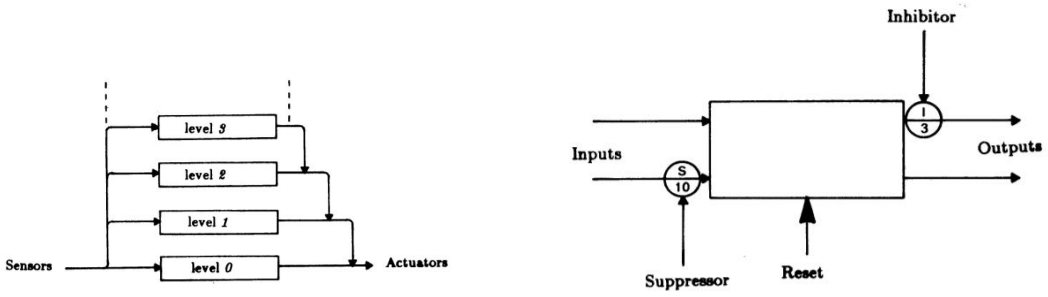

Figure 2.2: An example of Brooks' layered subsumption architecture [Brooks, 1986]. Higher level layers subsume lower level layers to control the system.

modules. The subsumption architecture splits the task to be preformed into layers (see Figure 2.2) [Brooks, 1986; Mataric, 1997; Arkin, 1998]. Each layer represents a level of competence for the agent. Layers are built on top of each other: higher layers can observe and inject data into lower layers, but lower layers operate without knowing anything about the layers above them [Brooks, 1986]. Scalability is achieved by adding layers onto an already working system. Since lower layers do not need to know anything about higher layers, the higher layers can be added without modifying the lower layers. The higher layers can be viewed as subsuming the lower, hence the name.

Each layer is composed of modules (see Figure 2.2). Each module is a finite state machine (with some additional variables) that takes in one or more inputs and produces one or more outputs. The modules are the components that couple perceptions to actuators [Brooks, 1991a]. Perceptions are fed into modules, which

immediately produce some output. The output of modules are fed into other modules and finally directly to the robot's actuators. As with many reactive systems, there is no planning: behavior emerges from the interactions of many modules acting in parallel [Brooks, 1986]. Noisy data entering one module, or even the failure of one module, should in theory have little effect on the emergent behavior, and since there is no internal world model, errors are not cumulative [Brooks, 1991b]. Many robots have been implemented using the subsumption architecture (e.g. [Horswill, 1993; Mataric, 1992; Brooks, 1986, 1989, 1990; Brooks et al., 1999]).

A disadvantage to the purely reactive agents proposed by Brooks is the difficulty of representing complex behaviors without any internal representation and very little memory. Schematically, even simple behaviors such as foraging require complex subsumption diagrams. Since the complexity of the subsumption diagram grows rapidly as the emergent behaviors require more complexity, the scalability of the subsumption architecture is questionable. Another disadvantage of a subsumption architecture is the likelihood of the system becoming trapped in a *local minima*. A *local minima* is a position where every action that the agent executes will lead it back to the same position. Since the responses to a sequence of events can potentially repeat endlessly, the agent can become trapped by continuing to repeat the same actions over and over again and making no progress. Getting trapped in local minima highlights the need for agents that are able to use past experience to modify future behavior.

2.2.2.2 Schema-based Agents

Schema-based agents emerged as an approach to robot control that realizes the advantages of purely reactive agents while loosening some of the constraints proposed by Brooks [Arkin, 1998]. Schema-based agents retain the reaction speed achieved by purely reactive agents by discouraging complex world models and instead containing multiple simple world models that are efficient to maintain. This addresses many of the disadvantage of purely reactive agents. Like many approaches to robot control, schema-based agents support the notion that there is a lot more to be gained by increasing the functionality of reactive approaches before adding planning facilities.

Schema-based agents are based on schema theory. Briefly, schemas are a mapping of perceptions to actions associated with the perceptions. They contain the information necessary to encode agent behavior by means of the sensory data required to illicit an action, where the computational process for how the action is performed is embedded in the schema [Arkin, 1998].

Schema-based systems are another implementation of the behavior-based model. The schema-based approach bears some similarity to subsumption based systems, in that schema-based systems are composed of several modular behaviors organized into a finite state machine. Each behavior takes sensory information as an input and produces the desired action to be performed. The output of several behaviors can be combined to produce overall actions, and active behaviors can be turned on or off according to the state of the finite state machine governing the control system. The

major difference between schema-based systems and subsumption based systems is how the behavior modules are used. Each behavior modules produces the same format of output, a vector describing a suggested trajectory. Unlike the layered approach of subsumption, outputs of the behavior modules are combined to produce the output of the system, allowing each behavior to contribute in varying degrees to a robot's overall response. No predefined hierarchy exists for coordination: instead modules cooperate using vector addition.

In a schema-based approach, behaviors are implemented through a combination of two types of schemas: *Perceptual schemas* and *Motor schemas*. Perceptual schemas serve to filter perceptions and allow the robot to discern elements of interest. Perceptual schemas connect to motor schemas, which embody desires for the robot's actuators to perform some action based strictly on the perceptions obtained by the robot's sensors. Perceptual schemas may be connected so that some motor schemas stimulate others in turn, allowing complex emergent behavior to occur [Arkin, 1998].

The schema-based approach is similar in spirit to the potential fields method described earlier. The benefit of motor schemas over potential fields lies in only requiring the control system to calculate a small portion of the global field. Only those objects eliciting responses from the schemas are used in the global field calculation, centering around the robot and only extending to the immediate course of action [Arkin, 1998]. Potential fields require the whole global field to be calculated, which is computationally expensive.

A system implementing motor schemas combines the motor schemas producing an

action vector, which is a vector containing a magnitude and orientation representing an action that the robot can execute. The robot interprets the action vector and executes the resulting motor commands on the actuators. Motor schemas are combined in the following fashion [Arkin, 1998]:

1. Perceptual Schemas are updated on every perceptual cycle.
2. Motor Schemas produce a vector representing the action desired by the motor schema depending on its associated perceptual schemas.
3. The output of all motor schemas is weighted by gain values. The gain values distinguish the importance of different motor schemas.
4. All resulting vectors are added together using vector addition.
5. The vector resulting from the sum of all motor schema vectors is interpreted by the robot.
6. The robot sends motor commands to the actuators, which execute the action.

The magnitude of a vector indicates the urgency with which the motor schema wishes the action to be executed.

Many schema-based robots have been implemented (e.g. [Arkin, 1989, 1992; Cameron et al., 1993; Balch and Arkin, 1995]. While schema-based agents are meant to address the disadvantages of purely reactive systems, some disadvantages still remain. Schema-based agents are not proactive. They lack the ability to reason about the future or the past having little to no representation of the world. In addition, schemas

are mapped onto the hardware for which they are designed, making it difficult to move systems from one platform to another [Arkin, 1998].

2.2.3 Hybrid Approaches

There are a number of control architectures that attempt to combine the advantages of both symbolic reasoning and reactive control [Graves and Volz, 1995; Arkin and Balch, 1997; Gat, 1992; Lyons and Hendriks, 1995; Lee et al., 1994]. The majority of these systems use a reactive control system as their base. The agent reacts to the environment as the environment changes and in general is designed to handle the agent's short term, underlying behavior. Often the reactive component is responsible for keeping the agent safe when entities in the environment could pose a risk to the agent, and when reaction time is important, such as the potential for colliding with a moving object. Above the reactive subsystem, there is generally a higher level planner. The planner is responsible for planning ahead and directing the high level goals of the robot. Ideally the planner can take as much time as it needs to maintain a world model and plan using that world model, since while planning, the agent is continuing to function using the reactive subsystem. The agent does not have to wait until the planner is ready to send actions before the agent can perform productive work.

Knowledge used to represent internal world models for the planning component of the hybrid system can make the behavioral configurations of the underlying reactive

system more versatile by adding knowledge about the task and the environment. Additionally the dynamic nature of the internal world model may provide insight into ways of achieving goals that the reactive system is not able to, since it has no representation to work with Mataric [1997].

The planning component of a hybrid system still shares the disadvantages of planning agents described in Section 2.2.1. If the environment is not stable or consistent from one time step to the next, the planner may still develop useless plans [Arkin, 1998]. Continually replanning will result in a slowdown of the overall system, especially if the plans are not relevant. The issue of physical symbol grounding is still present, even with a reactive subsystem: if the symbols used by the planner are not well grounded they will be of little use [Brooks, 1990]. Finally, the world model may still be inaccurate, adding to poor plans [Mataric, 1997].

There are, however, some important concepts captured by one particular hybrid architecture important to my research. Arkin and Balch [1997] developed the *Autonomous Robot Architecture* (AuRA). AuRA blends traditional symbolic reasoning with behavior-based reactive processing. The behavior-based reactive processing is achieved using motor schema-based control. The ability to do such blending is important in many problem areas, since agents have to react to new situations, yet be able to reason about the situation they are in and the consequences of actions in that situation. AuRA is designed for controlling autonomous agents. The control architecture designed in this work (described in Chapter 3) is similar in spirit to AuRA in that a symbolic reasoning component will work along with an underlying behavior-based

system. The main difference between my work and the work of Arkin and Balch is that I concentrate on blending autonomy and teleoperation in my architecture, which is beyond the focus and abilities of the AuRA architecture. While AuRA takes a high level approach, where the symbolic component specifies goals to the behavior-based component, my work focuses on the more complex relationship between the human operator and the agent that is required in many domains. The operator requires varying levels of detailed control over agents in my architecture, which is currently not a supported feature of AuRA. As such, some of the structural ideas from AuRA will be replicated in my architecture, but modifications and additions will be required. The additions and modifications contribute to furthering the complexity of tasks that an agent is able to accomplish by making the human operator and the agent cooperate seamlessly and effectively.

The development of autonomous agents is a widely studied field, and while autonomous agents for very complex environments are a long way away, autonomous agents for simple structured tasks exist today (e.g. [Arkin, 1998; Arkin et al., 1999; Huang and Krotkov, 1997; Kortenkamp and Weymouth, 1994]).

2.3 Teleautonomy

Direct remote control is well understood and common in today's world: it is observable in everything from remote controlled television sets to toy cars. Compared to this and the state of autonomous approaches (as described in Chapter 1), the state of

teleautonomous approaches is far behind, and the application of teleautonomy to complex environments is extremely limited. This section reviews the major work in teleautonomous control to date.

Arkin and Ali [1994] describe one of the primary motivations requiring a balance between autonomy and teleoperation in robotics: cognitive overload, which has already been mentioned in Section 1.4. Arkin and Ali [1994] propose two approaches for teleautonomy with respect to multiple agents. Both of these are schema-based approaches, where behaviors (wander, avoid, etc.) are encoded in the form of motor schemas, which interact at run time to produce output to robot effectors. The first approach has the operator's control as a behavior that influences the robots' effectors just as any internal behavior does. All active behaviors contribute a vector representing some desire that the robot has to perform each behavior. The operator introduces his or her intentions through an additional schema contributing to the emergent behavior in the exact same way as all other behaviors [Arkin and Ali, 1994]. Experiments tested the operator-as-a-schema approach by having the operator input a magnitude and orientation representing the intentions of the operator. A group of robots accepted the low level instruction from the operator and summed it with the rest of their active schemas. The whole group of robots was therefore influenced by the single command of the operator. The second approach for teleautonomy involved having the operator act as a supervisor. The operator had access to the behavioral parameters of the society, where the behavioral parameters included abstract components, such as aggressiveness and wanderlust, down to the low level gains of each

motor schema. The operator could effect the emergent behavior of the society of agents as a whole by adjusting their behavioral parameters. Arkin and Ali [1994] tested both of their approaches in three tasks: foraging, grazing and herding. They found that in the foraging task the total number of steps required to complete the task by the robot could be reduced significantly if the operator guided the robots to areas where more attractors existed. However, once an attractor was in sight, they found that human intervention would hinder more than help the robots. In the grazing task, robots performed best when very little amounts of teleautonomy were introduced. The operator was only helpful when agents could no longer find ungrazed areas. Once ungrazed areas were located, the agents performed better autonomously. For herding, operator intervention was found to be very helpful. The operator could herd a group of agents into a confined area without much difficulty, and robots rarely left the confined area [Arkin and Ali, 1994]. Arkin and Ali [1994] found that only small amounts of teleoperation were required to increase the performance of a society of agents performing simple tasks: too much teleoperation would often hinder performance. Influencing a group of agents reduces the amount of attention paid to each individual agent, thus reducing the cognitive load of the operator. Their approach was tested only in very structured and simple problem domains. My thesis will deal with teleautonomy in domains with much less structure than these. It will also avoid the main problem with Arkin and Ali's approach to dealing with cognitive overload - only limited results can be obtained by giving mass direction to every agent in a team environment. Their approach is equivalent to telling every single agent on a field to

"turn left" to get one agent to its target. By doing this, Arkin and Ali doom a large number of agents to receiving commands that will not likely be helpful, in exchange for the expectation that this cost is overridden by the small direction obtained by those that needed it. Their own results showed that the expectation that the cost is overridden by the small direction obtained by those that needed it was not a rational expectation for many domains (though their approach did work in domains where the misdirection was not a significant problem).

Crandall and Goodrich [2001] present the notion of *neglect* in remotely controlled agents. They describe neglect as the amount of time during which the agent is not receiving some sort of instruction. They show that this down time can hinder the performance of the robot, and can be due to the operator turning his or her attention away from the agent, or from delays between issuing commands and the agent receiving those commands. The important thing to note from the standpoint of this thesis is that while the agent is being neglected it could make use of this time by performing tasks autonomously to increase the effectiveness of the robot. Crandall and Goodrich [2001] describe a robot control system consisting of a set of robot behaviors and a user interface for controlling the agents. Their systems uses five levels of autonomy: fully autonomous, goal-biased autonomy, waypoints and heuristics, intelligent teleoperation and dormant. The fully autonomous mode is based on a utilitarian-voting scheme that allows the agent to initiate interactions with the environment. Goal-biased autonomy allows the operator to mark goals on a 2D map and have the agent attracted to that location, where the attraction is treated

as another behavior. Waypoints and heuristics involves having the operator add icons to the 2D map that heuristically influence the robot's actions using a potential field based approach. Intelligent teleoperation uses a joystick to operate the robot, where the robot could assess the risk of the human input and decide whether to perform the instruction or not. The goal of the above control system is to reduce the amount of time the agents spends without some sort of operator direction, and having the agent be able to initiate interactions with the environment in the absence of operator direction. Crandall and Goodrich [2001] emphasize the directly controlled portion of the agents in this work, while realizing the autonomous portion of the agent to only a very limited degree. Although Crandall and Goodrich [2001] discuss degrees of autonomy they do not describe an implementation in their work to show that any balancing has been implemented even to the degree of some of the prior work described above.

Casper and Murphy [Casper, 2002; Casper and Murphy, 2002] offer a tremendous amount of insight into robot-human interaction in USAR. Many of the issues they discuss are important to consider for this thesis. First, the need for robot involvement in USAR can be justified by the number of human and canine rescuers who are injured in rescue activity: for example, 135 rescuers where killed in the 1985 Mexico City Earthquake alone. Their work also provides insight into operator fatigue, illustrating the need for more autonomy. While at the World Trade Center, Casper and Murphy observed that lack of sleep on the part of operators introduced cognitive fatigue that significantly reduced operator performance, leading to obvious mistakes.

For examples of these see [Casper, 2002]. Murphy and Casper also observed the need for better human-to-robot ratios, explaining that a 2:1 human-robot ratio was not an efficient use of personnel. Finally, the need to automate certain tasks was apparent. Repetitive simple tasks such as stairwell searching and room searching can be tremendously tiring for the operator when the operator must also scrutinize a visual image for victims and recognize when structure stability is an issue [Casper, 2002; Casper et al., 2000]. Casper and Murphy describe valid reasons for why increasing the amount of automation in USAR robots would be beneficial.

Cao et al. [1995] describe work on a remote robotics laboratory where ten small UCLA/ISR R3+ robots are controlled through the Internet via Unix workstations. The laboratory is considered a resource shared among experimenters across the country. Experimenters submit experiments (programs) or real time (low-level) instructions across the Internet, and then observe their experiments being carried out. The execution of experiments in the remote robotics laboratory requires teleautonomous control of the robots to various degrees, either by complete moment-by-moment control (instruction by instruction) or by writing a program to be executed by the machines. Cao et al. [1995] plan also to introduce some additional autonomy in the robots in the form of simple obstacle avoidance algorithms that will limit the amount of potential damage an operator can cause to robots through execution of their teleautonomous experiments. While Cao et al. [1995] demonstrate why the balance between teleoperation and autonomy is important, my proposed research involves demonstrating these issues both in a more demanding and practical domain, and in a more flexible

manner.

Trivedi et al. [2000] designed a system that would allow robotic units to recognize traffic collisions and other accidents. This system is strictly a laboratory design and years away from being deployable, but makes use of teleautonomous robotic agents that can form a perimeter around a collision. These agents specialize in forming a perimeter, and the remote operation provides very basic instructions to guide the robots to form perimeters around specific areas. This application of teleautonomy demonstrates the potential to have equipment constantly monitoring an area without the full attention of an operator, but is once again extremely simplistic: the agents have one purpose, and can achieve that fairly simply through a polygon forming algorithm where each agent takes the role of a point on the polygon. The operator supplies only location guidelines for the polygon forming activity, and the balance between autonomous ability and remote control has been fixed as well - human operators guide the placement of the perimeter, and the agents form the group using the operators guidance. Finally, the environment is also much more structured (flat pavement) than can be assumed in USAR.

Arkin and Balch [1998] and Bentivegna et al. [1997] demonstrate a teleautonomous hummer designed to perform scouting missions for military operations. The hummer is equipped with an onboard computer that controls the steering, throttle and brake. An operator can control the hummer using an onscreen joystick, which is part of the Mission Lab application [Bentivegna et al., 1997]. The Mission Lab application also allows the hummers to act autonomously by sending the hummers mission goals and

having the hummers execute them autonomously [Arkin and Balch, 1998]. Although the domain is less structured and more complex than the domains explored previously by Arkin and Ali [1994], the teleautonomy is still realized through a relatively simple additional motor schema in the behavior-based system. The missions themselves are also simple waypoint missions in large open areas, and lack the potential errors and unpredictability that the USAR domain provides.

Murphy and Sprouse [1996] describe a strategy for mixing robot and human control in the USAR domain by assigning a different search task to the operator than to an autonomous robot. The robot would perform a systematic search of an area, covering the entire area by splitting the area into sections and applying its senses to each section. The operator then performed the semantic search; in this case the operator directed the robot to semantically similar areas of interest. Casper et al. [2000] describe a paradigm for automating victim detection by robotic agents, while the operators controlled the navigational system. Casper et al. [2000] implement their strategy on a three-agent society architecture, where the robot, human and an Intelligent Assistant Agent together composed the society. The Intelligent Assistant Agent handled victim identification by taking the sensory readings from the robot, checking them for victims and presenting the results to the operator. In both of these cases, the boundary between autonomous processing and operator control is cleanly divided by task and is fixed - there is no concept of an agent having a degree of responsibility for the entire task, which my research will include.

2.4 User Interfaces

While user interfaces are not the focus of this work, an interface is obviously an important part of any computer system expected to interact with humans. There has been little work pertinent to user interfaces and robot control specifically, but volumes of work have been published on user interfaces in general.

I am attempting to design and evaluate an ergonomic user interface, and there is one important point in the user interface literature that is imperative to follow. The user must always be in control [Krogseter et al., 1994; Krogseter and Thomas, 1994; Strachan et al., 2000]. I will be designing the user interface to place the ultimate control in the hands of the operator. The operator is able to make decisions on how the system works and can reduce the agent's autonomy at any time to a point where the agent is entirely controlled by the operator.

2.5 Summary

In this chapter I described three primary types of mobile agent control approaches: planning agents, reactive agents and hybrid agents. I also introduced a variety of work in teleautonomy that is related to this research. Finally, I discussed some of the important issues concerning user interfaces. The next chapter is concerned with the design of control system that blends autonomy and teleoperation.

Chapter 3

A Blending Approach to Mobile Robot Control

3.1 Overview

The design of the control system implemented as part of this thesis is described in four sections. First, I discuss how I designed the autonomous control system, which is the underlying method of robotic control. Secondly, I describe how teleoperation is achieved in the system. Thirdly, I discuss how the output of the autonomous control system is blended with the commands of the human operator. Finally I describe the design of a user interface that allows the blending to take place. Throughout the rest of this work I will refer to the system that was designed and implemented as part of this research as the *blending control system.*

3.2 Autonomous Control

The autonomous control system was designed to be the base control system of the agents in this research; as such it functions independently of the other portions of the system. Its functionality is sufficient to enable the robot to explore its surroundings in a basic fashion, seeking out objects of interest and avoiding obstacles, which are the primary autonomous abilities necessary to the USAR domain. The goal of the autonomous control system was to be as parsimonious as possible in providing these abilities.

I chose a behavior-based approach to develop a simple control system that does not use symbolic reasoning and requires no world model. The autonomous control system uses a schema-based approach [Arkin, 1998] that operates as described in Section 2.2.2.2. A schema-based approach was chosen because it is a behavioral approach that couples perception and action very closely and requires no single global representation of the domain, making it an ideal choice for developing a simple but sufficient autonomous control system that is able to react to the environment in real time.

The particular schemas written as a basis for this research are those that provide a bare-bones autonomous agent. The schemas enable the robot to operate successfully in the USAR domain by enabling the robot to wander, avoid obstacles, move to a desired unmapped point, and perform the very basics of looking for victims. First, the agent is able to explore the area by moving toward unexplored portions of the

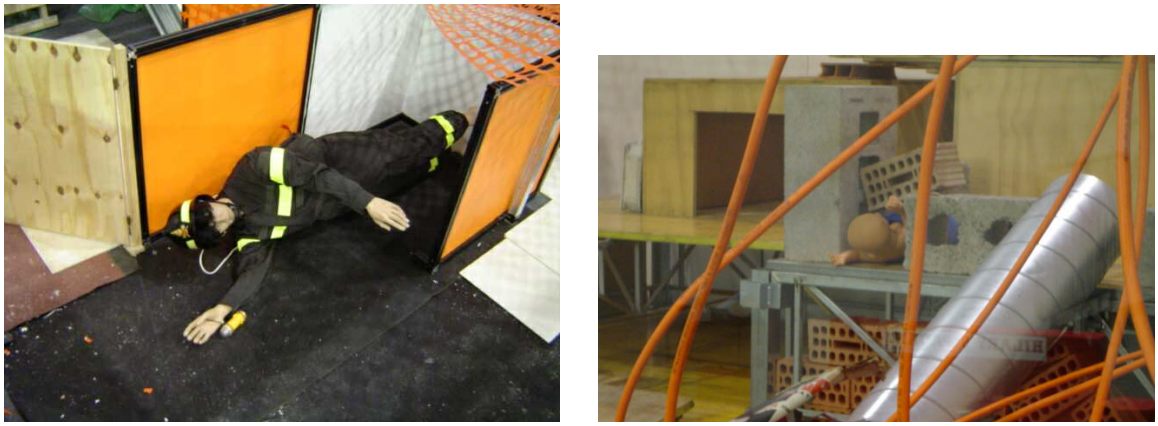

Figure 3.1: Examples of victims in the NIST USAR standard test bed, Robocup 2003, Padua, Italy, July 2003

map in a manner similar to grazing. The agent is never idle, always moving about the environment. Secondly, the agent is able to recognize obstacles from a safe distance and begins avoiding the obstacles before they hinder its movement. Obstacle avoidance is a vital ability for any agent to operate in a domain with as much debris as USAR (refer to Section 1.3 for a complete description of the USAR domain). Finally, the agent can distinguish objects resembling victims and move toward those objects. Figure 3.1 shows two examples of victims in the NIST standard test-bed for USAR. Victim identification can be done through various means. Casper et al. [2000] demonstrate that the task can be automated using a combination of different search strategies on a video source. Each search strategy identifies certain distinguishing shapes and colors: skin tone, motion, shapes, etc. If there is enough evidence present to support victim identification, the operator is notified. Victim identification itself is a huge problem within this area, and in no way am I attempting to provide a

thorough model of human victims. Instead, I will be using very simple color information in order to provide a rudimentary implementation of victim identification for the purpose of studying agent control. For more about how victims are identified see Section 4.5.1. The functionality of the schemas described above is intended to approximate the abilities of agents which have competed in the past in robotic rescue competitions (e.g. [Baltes and Anderson, 2002]. Having said that, the autonomous system described in this section is not intended to be a perfect agent for robotic rescue, but rather a vehicle for exploring the issues in teleautonomy described later in this work.

In this behavioral design, the control system is composed of several behaviors, where only one active behavior produces output for the agent's actuators at one time. A finite state machine governs the behaviors, and ultimately the state of the agent dictates which behavior is active at any given moment. Each behavior is composed of a set of motor schemas which serve to activate them. Each motor schema is associated with one or more perceptual schemas. On every cycle of the autonomous control system perceptual schemas gather the relevant sensor data. Each perceptual schema is designed to identify a particular feature of the environment (e.g. the location of a victim). The perceptual schemas produce output that is read by the motor schemas for the active behavior. The behavior reads the output of all of its motor schemas and produces an action recommendation, which is the output of the autonomous control system.

To control an agent in a behavior-based approach, we are concerned with 3 partic-

ular parameters. We need to refer to the desired rotation, the forward motion of the agent, and we need to consider the urgency associated with a motivation. Desirable forward motion is generally either assumed to be a constant value or directly associated with the urgency. We thus formally have 2 components which can be represented using an action vector $\overline{a}$.

$$\overline{a} = \begin{bmatrix} M \\ \\ \Theta \end{bmatrix} \tag{3.1}$$

Here Θ represents the direction to turn, and M is the magnitude. The magnitude represents the urgency associated with a particular action. A high magnitude indicates that the action is more urgent, giving it more influence in the calculation of the resulting action. Each motor schema has an action vector $m(\overline{a})$ associated with it. Each behavior b also has an action vector $b(\overline{a})$ associated with it. Since each behavior b is associated with a collection of motor schemas, to get the resulting action vector $b(\overline{a})$, each motor schema must first generate its own action vector and multiply the magnitude of the vector, M, by its gain value, ϕ.

$$m(\overline{a}) = \begin{bmatrix} \phi & 0 \\ \\ 0 & 1 \end{bmatrix} \begin{bmatrix} M \\ \\ \Theta \end{bmatrix} \tag{3.2}$$

The gain value increases the magnitude of the vector, increasing its priority. The motor schema vectors are changed into their component forms to make their addition simple:

$$m_c(\overline{a}) = \begin{bmatrix} x \\ \\ y \end{bmatrix} = \begin{bmatrix} \cos(\Theta) & 0 \\ \\ \sin(\Theta) & 0 \end{bmatrix} \begin{bmatrix} \phi M \\ \\ \Theta \end{bmatrix} \tag{3.3}$$

To find the resulting action vector for the behavior in component form the sum of all the vectors and their gains is found.

$$b_c(\overline{a}) = \sum_{i=1}^{|schemas|} m_c(\overline{a})_i \tag{3.4}$$

In order for the actuators to understand the action vector, it is transformed back into its magnitude/orientation format:

$$b(\overline{a}) = \begin{bmatrix} M \\ \\ \Theta \end{bmatrix} = \begin{bmatrix} \sqrt{x^2 + y^2} \\ \\ \arctan\left(\frac{x}{y}\right) \end{bmatrix} \tag{3.5}$$

Once transformed into its magnitude/orientation format $b(\overline{a})$, it is ready to be executed by the agent. This completes one cycle for the autonomous control system. See Chapter 4 for the details of how the autonomous control system was implemented.

3.3 Teleoperation

Since the purpose of the system is to blend autonomy and teleoperation, methods for teleoperating the agent were designed. Three methods of control were chosen for the blending control system. These will be described in order of the least abstract to the most abstract in terms of their level of control.

The most basic method of operating an agent is to send the agent low level instructions, such as *move forward 50cm*, or *turn 90 degrees*. These commands are considered the least abstract because in a purely teleoperated system, the agent obeys them directly. The agent has no freedom to decide how the command is to be executed. This requires an interface that allows the operator fine grained control of the agent, such as a joystick. Joysticks are a popular method of controlling both robots, software agents and various vehicles [Arkin, 1998; Arkin et al., 1999]. Joysticks can provide the operator with all the degrees of freedom that a basic wheeled robot is capable of, and are a natural choice for agent control.

An agent can be influenced at a higher level of abstraction through the definition of waypoints, which allow the operator to avoid producing and communicating a series of low level commands. In this type of control, the operator chooses a location or a list of locations that he or she would like the agent to visit. In a purely teleoperated system, when assigned a waypoint, the agent is expected to turn directly to it and approach the waypoint in the shortest path possible, blindly obeying the directional instructions represented and paying no attention to obstacles or other elements of the environment. In a purely teleoperated system the operator is expected to deal with guiding the agent around obstacles and considering any other details of the environment. Such abstract commands can be very dangerous in a purely teleoperated system, since the agent must obey the instruction, even if it is not safe (see Section 1.4).

Enabling the operator to manually switch behaviors and adjust weights associated with the motor schemas contained in those behaviors provides the most abstract

level of control available to the operator in the blending control system. This is considered the most abstract because the operator is not giving direct commands for the agent to execute, or even a series of commands. The operator is giving the agent high level guidance, modifying the agent's personality traits such as aggression and wanderlust [Ali and Arkin, 2000]. Aggression, for example, can be increased by lowering the priority of obstacle avoidance. Wanderlust, on the other hand, can be lowered by making the agent more attracted to its goals. Adjusting weights or switching behaviors effects the output of the autonomous control system in real time.

All three of these modes of control were implemented in order to allow an operator to teleoperate a set of agents. See Chapter 4 more details on this implementation.

3.4 Blending Teleoperation and Autonomy

The autonomous control system described in Section 3.2 and the teleoperated control interfaces described in Section 3.3 provide two distinct sources of control for an agent. Blending those sources of instruction provides significant advantages to using only one of the sources (see Section 1.4 for a discussion of those advantages). There are two tasks that are associated with the blending of instructions. The first is identifying when blending would be the most beneficial and notifying the operator. The second is evaluating the sources of instruction and combining those sources into a single instruction that will be executed by the agent's actuators. The task of notifying when multiple sources of instruction are desirable is referred to here as *Intervention*

Recognition. The task of blending two or more sources of instruction is referred to here as *mediation*.

3.4.1 Intervention Recognition

Recognizing when agents require operator intervention requires examining specific situations in the form of an *intervention recognition system*. The intervention recognition system is ultimately responsible for indicating when the balance between autonomy and teleoperation of agents must be changed, by requesting a change of control to the operator or requesting assistance from other agents. It does this through a knowledge base estimating the degree of likelihood that an agent can or should carry on. The intervention recognition system runs on each agent, analyzing its perceptions and identifying specific scenarios indicative of specific problems and separating these from the situations where autonomy is likely still possible. The design of the intervention recognition itself is one of the contributions of this thesis - it is designed in an extendable manner so that specific scenarios of interest can be encoded in a knowledge-based fashion, resulting in a system that can be used in a wide range of environments.

For the purposes of encoding knowledge useful to the USAR topographical mapping problem, I identified three specific scenarios within the intervention recognition system that make this system useful and successful:

Stuck Agent: The simplest problem to address, but the most common, is a robot

becoming stuck or otherwise immobile. The intervention recognition system identifies when the agent is stuck and signals the operator. A stuck agent is defined as any agent that is sending instructions to its actuators, but the actuators are not completing those instructions. If the agent's actuators are receiving commands, the agent will compare its current sensor readings to past sensor readings attempting to distinguish if there is any evidence supporting movement on the agent's part. If there is little or no evidence supporting movement within the last few perceive-act cycles, the agent is declared stuck.

Confused Agent: In dynamic domains such as USAR, agents can become lost or unable to complete their goals. Identifying when agents are lost or unable to complete their goals is therefore important. The agent designed here contains no elaborate world model, but small pieces of information regarding things that have occurred in the agent's past are stored and used to identify when agents are lost or confused. The agent is able to distinguish certain objects in the environment uniquely using its sensors. Those distinct objects, referred to as *landmarks*, can be tracked. The agent remembers how many times it has sensed a landmark and how much time has elapsed since the last time it has sensed the same landmark. The intervention recognition system uses this information to determine when an agent has returned to the same location too often (e.g. the agent has been to a landmark 100 times) or when an agents has returned to the same location too many times in a short period of time (e.g. the agent has

sensed a landmark 20 times in 5 minutes). In either case, the operator should be notified so that the agent can be encouraged to explore different locations in the environment instead of spending too much time in the same area. For details as to how landmarks were chosen and tracked see Section 4.7.4.

Victim Identified: An important event that occurs in the USAR domain is the location of victims. Victim identification is a very difficult task to perform autonomously [Murphy et al., 2000a; Casper et al., 2000]. Therefore, the intervention recognition system is responsible for identifying when an object in the environment resembles a victim and notifying the operator. Since victim identification is a critical event, the agent should not be operating autonomously once an object resembling a victim is identified. The operator may be required to make a judgment whether a victim is at the location, since the agent is likely to make errors in victim identification. An accurate model of victim identification is not the focus of this work: as such vision alone is used to identify objects resembling victims by their color. Victim identification is handled by the *victim* perceptual schema whose implementation is discussed in Section 4.5.1. Briefly, agents are able to distinguish between actual victims and objects that only resemble victims by their color when they are within $3m$, while outside of $3m$ victims and objects resembling victims are identified as objects of interest. Whenever an object of interest is identified the operator is encouraged to supervise the agent so that victims are identified properly.

When the intervention recognition system identifies a situation that requires the operator to intervene, the operator is notified through the user interface. The user interface is described in more detail in Section 3.5. Briefly, the user interface contains a list of the current available robots and their states. When the intervention recognition system identifies a situation where intervention is desirable, it changes the state of the current robot, updating the user interface. An operator working with the user interface can see that the agent requires assistance, along with a brief message describing the agent's current state, and is able to operate the agent by clicking on the agent's tab on the user interface. See Section 4.8 for more details on how the operator can interact with the blending control system through the user interface.

3.4.2 Mediator

While the intervention recognition system is solely responsible for indicating to the operator that transfer of control is recommended, the mediator is responsible for integrating the operator's commands with those of the autonomous control system (see Section 3.2). While previous approaches (see Section 2.3) have focused on blending operator instructions directly with the autonomous control instructions, this approach is more flexible, allowing the agent to intelligently evaluate instructions before they are blended to ensure that instructions are safe and appropriate to execute. To blend autonomy and teleoperation appropriately, the agent is capable of reasoning about commands that have been sent to it via the human operator. Some commands may

be followed to the letter, while other commands may be integrated with the agent's own desires or completely refused. The agent's ability to reason about commands is of special importance when the operator instructs the robot to perform actions that would put the robot in danger, such as getting too close to an edge and risking a fall. The agent is able to reason about the risks of performing actions requested by the operator (while determining the risk of a particular action would be very difficult in general, evaluating such risks is well within the scope of this research in the task I have chosen). As discussed in Section 2.4, there is a need that the operator ultimately be in control of the system through the user interface. The mediator is equipped with an override facility to allow the operator's commands to be unquestioned. Unlike Arkin and Balch [1998] who implemented teleautonomy as another behavior in a behavior based system, I have implemented a symbolic reasoning system to intelligently blend the human operator's commands and the autonomous behaviors. The presence of symbolic reasoning also allows for a much finer grain of control over the blending, and allows different blending techniques to be active in appropriate situations. The symbolic reasoning system effectively separates the autonomous control system from the human operator's controller interface, something not done in Arkin's earlier approaches.

The mediator acts as the reasoning portion of the architecture, and is responsible for balancing autonomy and teleoperation in the agent. The mediator's operations fall into one of five modes:

Weighted teleautonomy is the expected 'normal' operation of this controller. In

this mode, the mediator is required to observe the current situation and attach appropriate weights to the incoming action requests of the autonomous control system and the controller interface. These weights determine to what extent actions from either source are performed. For example, the operator may indicate a desire to have the agent move to the left, while one of the robot's perceptual schemas (refer to Section 2.2.2.2) may trigger a motor schema provoking the robot's actuators to move to the right. The mediator may reason that the active perceptual schema takes precedence, since it is avoiding a potentially dangerous situation such as moving too close to an edge. The mediator reasons about how much control the operator and autonomous control system should have and assigns weights to those behaviors proportional to their current control priority. Deciding how much control the operator and autonomous control system should have requires a knowledge based system that can apply rules to the current situation and derive appropriate weights. I have performed knowledge acquisition regarding some sub-tasks of USAR. Those sub-tasks and the rules are described further in Chapter 4. Briefly, the mediator uses information from the trouble recognition system as well as the current perceptions of the agents and applies rules to derive weights that are applied to the operator's and autonomous control's instructions. Again, while doing the reasoning described above across all domains would be exceedingly difficult, encoding the knowledge necessary to work within this specific problem is well within the scope of a thesis.

From weighted teleautonomy, we can derive *Fully Autonomous* and *Fully Teleoperated* modes simply by setting the weight from one of the two sources (robot control

or operator) to a zero value, forcing the other source to be the sole contributor. In addition to these, I have also implemented two further modes to allow the human operator to have more detailed control. *Manual Behavior Switching* mode allows the operator to switch through behaviors the agent is capable of, and the agent operates autonomously using only the chosen behavior (see Section 4.7.3). Finally, *Manual Behavior Weight Modification* mode allows the operator to specifically set the internal weights the robot controller places on various behaviors (see Section 4.7.3), allowing the operator to alter how the agent runs autonomously. If the operator believes that the agent is coming too close to a potentially dangerous object, for example, he or she can increase the weight of the obstacle avoidance behavior. The agent still performs the behaviors autonomously in both of these latter modes, but the operator influences the behaviors without explicitly controlling the agent.

Together, these modes allow the subsumption of previous approaches to teleautonomy within a single architecture.

3.5 User Interface

In order to have a means by which the operator can control the robot, a user interface was designed. This user interface allows the operator to see what tasks the agents are currently performing and control the agent should the operator feel that it is required, or if the intervention recognition system (see Section 3.4.1) decides that control should be transferred. The goal of the user interface is to show sufficient

information for the operator to form an accurate understanding of the agent's current situation. The controller interface includes displays for all the relevant sensor data collected by the agent. The sensor data is displayed in a fashion that is meaningful to the operator, easy to understand and that does not clutter the user interface. The user interface displays a map of all areas explored in the environment to aid the operator in teleoperating and supervising agents. The user interface contains the joystick described in Section 3.3 as well as an interface to add and remove waypoints described in Section 3.3 and an interface to manually modify behaviors described in Section 3.3. Details of the implementation of the user interface will be found in the next chapter.

3.6 Summary

This chapter described the design of the four components that together form the blending control system. The next chapter will discuss how each of the four components was implemented, as well as describing how the environment was simulated and implementation issues encountered during the implementation of the blending control system.

Chapter 4

Implementation

4.1 Overview

For this work I implemented a blending control system for multiple robots, and studied this implementation using a widely-accepted multirobot software simulator. The blending control system is able to recognize when a robot's current situation is outside of its abilities to handle, or when some important event requiring the operator's attention occurs, within the confines of the domains I have chosen. At this time the operator is signaled and a suggestion for control transfer can be made to the operator via the operator's user interface. This software is also able to blend operator commands and autonomous processing to the degree the robot believes itself capable of performing adequately, which results in a significant improvement over both autonomous and teleoperated approaches.

This chapter describes the simulator used in the development of the blending

control system. The chapter also discusses the implementation of all four components of the blending control system whose design was described in Chapter 3. Finally, this chapter discusses implementation issues encountered during the implementation of the blending control system.

4.2 Player/Stage

I employed a software simulator already validated by the robotic research community as the main vehicle for simulating the USAR environment. There are several reasons for choosing a simulated environment for this work as opposed to a physical environment.

The first deals with the cost of running agents. Robotic agents able to perform the type of tasks required for this thesis are generally expensive. The Pioneer robot that my work is based on costs between $1,500 to $3,000 USD. Budgetary constraints limit the number of robots available (I verify my techniques on upwards of 3 robots, which was not realistic at the time this research was undertaken). Simulation allows me to run multiple agents without the cost of maintaining and purchasing this equipment. It is my intent in future work to deploy physical agents in the USAR domain that are both cheap and effective.

The second reason is the physical danger to the robots of repeated experimentation. The core of this thesis deals with the idea that robots are expendable in situations that are too hazardous for people to work in. Robots are expendable in

hazardous environments, but the experiments done for this work repeatedly put the agents in dangerous situations. Some situations that I handle are too dangerous to risk the expensive physical robots given our laboratory budget (e.g. agents recognizing not to go down a flight of stairs is fine to test in simulation, but risky in terms of lab equipment in real life).

Finally, it is much more practical and efficient to run simulated experiments then physical experiments in this work since it deals with control issues that are represented well in simulation. I required controlled, repeatable environments for experimentation: a setting nicely provided for in simulation.

I used the University of Southern California's Player/Stage Software package [Gerkey et al., 2001] as the simulation vehicle for testing this work. What makes Player/Stage appealing is not only the fact that it has been validated by the research community and is widely used, but also that it is designed to run simulations using a Pioneer robot and code generated in the simulations can be run on a physical Pioneer robot in the real world. Player/Stage is organized as a client/server platform that allows a client control program to subscribe to the sensors and effectors of a robot in a simulated environment, which precisely models the abilities of those same components of real Pioneer robots.

Player/Stage has two components. The Player component is responsible for providing a network server for robot control. The Player server accepts commands from a client over TCP and sends those commands over an IP network to the robotic platform to be executed by the robot's actuators. Figure 4.1 shows an example of

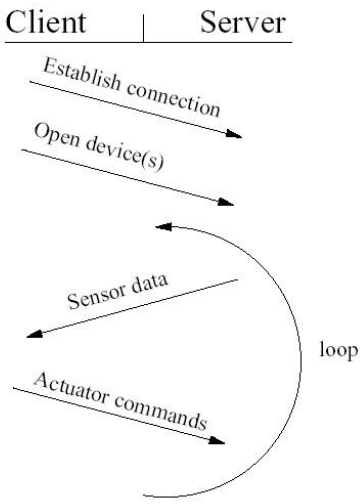

Figure 4.1: Example of communication in Player/Stage [Gerkey et al., 2003].

communication between the Player server and a client. Player provides interfaces to many devices that may be present on a mobile robot.

The Stage component of Player/Stage simulates mobile robots and environments. A client running a control algorithm will connect to Player, and Player connects to Stage. Stage will simulate a physical robot by accepting the commands from Player in the exact same way as a physical robot does. From the perspective of the Player network server, a physical robot and a stage-simulated robot are exactly the same. World files describe environments for stage and are modified to represent the desired environments. The Player/Stage package does not limit the number of robots that are allowed in a simulation. This makes it ideal for multi-agent research.

The simulated USAR environment shares many of the same characteristics of the physical USAR environment. The most important of these is irregularity. The

environments are designed to be difficult for the robot to operate in. Obstacles often block an agent's access to portions of the environment and their placement makes it difficult to use simple wall-following techniques.

Player/Stage allows the freedom of making objects in the environment appear differently to different sensors. For example, Player/Stage allows the user to change the amount of light that is reflected from objects, which affects the accuracy of laser range finders. In addition, Player/Stage allows certain obstacles to be made invisible to vision, sonar and laser. This enables the user to simulate debris that is too low for a robot to detect, or debris that cannot be identified by the CCD camera.

All these features of Player/Stage allow the simulated environment to contain many features of the physical environment, making result obtained in simulated environments more likely to be transferable to the physical world.

4.3 Simulated Agent

The agent used in this work is based on the Pioneer mobile robot. The Pioneer mobile robot is an extendable platform that is capable of supporting a variety of sensors. In this work, agents are assumed to have the following sensors: a laser range finder, a CCD Camera and odometry.

The laser range finder is designed to scan an area and take range and angle samples from objects that reflect light in the environment. The laser range finder can also identify landmarks in the environment to keep track of where an agent has been. This

is useful to identify when an agent is going around in circles.

The CCD Camera is used primarily to identify victims in the environment. The video feed from the camera is processed by a color matching subsystem, the output of which is a list of patches of matched colors, termed *blobs*, identified in the camera's field of view. Pixels in the image representing the camera's field of view are grouped together so that similar colors are identified as being part of the same blob. Each blob identified by the color matching subsystem is listed along with its size, color and estimated location.

Localization is achieved through odometry. The agent "knows" what its position is at all times. Since this work focuses on control issues in teleautonomy and not localization, *perfect localization* is assumed. Perfect localization means that the agent always knows exactly where it is at any point in time. Perfect localization is not a realistic assumption since real-world odometry is susceptible to cumulative errors. If the robot's wheel slips several times, causing noisy odometry data, the robot's position will be offset by some amount, which will gradually increase as the robot's wheels continue to slip slightly. The errors make little difference at first, but after some time the errors accumulate and the robot's position becomes largely inaccurate. Since localization is a widely studied field and a solution to the localization problem is not required to study the relationship between autonomy and teleoperation, perfect localization will suffice for the purposes of this work.

The architecture (described in Chapter 3) is designed such that sensors can be replaced or removed all together. If sensors are substituted or removed, only small

portions of the architecture have to be modified: the respective perceptual schema code. As long as the perceptual schemas are able to identify their target perceptions, the system will still perform its task. This makes the architecture robust to sensor failure.

4.4 Implementation Language

I chose to implement the architecture in Java 1.4.1 on Mandrake Linux 9.0 using the Eclipse development environment.

The decision to use Java as an implementation language was influenced by its ease of use and standard libraries. As a language Java contains many object oriented features that make development using it attractive. Agent development naturally fits with the object oriented paradigm and Java is a powerful object oriented language.

The disadvantage of using Java as opposed to C or C++ is speed. Java applications tend to run slower than their C or C++ counterparts (often significantly so). To improve the speed of the application, instead of using the standard Swing components that come bundled with the SDK, I chose to use the Eclipse development environment [Obj, 2003] that substitutes Swing with SWT components. The SWT components interface directly with the underlying operating system's windowing tool kit, making applications run faster than Swing components.

Using Java requires a Java client to connect to the player server. Maxim Batalin's developed a Java Player/Stage client that is used in this work [Batalin, 2003]. This

code opens a connection to the Player server and parses the information from sensor messages into very simple Java objects. The client is used primarily as a method to request, parse and organize raw sensor information into Java classes, so that the raw data can be accessed by other Java classes.

I am confident that this architecture could be implemented using a variety of languages. It was my intention to choose the one that suited the task best.

All code for this research was designed and implemented with object oriented techniques. Since the code is object oriented, it is very extendable, as well as easy to read and understand. Wherever possible, objects related to one another were organized into object hierarchies.

Now that the tools used in this work have been described, the rest of this chapter is devoted to describing how I implemented the four components of the blending control system, whose design was described in Chapter 3.

4.5 Schemas

The implementations of two types of schemas are described here: perceptual schemas and motor schemas.

Any command sent to the robot is stored as an action vector, and all blending of commands is done through the manipulation of action vectors. For more information on the design of action vectors see Section 3.2. An action vector is used primarily in two operations: first, an action vector may be summed with another action vector,

which requires the vector in component form; secondly, the agent will interpret an action vector as an instruction to its motors, which requires the vector in magnitude/orientation form. An action vector object is implemented as a simple Java class that represents a vector in both its component and magnitude/orientation forms. It contains four data members used to interpret the vector, magnitude, orientation and, x and y components, as well as several methods for manipulating and keeping this data up to date.

4.5.1 Perceptual Schemas

Perceptual schemas are contained in the perception package, which I coded. The perception package is composed of several helper classes that gather sensor data from the Java Player Client and make it accessible to the various components of the blending control system, including the perceptual schemas, the intervention recognition system, and the GUI package.

The *perception* class is responsible for signalling the perceptual schemas that there is new sensor data to process and updates some internal state regarding the position of the robot in the environment and the readings from the laser range finder. The only perceptions that are stored for longer than one perceptual cycle are the laser range values that are integrated into the map of the environment, described in Section 4.8.2. The perceptions are updated by an independent thread that executes every $100ms$.

In order to perform search and rescue in the USAR domain two perceptual schemas

have been implemented: obstacles and victims.

Obstacles The *obstacles* perceptual schema consists of a list of all obstacles that are visible in a single perception cycle. The distance and direction of each obstacle in the list of obstacles is stored. An obstacle is identified using a laser range finder. Each point returned by the laser range finder has a direction and distance, and is considered an obstacle.

Victims Victims are identified using a video camera feed. The *victim* perceptual schema receives a list of blobs from the color matching subsystem and searches for red, green and blue blobs, while all other colors are ignored. Red blobs are identified as other agents, green blobs are identified as victims and blue blobs are identified as objects that appear to be victims from a certain distance. To simulate the uncertainty inherent to the autonomous identification of victims described in [Casper et al., 2000], if a green or blue blob is more than a preset distance from the agent, it is classified as an unknown object of interest that the agent must approach to identify. Once within the predefined identifying area of the robot, green blobs are identified as victims and blue blobs are identified as non-victims and ignored. The direction and distance to unknown objects of interest and victims are estimated and stored in a list of current victims. Victims are re-identified every perceptual cycle. Due to sensor noise, each time a new victim is identified, it is compared to all victims that are currently listed. If the location of the new victim is within a small distance threshold of a victim

that is already known, the victim is not considered distinct. If two victims are close to one another, the perceptual schema will identify them as a single victim. So long as the threshold for deciding whether a victim is distinct or not is small, this will have little effect on the performance of the agents. Identifying that one or many victims are located with close proximity is sufficient for the USAR task, since by definition the purpose of the robots involved in search and rescue is to locate victims so that rescue workers can find them later - the precise number of distinct victims found is not as important.

Each perceptual schema is implemented as a Java class and instances of those classes are stored in the perception class as public data members, so that they can be accessed by components outside of the perception package.

4.5.2 Motor Schemas

Motor Schemas have several things in common. Each motor schema requires an action vector containing the motor schema's recommended action, a gain representing the priority of this motor schema over other motor schemas in the same behavior, a link to the perceptual schema associated with the motor schema and a link to the motor schema's display panel (described in Section 4.7.3). They also require code that the motor schema uses to decide what direction (orientation) and with what urgency (magnitude) the motor schema would like to move. The common elements shared between motor schemas are extracted into an abstract class called MotorSchema.java,

and each motor schema extends the abstract class. The abstract class contains private data members for the gain, the action vector, the associated perceptual schema, and the motor schema's display panel. The abstract class also has an abstract method, updateVector, that is implemented by each motor schema describing how that motor schema derives its magnitude and orientation.

In order to perform search and rescue in the USAR domain four motor schemas have been implemented: *move-ahead*, *avoid-static-obstacles*, *noise* and *move-to-goal*. Each motor schema is described below:

Move-Ahead The *move-ahead* schema is the simplest schema of the four used in this work. Its purpose is to encourage the agent to move straight ahead instead of turning. The gain is a constant value and is manually adjusted to achieve the desired performance. The magnitude of the action vector is 1 multiplied by the constant gain. The orientation for the action vector of this motor schema is always straight ahead.

$$MS\_MoveAhead(\overline{a}) = \begin{bmatrix} \phi & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{4.1}$$

Noise The *noise* motor schema helps prevent the robot from falling victim to a local minimum, a point where the robot cannot make any progress to its goal. *Noise* is implemented as a pseudo random orientation from 0 to $\frac{\pi}{2}$ and a magnitude of 1. The robot will attempt to face the direction for a constant number of time steps. The time steps and gain can be adjusted to achieve the desired results.

$$MS\_Noise(\overline{a}) = \begin{bmatrix} \phi & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 to \frac{\pi}{2} \end{bmatrix} \tag{4.2}$$

Move-To-Goal The *move-to-goal* motor schema takes as a parameter a point of interest. The orientation of the action vector, Θ, is the error of the current facing direction and the direction of the object of interest, $\Theta = \dot{\Theta} - \hat{\Theta}$. Here, $\dot{\Theta}$ is the direction of the object of interest, while $\hat{\Theta}$ is the direction the robot is currently facing. The magnitude of the action vector is a related to the distance of the object of interest. If the object of interest is further away, the magnitude is smaller. The formula for the magnitude is $\frac{1}{d}$, where d is the distance to the object of interest. The relationship between the distance to the object of interest and the robot helps the robot avoid local minima by increasing the chance that the robot will not get stuck trying to get to an object of interest.

$$MS\_MoveToGoal(\overline{a}) = \begin{bmatrix} \phi & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{d} \\ \dot{\Theta} - \hat{\Theta} \end{bmatrix} \tag{4.3}$$

Avoid-Static-Obstacles The *avoid-static-obstacles* motor schema is responsible for ensuring that the robot does not collide with obstacles. A vector is calculated extending from every obstacle towards the robot. Each obstacle vector has an orientation equal to $\pi + \dot{\Theta}$, where $\dot{\Theta}$ is the direction of the obstacle. The magnitude of each obstacle vector is $1 - \frac{1}{d^2}$, where d is the distance of the

obstacle.

$$Obstacle(\overline{a}) = \begin{bmatrix} 1 - \frac{1}{d^2} \\ \pi + \dot{\Theta} \end{bmatrix} \tag{4.4}$$

Avoid-static-obstacles converts each obstacle vector, $Obstacle(\overline{a})$ to its respective component form before and uses them to find $MS\_AvoidStaticObstacles(\overline{a})$:

$$Obstacle_c(\overline{a}) = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \cos(\Theta) & 0 \\ \sin(\Theta) & 0 \end{bmatrix} \begin{bmatrix} M \\ \Theta \end{bmatrix} \tag{4.5}$$

The next step in deriving $MS\_AvoidStaticObstacles(\overline{a})$ is to find the sum of all the obstacle vectors:

$$AvoidStaticObstacles_c(\overline{a}) = \sum_{i=1}^{|obstacles|} Obstacle_c(\overline{a})_i \tag{4.6}$$

Finally, *avoid-static-obstacles* changes $AvoidStaticObstacles_c(\overline{a})$ from component form to magnitude/orientation form deriving $AvoidStaticObstacle(\overline{a})$:

$$AvoidStaticObstacles(\overline{a}) = \begin{bmatrix} M \\ \Theta \end{bmatrix} = \begin{bmatrix} \sqrt{x^2 + y^2} \\ \arctan\left(\frac{x}{y}\right) \end{bmatrix} \tag{4.7}$$

4.6 Autonomous Control System Implementation

The autonomous control system is implemented with two components: the *behavior* object and the *Autonomous Control System* object.

A behavior is composed of several motor schemas. On each cycle a behavior produces an action vector that is the sum of all the outputs of its motor schemas. A behavior object is responsible for requesting that each motor schema update its action vector and the behavior calculates the sum of the updated action vectors. Summation of motor schemas was explained in Section 3.2. A behavior object (Behavior.java) contains data members for an array of motor schema objects, an action vector and a name identifying it. The method generateVector contains code that processes the array of motor schemas, calling updateVector on each one, and adding the resulting updated vector to a running sum. Once each motor schema is processed, the resulting action vector is a behavior's output, and describes the direction and speed that the action should execute.

The autonomous control system object (AutonomousControlSystem.java) consists of a list of behaviors, a behavioral index and an action vector. The list of behaviors contains all the behaviors that the agent currently knows how to execute. Only one behavior is active at any given time. The active behavior is identified by a behavioral index. Each cycle the autonomous control system will request the action vector from the active behavior and assign the value of the behavior action vector to its own action vector. Any object that requires the output of the autonomous control system has access to the resulting action vector via an accessor method, getActionVector.

The autonomous control system is able to control an agent autonomously. Blending occurs when the output of the autonomous control systems action vector is used in conjunction with the output of the teleoperated control system, whose implemen-

tation is described in the next section.

4.7 Implementation of Teleautonomy

Section 3.3 describes the requirement of three components for the blending control system to facilitate interactions between the operator and the agent. These components represent levels of abstraction in commands that the agent is able to interpret from the operator. The three objects, whose implementations are described here are: the joystick, the waypoint manager and the autonomous control panel.

The joystick and the waypoint manager have many things in common including: the requirement for a gain value representing their overall priority; an action vector storing the magnitude and orientation that the operator wishes the robot to move; and code that determines the magnitude and orientation of the action vector at every cycle. The common features of the joystick and the waypoint manager have been extracted into an abstract class called *exterior-control*. The joystick and the waypoint manager extend an exterior-control object, which contains data members for a gain and an action vector. The exterior-control object also contains an abstract method responsible for updating the action vector every cycle. The abstract method is implemented by each object extending exterior-control and describes the actions that must be taken to update the exterior-control object's action vector.

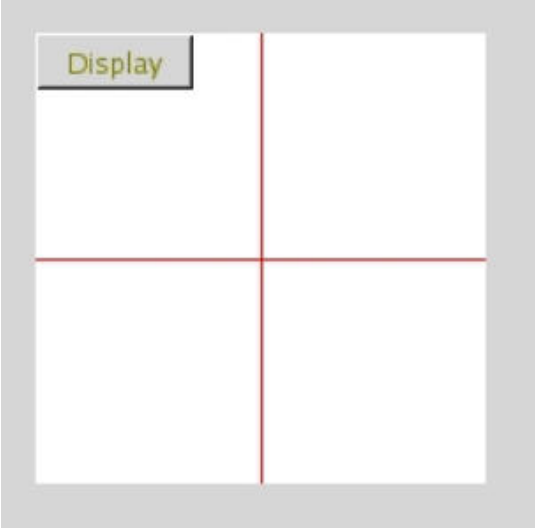

Figure 4.2: The joystick.

4.7.1 Joystick

For controlling the robot's movement, a graphic joystick is displayed on the user interface, shown in Figure 4.2. This simulated joystick, inspired by the work of Bentivegna et al. [1997], is a square area with a crosshair. Moving the robot is achieved by clicking the mouse pointer on an area in the crosshair. The magnitude of the action vector is determined by the distance from the centre that the operator clicks. Points further from the centre of the joystick represent higher magnitudes. For a discussion on how magnitude effects robot movement, see Section 3.2. Points above the horizontal line of the cross-hair produce forward movement, while points below the horizontal cross-hair line produce backward movement. The vertical cross-hair line represents straight ahead or 0 *degrees*. Clicking on a point off to the side makes the robot turn in the direction of that point.

The joystick is ideal for low level commands. The code for the graphics and the joystick logic are contained in the joystick class. The joystick implements the eclipse SWT mouse listener class, allowing the joystick to listen for clicks in the joystick area and find the speed using the Manhattan distance formula:

$$speed = \sqrt{(Click_x)^2 + (Click_y)^2} \tag{4.8}$$

where $Click_x$ is the x coordinate the operator clicked in the joystick area and $Click_y$ is the y coordinate the operator clicked in the joystick area. Finding the direction is done by finding the angle formed by the hypotenuse and the adjacent side of the triangle formed from the origin to the point clicked:

$$direction = \arctan(\frac{Click_x}{Click_y}) \tag{4.9}$$

4.7.2 Waypoint Manager

The waypoint manager allows the operator to direct the agent to a desired location by using the mouse to click on that location in the map window (see Section 4.8.2). A right mouse click on a point in the map window generates a waypoint for the robot that is being controlled when the click is made. This creates a desire for the robot to move toward that point. A series of waypoints can be generated to create a path for the robot. Figure 4.3 shows an example of a series of waypoints generated for an agent. These path points are stored in a queue, and each one is visited in

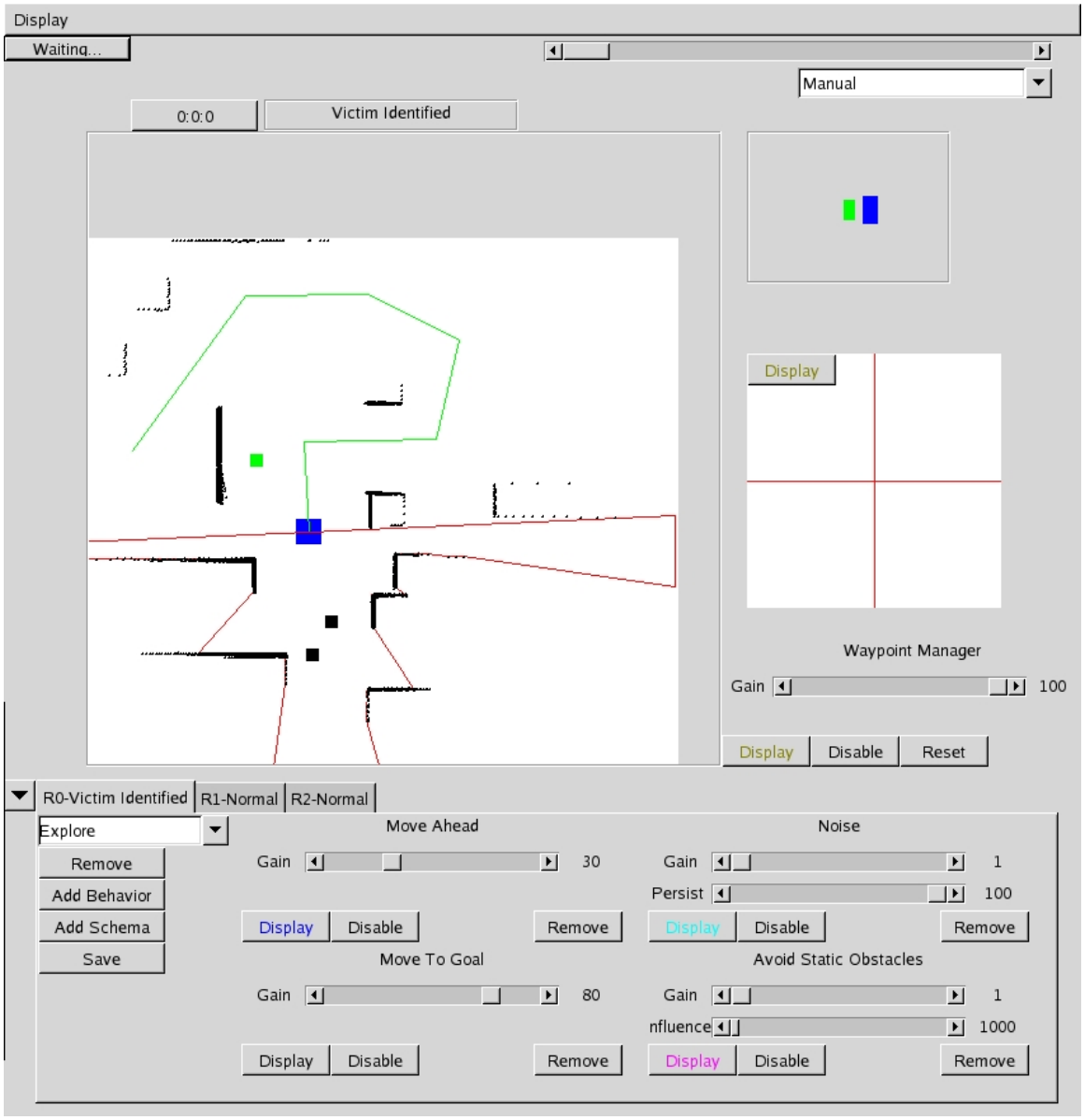

Figure 4.3: Example of an agent with several waypoints selected.

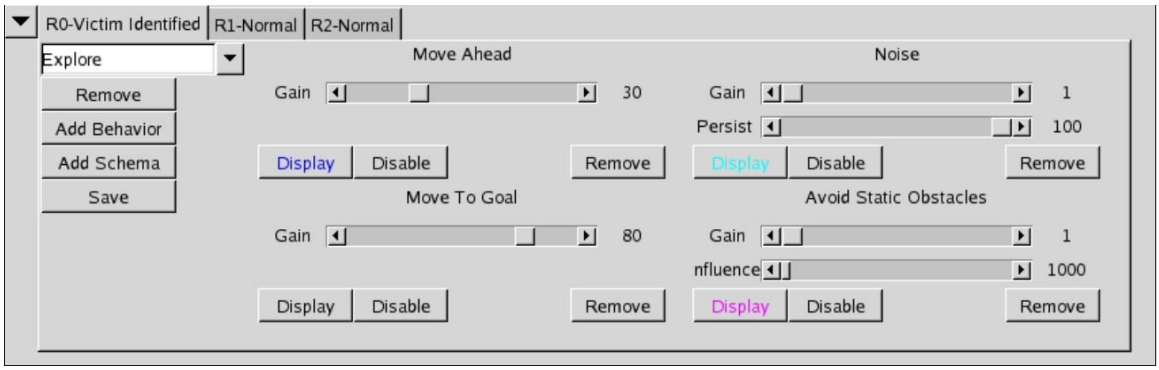

Figure 4.4: The autonomous control panel.

turn. When an agent gets within a certain distance threshold of a point ($12cm$ was chosen for this implementation), it switches to the next waypoint. This method of teleoperated control is less fine-grained than the joystick described in Section 4.7.1, but allows the operator to directly control the movement of the robot with a degree of abstraction. Simple trigonometry is used to determine what direction the robot should move in to get to a waypoint:

$$direction = \arctan(\frac{Click_x}{Click_y}) \tag{4.10}$$

The gain of the waypoint manager can be set manually via a control window in the user interface.

4.7.3 Autonomous Control Panel

Manual control over behaviors and motor schemas is provided by the autonomous control system panel (see Figure 4.4) available via the user interface. Each robot has an autonomous control panel associated with it, and at any point in time only one

autonomous control panel is visible and active. A list of tabs on the active control panel allows the operator to switch from operating one robot, to operating another by clicking on the tab containing the name of the robot the operator would like to control. Clicking on a tab focuses the map (see Section 4.8.2) on the chosen robot.

Each panel has a drop down box that allows the operator to change the currently executing behavior. By default, all robots begin executing the first behavior in the list of behaviors loaded from the configuration file. At any point during operation, the operator may choose to switch the current executing behavior by selecting a new behavior from the *behavior* drop down box. This is an abstract form of teleautonomy that allows the operator to have high level control over the agent's actions. Switching behaviors allows the operator to influence what actions the agent performs without performing them one at a time, reducing the amount of attention the operator must pay to an agent in order to influence its actions. New behaviors can be created using the *add behavior* button. Once the add behavior button is pressed, the operator is prompted for a name, this name will identify the behavior in the list of behaviors displayed by the behavior drop down box. Behaviors can also be saved to a file or loaded from a file.

The autonomous control panel also lists all the motor schemas that contribute to the current behavior, and all the parameters to those motor schemas can be modified using sliders. Motor schemas can be added or removed from a behavior, making the behaviors flexible and customizable. Each motor schema has a *gain* slider so that the gain of the motor schema can be adjusted while the agent is running in order to

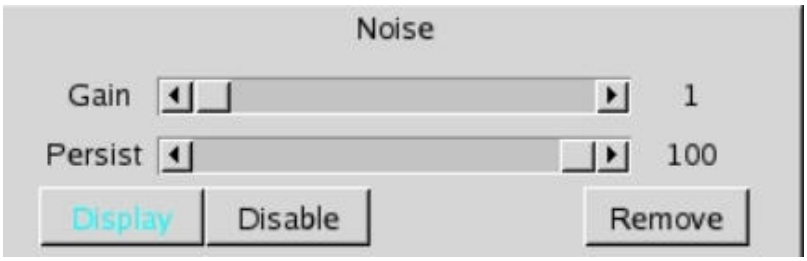

Figure 4.5: The motor schema panel for the noise motor schema.

fine-tune the emergent behavior. This fine tuning allows the operator to help agents achieve their goals at a lower level than simply changing their current behavior, but at a higher level than directly instructing the agent where to go. For example, if the agent is unable to get through a corridor because the corridor is too narrow, the gain on the *avoid-static-obstacles* motor schema can be lowered to effectively make the agent more willing to move closer to obstacles. Figure 4.5 shows the motor schema panel for the *noise* motor schema.

4.7.4 Intervention Recognition Implementation

The Intervention Recognition System is implemented as a package containing the *intervention event* objects, the *intervention recognition* object and a *perception memory* object. The perception memory stores snapshots of the agents perceptions for the past five perceptual cycles as an array of perception instances. The perceptions stored in the perceptual memory do not attempt to create a representation of the world: they are stored as raw perceptions that the perceptual schemas can use to identify interesting things in the environment.

There are three intervention event objects to address each of the three important events that have been identified as occurring in USAR (see Section 3.4.1), *confused identifier*, *stuck identifier* and *victim identifier*. Each of the intervention event objects contains a link to the current perceptions of the robot via the perception class. The stuck identifier object looks at the agent's current location, x and y coordinates, and compares them to the agent's location four cycles previous. If the location right now and the location four cycles ago are the same, and there is a movement instruction being sent to the robot with a speed higher than 0, the agent is considered stuck. The victim identifer relies solely on the victim perceptual schema whose implementation is described in Section 4.5.1. If the victim perceptual schema has found an object that resembles a victim (e.g. a blue or green object) then the agent is considered to be in a victim identifying state. The confused identifier relies on the agent's ability to uniquely identify beacons in the environment. Each victim is labelled with a bar-code like inscription that can be read using a laser range finder, termed a beacon[1]. The confused identifier object keeps track of how many times the agent has been within bar code reading range of a beacon. If the agent surpasses the threshold, then it is identified as confused. The system is extendible since new intervention events can be coded as additional intervention event objects.

The intervention recognition object contains instances of all intervention events that are important to the agent. It is set up to run as an independent thread, but

[1]Interestingly enough, analogous identification tags were added to victims in the 2003 NIST test bed, which was not known at the time of the implementation of this thesis.

in my implementation, it is executed by the mediator each cycle (see Section 4.7.5). The intervention recognition object sends a message to each of the intervention events, inquiring whether the state of the agent should be changed. Once the intervention recognition object is finished interacting with the intervention event objects, it updates the perceptual memory by removing the oldest memory snapshot, shifting the other memory snapshots down and adding the new memory snapshot to the perceptual memory array.

4.7.5 Mediator Implementation

The mediator acts as the central control of the agent. Each cycle, the mediator polls any sources of instruction, blends those instructions, and interprets them so that they can be sent to be executed by the mobile robot. It contains links to the autonomous control system, the intervention recognition system, and a list of exterior control signals.

A cycle of perceiving and acting starts and ends with the mediator. The mediator continues to execute one cycle after another until control is terminated. One cycle of the mediator contains several steps. First, the mediator signals the perception object to refresh all its perceptions. Secondly, the mediator requests an action vector from the autonomous control system, representing the autonomous desires of the robot. Three things can occur next depending on whether the control mode selected by the operator is autonomous, blending, or teleoperated (see Section 4.7.3). In any of the

three cases, control signals are gathered from any exterior controls that apply to the agent's current control mode, and each control signal (including the control signal received from the autonomous control system describe earlier) is evaluated by the *command evaluator* object. The command evaluator is responsible for identifying commands whose execution is dangerous or counter-productive to the goals of the agent. Two such commands have been identified: those that lead the robot into an obstacle, and those that lead the agent away from an unidentified victim. Identifying potential danger or counter-productivity is done by predicting the location the robot would be in if it executed that command, and if that command would leave the robot in a location that is too close to an obstacle or further from an unidentified victim. The magnitude of action vectors for dangerous and unproductive commands are lowered, but not completely discarded, since situations may arise when executing dangerous or unproductive commands are necessary to eventually reach a desired goal. Once commands have been individually adjusted, they are further blended depending on the degree of autonomy the operator has set using the autonomy slider (see Section 4.7.3). If the agent is operating in autonomous mode, the autonomous instruction is executed directly. If the agent is operating in teleoperated mode, the exterior control signal representing the operator's instructions is executed directly. If the agent is operating in blending mode, the autonomous control signal and the exterior control signal are weighted by the degree of autonomy set by the operator and added to each other using vector addition. The magnitude of each vector determines to what degree each of the instructions are followed. The resulting vector is then

interpreted and sent to be executed by the agent's actuators.

4.8 User Interface

For the USAR domain I have identified some important data that will be shown on the user interface: the vision data returned by the CCD camera, a map consisting of laser range points collected by the agent throughout its exploration, the current laser range reading of the robot, the locations of all identified and unidentified victims, and the location of all robots as well as the interfaces for the teleoperated control (see Section 4.7). Figure 4.6 illustrates the complete user interface used in this work.

4.8.1 Vision

The upper right corner of the user interface contains a window displaying the output of the CCD camera. The four types of objects displayed in the window are agents, unidentified victims, victims and negative victims. Agents are displayed as red boxes, unidentified victims are displayed as black boxes, victims are displayed as green boxes and negative victims are identified as blue boxes. The vision is updated on every perceptual cycle, drawing its information from the perception object (see Section 4.5.1). Figure 4.7 shows the vision panel.

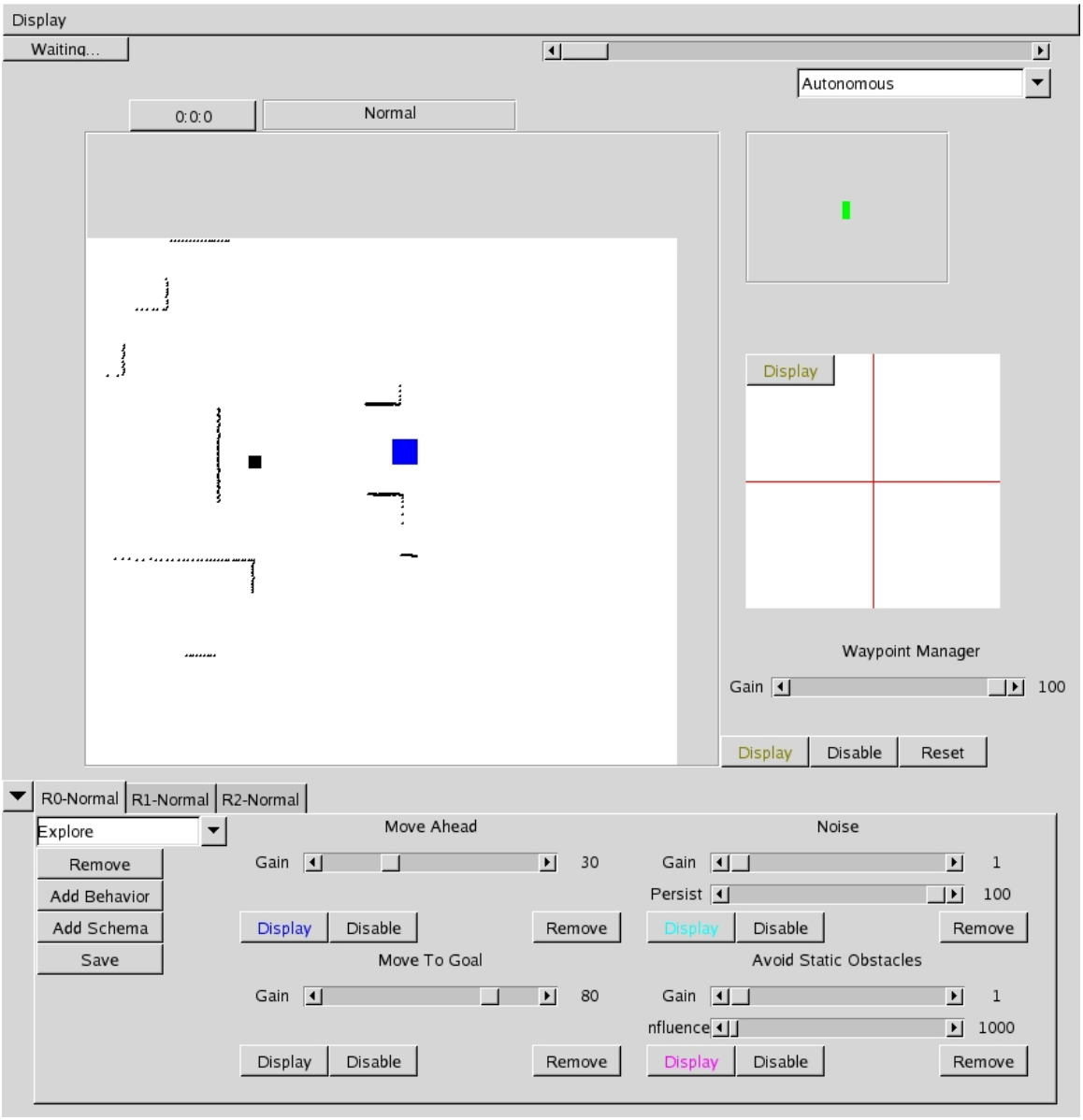

Figure 4.6: Complete user interface.

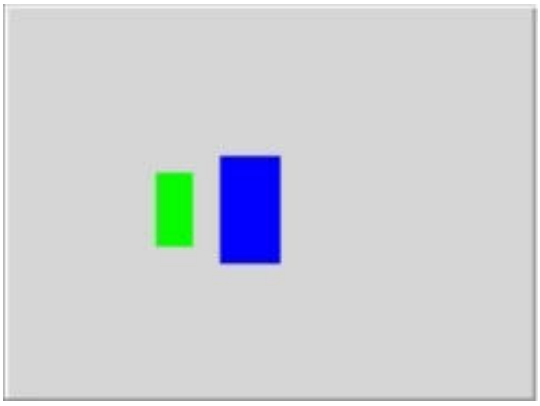

Figure 4.7: The vision panel, currently a victim (larger) and a non-victim (smaller) are in the agent's field of view.

4.8.2 Map

The purpose the map is to organize the perceptions of the robot into a map of the environment to simplify the task of the operator. The map is continuously updated and displayed to the operator on a grid. In USAR, the map is used to show where victims are located for rescue workers. The map also serves as a vessel for communicating information from the operator to the robots, and from the robots to the operator.

The map displays the location of all agents as empty blue boxes. The agent that is currently being controlled is displayed as a shaded blue box. As laser range points are collected, they are added to the map to display obstacles. Victims are displayed on the map as small solid colored boxes: small green boxes represent identified victims, small blue boxes represent identified negative victims, and small black boxes represent

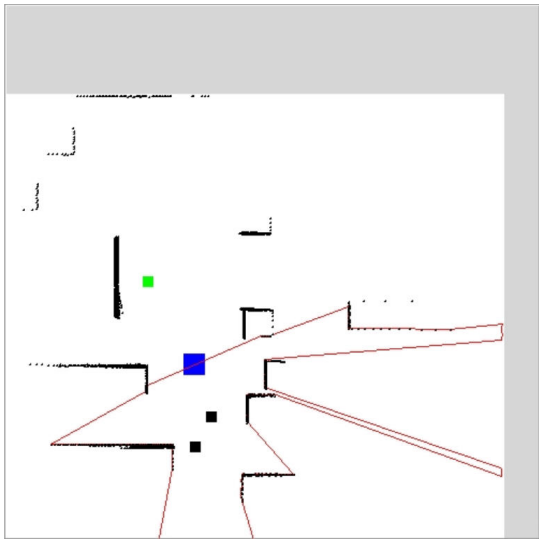

Figure 4.8: The map of the environment so far.

objects that are either victims or negative victims (i.e both the operator and the robot cannot distinguish them).

The operator can interact with an agent through the map in several ways. By left-clicking with the mouse on the map, the view of the map is centred on the clicked location, allowing the operator to explore different sections of the map. By right-clicking with the mouse on the map, a waypoint for the current controlled agent is generated and displayed as a green line. Clicking on the small unidentified victim boxes identifies the victim as either a true victim (left click) or a negative victim (right click). Once the operator identifies a victim, all agents know about the identified victim, ensuring that no victim is identified twice.

The information the map displays can be customized through a menu interface. Clicking on the display menu gives the operator the option of displaying the laser readings of the robot, the path the robot has travelled so far, any waypoints cur-

rently queued, the action vector for all the motor schemas of the active behavior and the exterior signals, a grid across the map, and the map itself can also be toggled. Figure 4.8 shows a sample map of the environment with the agent's laser readings displayed, while Figure 4.9 shows an agent with the motor schema vectors displayed as arrows on the map.

The approach chosen for implementing the map was an occupancy grid approach. Occupancy grids partition an area into a regular matrix with a specific cell size, and the contents of each cell are maintained [Murphy and Sprouse, 1996]. Other available mapping choices exist [Duckett and Saffiotti, 2000], but the simplicity of this approach makes it the most attractive: the robot is required simply to scan the area perceptually, and derive distances to objects in the environment. These maps are not entirely accurate due to sensor error, but they are sufficient for this research. To deal with the inaccuracy, objects in the environment that are sensed infrequently fade from the map's memory, since if they are not sensed often, they are likely inaccurate readings. The map is implemented as a two-dimensional array of short integers, each one containing a number from -1 to 21. Each time the agent discovers an object in the environment, the object's location is converted into an index to the map's array, and the value contained at that array index is incremented by a fixed value (2 was used in this implementation). If the value contained in the map array is less than 20 but larger than 0, the object is displayed as a grey point on the map display. If the value contained in the map array is larger than 20, the object is displayed as a black point on the map display. Values in the array that are smaller than 0 are not displayed.

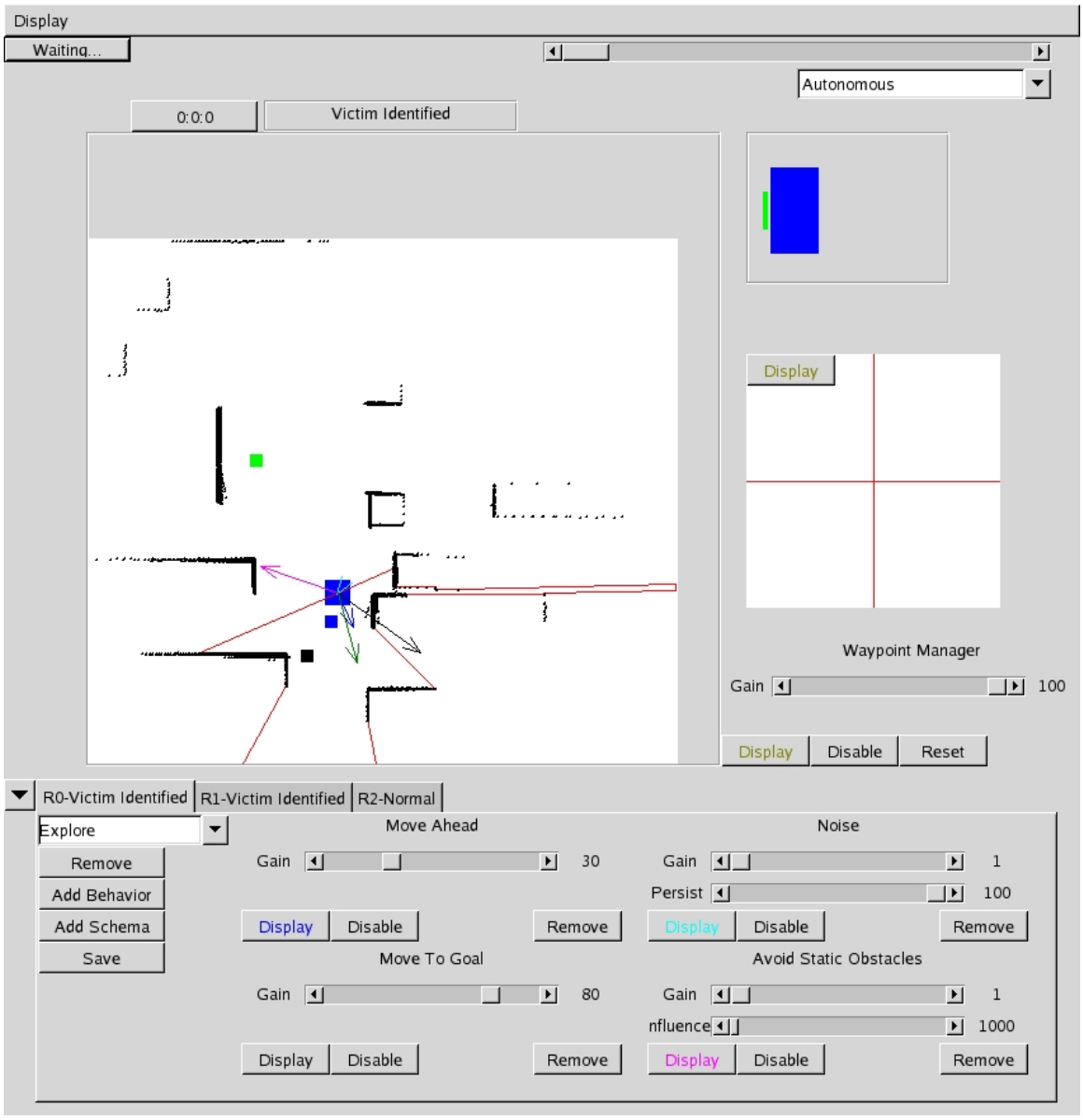

Figure 4.9: Screenshot of the user interface displaying an agent motor schema vectors as colored arrows.

On each cycle, all values in the array that are less than 20 are decremented. By manipulating the array values in the above mentioned fashion, objects that the robot has seen infrequently appear grey and disappear completely if they are not sensed again. If an object is sensed quite often, eventually it will persist in the maps memory and be displayed as a group of black points on the map display.

4.9 Summary

This chapter described many aspects dealing with the implementation of the blending control system. Player/Stage was presented as the simulator chosen for development of the blending control system, and the implementation of each of the three components described in Chapter 3 (the autonomous, teleoperational, and blending control systems) were each described in detail. The next chapter describes a series of experiments (and the analysis of the results of those experiments) designed to evaluate these control approaches and answer the research questions posed in Chapter 1.

Chapter 5

Experimentation

5.1 Overview

In order to examine the effectiveness of the control system developed for this thesis, an experiment was conducted in a simulated rescue environment. The performance of the control system that blends autonomous and teleoperated commands was compared to the performance of an autonomous control system and a teleoperated control system. This chapter discusses the experiment trials and the results obtained. First, the purpose of the experiments is discussed. Secondly, the experimental environment is described, including the types of objects in the environment, how environments are generated, and a method for evaluating the difficulty of the generated environments for the purposes of experimental control. The methodology used in the experiment is then presented, followed by the results of the experiment and analysis of those results.

5.2 Purpose

The purpose of the experiment is to answer the research questions presented in Chapter 1. First, will the addition of teleoperation to autonomous agents increase their overall performance? Secondly, can the introduction of autonomy reduce the number of interactions required between the agent and the operator while maintaining a comparable overall performance? Finally, can an infrastructure be designed to support a practical balance between autonomy and teleoperation in complex dynamic domains? Evaluating agent performance in standard repeatable experimental environments under specific conditions using all three control systems (autonomous, teleoperated, and blending) will allow us to answer these research questions.

5.3 Experimental Environment

All trials were run using the Player/Stage simulation environment package. For details on the Player/Stage package, refer to Section 4.2.

5.3.1 Types of Objects

Several types of objects exist in the USAR environment and must be simulated if an accurate representation of the USAR environment described in Section 1.3 is to be obtained. The Stage simulator allows programmers to create objects, referred to as *entities*, by building them from various components native to Stage. To simulate the USAR environment I built three entity types: robots, obstacles and victims.

Robot entities simulate physical robots. The simulated robot was based on the Pioneer line of mobile robots. An environment file defines a list of sensors and actuators that are available to a control system connected to any particular robot. Each robot is associated with a port number for the Player component of Player/Stage to connect to, and acts as an interface between the intelligent control system and the simulated world. For the trials described here, the dimension of robots were $40cm^2$ and each was equipped with a differential drive, a SICK laser range scanner and a video (CCD) camera with a wide angle lens.

Simulation of the USAR environment requires entities to represent the debris that is found throughout the environment (as described in Section 1.3). This is the purpose of the obstacle entity. Obstacles make robot navigation difficult by blocking the robot's access to portions of the environment, and also make the environment more irregular. Obstacles also provide a method for partitioning areas of the environments into voids (see Section 1.3 for a description of voids). Obstacles in my environment had a dimension of $50cm^2$, and were unmoveable. Obstacles could not be identified by the CCD camera mounted on the robot, simulating both debris below the camera's field of view and the lack of visibility inherent to the USAR domain. Obstacles do, however, reflect the laser emitted from the SICK laser range finder.

Since autonomous identification of victims is not entirely accurate (refer to Section 4.5.1 for a discussion on victim identification), I devised a method for simulating differences between agent and operator victim identification. Environments contained two distinct types of victim entities: real victims and negative victims. From three

meters away, both victims and negative victims looked the same, representing an object that may or may not be a real victim. Once a robot was within three meters, an operator could make a distinction between negative victims and real victims. Once a robot was within 1 meter, the robot could distinguish between negative victims and real victims. The difference in the distances required for operators and robots to distinguish whether an object of interest is a victim or not is an attempt to reflect the capabilities that the current state of victim identification for mobile robots in dynamic environments compared to the ability of an operator trained in urban search and rescue. The actual distances are a generous approximation and do not represent the exact abilities of real robots and real human operators. Real victims were represented by green boxes in the environment, while negative victims were represented by blue boxes in the environment. Victims did not reflect the laser emitted from a laser range finder, but they were visible to the CCD camera.

5.3.2 Generating and Evaluating Experimental Environments

An environment generation program, written in Java as part of this thesis, generates the simulated rescue environments used in each trial. A call to the environment application passes the desired characteristics of the environment to the Environment Generator and the number of environments to generate. The characteristics supplied are discussed below, and allow the generation of simulated rescue environments having varying degrees of difficulty.

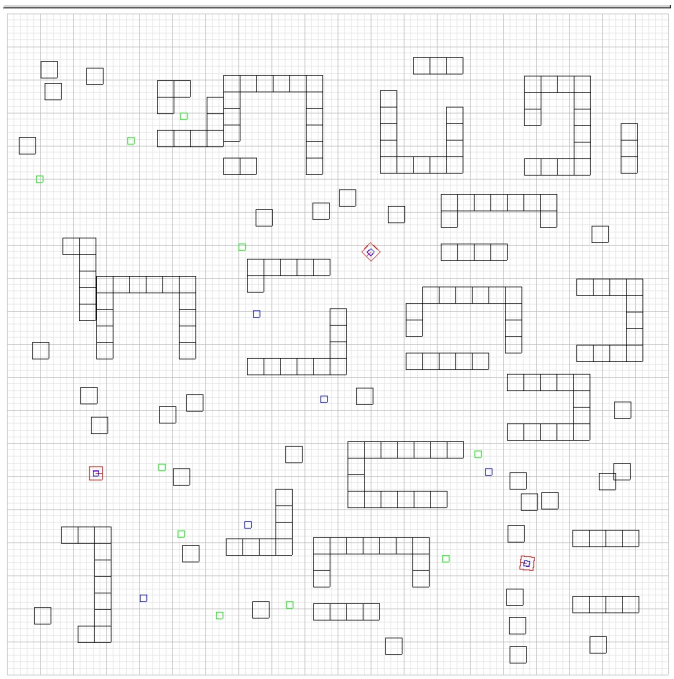

Figure 5.1: Example of an environment generated by the environment generator.

The environment generator creates environments using a three step process. First, the environment generator creates and places all obstacles. Second, the environment generator creates and places all victims. Finally, the environment generator creates and places all robots. Figure 5.1 shows a sample environment.

Each environment is assigned a *target obstacle coverage* by the experimenter, dictating how much of the total area of the environment is allowed to be composed of obstacle entities. To satisfy the target obstacle coverage, the environment generator splits the generation of obstacles into two tasks: generating voids and adding single obstacles. Voids are sets of obstacles arranged in such a way that they form open areas. Since voids are formed from obstacles that in turn form walls, obstacles are placed to form voids to the degree allowed by the target obstacle coverage. The size of

the voids are pseudo-randomly generated using a random number generator (supplied with Java SDK 1.4.1). The size of a void is randomly chosen between the maximum size of voids, $300cm^2$, and the minimum size of voids, $100cm^2$. The height and width of the voids are generated independently. Candidate voids are placed in the environment by randomly generating an x and y coordinate for the upper left corner of the void. The environment generator checks to see if the candidate void overlaps with any voids that are already in the environment, encouraging void dispersement. If the candidate void overlaps another pre-existing void, the candidate void is discarded and a new one is generated. After every void is generated, the total environment coverage is updated. While voids are made up of obstacles, the process of generating voids is unlikely to generate the precise degree of desired total obstacle coverage. It is also possible for the purposes of consistency between environments to set a maximum limit on the number of voids. In either case, obstacles must be generated outside of voids in order to meet target obstacle coverage.

The second task for obstacle generation is adding additional single obstacles to meet the target obstacle coverage. To add single obstacles to the environment, the environment generator randomly chooses valid coordinates in the environment. Valid coordinates are those that will place obstacle at least $120cm$ from any void, and at least $50cm$ from the nearest obstacle (the distance between two entities is measured from center to center). The environment generator continues to add obstacles to the environment until the actual environment coverage is within one half of a percentage point of the target obstacle coverage.

The environment generator adds victims and negative victims to the environment using the same pseudo-random number generator described above to generate a valid x and y coordinate. The environment generator calculates the distance between the candidate victim position and all obstacles that have already been added to the environment, in order to ensure that the distance is larger than $60cm$, which is the minimum distance allowable between victims and obstacles.

The environment generator adds robots to the environment with the same method used for victims, ensuring there is $140cm$ between the center of robots and walls.

5.3.3 Environment Evaluation

All environments were evaluated to determine how difficult they were relative to one another. Environment evaluation enables the results of the experiment described here to be repeatable and comparable, by using environments that are consistent with respect to their difficulty. Environment evaluation also allows the construction of experiments that examine the scalability of the system, since as the environment gets more difficult, the agent's effectiveness should not deteriorate too quickly.

Environments were evaluated in two ways, based on different levels of what is means for an environment to be difficult.

One way of measuring the difficulty of an environment is to count the number of local minima the environment contains. When in a local minimum, the agent is unable to make any progress towards its goal and stays stuck in the local minimum

position until there is outside intervention. This is because the agent is unable to perceive a more rewarding place to be, and so the agent believes that it's current position is the best possible, since none of the actions that the agent can perform at that time will take it to a position that is more rewarding.

The number of local minima contained in any environment generated for this thesis was calculated and stored when the environment was generated. Environments having a local minimum count above a specified threshold were discarded. Environments having a local minimum count below a specified threshold were also discarded.

Local minima are a consistent method of examining the difficulty of the domain at a low level. Above this level, however, the number of particular domain features (e.g. hallway width, objects in the environment, and so on) also directly affect the difficulty of traversing the environment. These higher level features may interact to cause local minima, but may also have difficulties independent of local minima. Because of this, a set of higher level heuristics was developed to supplement a local minima count in order to objectively describe the difficulty of an environment. These heuristics are:

1. Environments containing unaccessible voids because of their placement or the placement of obstacles are considered more difficult.

2. Environments where the placement of voids created too many narrow hallways where agents were likely to get stuck are considered more difficult.

3. Environments where objects are clustered are considered more difficult.

The difficulty rating reflects how difficult the environment appears to be accord-

ing to the above heuristics, and is assigned by an individual who manually checks the environment. The precise criteria for selecting environments is explained in the experimental methodology section below.

I used a combination of the two methods described above to evaluate environments.

5.4 Methodology

All trials were run on a Mobile Pentium 4 1.7Ghz with 512MB of RAM running Mandrake Linux 9.0. In all trials requiring a human operator, I acted as the human operator. It is my opinion that the operator requires an understanding of the system developed as part of this thesis and USAR in general to perform effectively, making myself the obvious choice. A person selected randomly from the general public would likely not have the skills required to perform the human operator task successfully. I believe, however, that if another person experienced in operating mobile robots were to be given a brief introduction to the system described in Chapter 4 and then attempt the trials described here, the result of those trials would not be significantly different.

All environments generated for the trials described in this chapter shared several characteristics. Every environment represented a $20m^2$ area in the real world. As a result of the environment size every pixel in the map displayed on the user interface (see Section 4.8.2 for details on the map) represented $2cm^2$, and therefore the map

was $1000 pixels^2$. Every environment contained 3 robots, 10 real victims, 5 negative victims and a maximum of 20 voids. Additional criteria were added to the environments to ensure their duplicability and similarity. First, between every obstacle and every void there was a minimum distance of $120cm$, so that multiple obstacles could not cluster too closely to voids making too many voids inaccessible. The distance between the center of any two obstacles in the environment could not be less than $50cm$, making it impossible for obstacles to overlap more than a few centimeters. The distance between the center of any real or negative victim from the next closest real or negative victim was at least $60cm$, so that there was some clearance between victims and obstacles. All robots started in a position that was at least $140cm$ from the center of any obstacle, so that agents would not begin a trial already stuck or overlapping an obstacle. Voids were all between $100cm^2$ and $300cm^2$, containing at most 3 openings (passages between voids). Too many openings in the voids fragmented them too much to resemble or be useful as voids.

The principle varying factor in the environments employed in the trials described below was their target obstacle coverage. Four variations of coverage were chosen: 5%, 10%, 15%, 20%. I began by generating twenty environments for each of the chosen target obstacle coverage. In these eighty environments, the local minima count ranged from 0 to 49. From each group of twenty environments, I applied difficulty controls to select five environments for each obstacle coverage category by defining a range of acceptable local minima counts and applying heuristics to choose from the remaining environments. The range for each category was chosen by taking

the median local minima count for the twenty environments in that category and rejecting environments whose local minima differed from the median by more than a specific range, in order to eliminate any outliers. For most categories, that range was 2 local minima. In the 5% category, all twenty environments generated had between 0 and 3 local minima. As a result, 5 environments were chosen based purely on the heuristics discussed in Section 5.3.3. In the 10% category, the local minima count ranged from 4 to 9, but the majority of the environments had counts between 5 and 8, so of those environments 5 were chosen using the heuristics. In both the 15% and 20% categories the variation in local minima counts was much larger, ranging from 9 to 29 in the 15% category and 26 to 49 in the 20% category. For the 15% category, I rejected environments with less than 18 local minima or more than 20 local minima, since the majority of the environments had local minima counts in that range. For the 20% category, I rejected environments with less than 34 local minima or more than 38 local minima (a range of 4 rather than the 2 used in the other categories). The range for the 20% local minima was increased because as the obstacle coverage in the environment gets larger, there is less chance that environments will have the exact same number of local minima, since there is only so much room available for obstacles in the environment. After rejecting a portion of the environments by their local minima count, I applied the heuristics from Section 5.3.3 to reject environments that had undesirable features. Of the remaining environments, I chose 5 from each category to provide an equal number of environments for each desired level of obstacle coverage. As a result of this, a total of twenty environments of consistent difficulty

Control System	Obstacle Coverage 5%	10%	15%	20%
Autonomous	5	5	5	5
Teleoperated	5	5	5	5
Blending	5	5	5	5
Total Trials	**15**	**15**	**15**	**15**
				60

Figure 5.2: Breakdown of trials

were available upon which to perform trials.

The trials can be broken down into three categories (as shown in Figure 5.2): autonomous trials, teleoperated trials, and blending trials. The category of the trial defines which control system was used on each of the three agents in the trial. The autonomous trials used only the autonomous control system described in Section 3.2 as the mobile control system for each agent. The teleoperated trials used only the teleoperated elements of the blending control system described in Section 3.3. Finally, the blending trials combined elements of autonomy and teleoperation by using the complete blending control system described in Chapter 3. Each of the three trial types were run using all 20 of the environments chosen above, resulting in the deployment of each of the three control system in 20 individual trials, adding up to 60 trials in all. Table 5.2 lists a breakdown of all trials.

Each trial had a duration of thirty minutes. Trials halted if a target environment coverage was met by the robots. For the 5% obstacle coverage environment, the

trial would end if the total environment coverage reached 85%. For the other three experiment coverage values, the trial halted when 80% of the total environment was covered. The divergent halting condition for trials where only 5% of the environment was covered in obstacles was due to the speed with which the agents could explore such a low obstacle environment. I found that using a higher target environment coverage value for the trials with 5% obstacle coverage had more interesting results, while 85% would have taken a significant amount of time more for environments with higher than 5% obstacle coverage, since there were delays in getting agents to cover the last few percent of the area.

How the data gathered from these trials was used to evaluate the agent control mechanisms is described in Section 5.5. In addition to describing the particular criteria used for generating experimental domains, replicatability also requires some description of the particular agent behaviors employed in these trials.

At any point in time during the search and rescue task, the autonomous control system was performing one of two behaviors: exploring the environment searching for victims, or approaching an object that resembles a victim in order to identify it. This is similar to Balch's foraging behavior [Balch, 1998a], which is composed of exploring, acquiring and returning home. The primary difference between the agents deployed in these experiments and the ones described by Balch is that my agents do not return to home once they locate a victim. Once a victim is located, the agent continues to explore, looking for another victim. I defined an *explore* behavior that consists of four motor schemas (see Section 2.2.2.2 for a complete description of motor

schemas) *move-ahead*, *noise*, *move-to-goal* and *avoid-static-obstacles*. The parameters for all motor schemas were initialized with a set of default values. The default values were determined through various preliminary trials, manually adjusted so that the emergent behavior of the exploring behavior appeared effective. The *noise* motor schema started with a gain of 10 and a persistence of 100. The *move-ahead* motor schema started with a gain of 30. The *avoid-static-obstacles* motor schema started with a gain of 2 and an influence of 1280. Finally, the *move-to-goal* motor schema started with a gain of 80. For the autonomous trials, the behavior and its schemas remained unchanged throughout all trials. In the blending trials, however, as the trial progressed the operator was allowed to change the behavior configuration. For example, if an agent would not get close enough to a wall, the operator could decrease the influence of the *avoid-static-obstacles* motor schema. If the agent tended to run into walls, the operator could increase the gain of the *avoid-static-obstacles* motor schema. In most cases the *avoid-static-obstacles* gain and influence were decreased by the operator during the trial for more complex environments, encouraging agents to be more risky (i.e. approach walls closely), since the operator could help the agent if they became stuck. Also, the gain for the *noise* behavior was often decreased by the operator during the trial, so that the agent was more likely to directly obey a waypoint command without wandering too far off track. No behavior setup was used for teleoperated trials, since the agent obeyed every command directly.

5.5 Performance Evaluation

The experiment performed was designed to evaluate the performance of the blended control mechanism described in Chapter 3 and Chapter 4 in comparison to fully autonomous and fully teleoperated approaches. In order to assess the performance of the control mechanisms, data were gathered to allow comparison across four categories:

- Percentage of environment covered;
- Number of real (positive) and negative victims identified;
- Time spent immobile;
- Number of interactions between the operator and the control system.

Data was gathered in each trial to allow evaluation according to these criteria.

Environment coverage contributes to the overall performance of the system. The more area covered, the more likely all victims will be found. The goal of urban search and rescue is to search as much of the area as possible, starting with those areas that are most likely to contain victims. If coverage is low, then there can be no guarantee that areas unsearched did not contain victims. Any search and rescue control system should thus boast a high area coverage. Both the rate that area is covered and the total coverage are important factors in estimating the performance of a control system.

In the trials I performed, both the agent and the operator had the capability to identify real and negative victims once they were in range. The experimental results

in the next section will illustrate how many victims were properly identified. By properly identified, I mean positive victims labelled as positive, and negative victims labelled as negative. Ideally, all positive and negative victims will be discovered, and none misclassified. The total number of victims properly identified and the rate that they were found is significant to the overall performance of the system. Faster victim identification and higher numbers of victims identified contributes to better overall performance.

The amount of time spent immobile is the number of milliseconds that the agent was sending movement commands to its actuators, but the agent's position remained unchanged. When an agent becomes immobile, it is no longer effective at finding victims. Higher amounts of immobile time indicate lower effectiveness of the control system. The time spent immobile was only recorded for trials using the autonomous control system and the blending control system. Autonomous agents are very susceptible to getting stuck in environments such as those used for the trials described here. As such, I am interested in seeing if adding teleoperation to autonomous agents can help agents avoid getting stuck and also help them free themselves once stuck. Since the teleoperated agents have no autonomous control system, they are equally or less likely to get stuck then blending agents, making any comparisons between the time spent immobile by teleoperated agents vs. blending agents unimportant for this research.

Data was also collected indicating how many interactions the operator had with the waypoint manager. Three types of interactions were recorded: the number of

waypoints generated, the changes in waypoint gain, and the number of waypoint resets. The operator generated waypoints by right clicking on the map, the waypoint gain was modified via the waypoint manager using a slider, and finally, the operator could clear the waypoints before the agent arrived at them using the reset button located on the waypoint manager (see Section 4.7.2). The number of interactions the operator had with the user interface provides insight into how much attention the operator was focusing on the robots. Quantifiably measuring the cognitive load of an operator is difficult, and operator interactions was chosen as a satisfactory measure for the purposes of this research.

The next section summarizes and analyzes the experimental results collected for the three different experiment categories described earlier.

5.6 Results

5.6.1 Teleoperated Results

Figure 5.3 shows the average percentage environment coverage achieved by the teleoperated agents across all twenty teleoperated trials. Notice that as the amount of obstacle coverage grows, the total amount of environment covered by the agents decreases. Performance degraded slightly when obstacle coverage increased from 5% to 10%, and again from 10% to 15%. In the 5%, 10%, and 15% obstacle coverage environments, the agents reached between 80 and 85% total environment coverage. For all

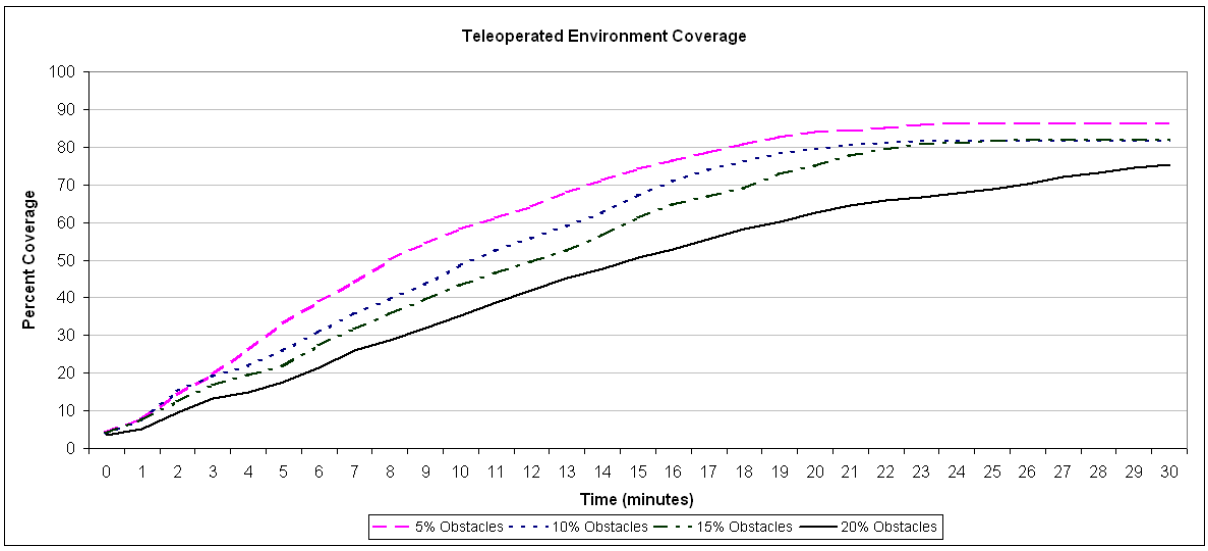

Figure 5.3: Average (n=5) environment coverage achieved by teleoperated agents in 5%, 10%, 15% and 20% obstacle coverage environments.

categories but the 20% coverage, the teleoperated agents consistently achieved target environment coverage before the trial's time limit of thirty minutes was reached.

Figure 5.4 illustrates the average total number of victims teleoperated agents properly identified across all trials. The graph shows the sum of all positive victims correctly identified and all negative victims correctly identified. Since there are 10 positive victims and 5 negative victims, the most victims found in any trial should be a value between 0 and 15. However, agents identify the number of unique victims according to their location in the environment. The location of each victim is estimated by the height, width and location of the blob representing the victim in the agent's field of view. Since sensor readings are subject to noise, victim locations often jump around slightly. Agents attempt to account for sensor errors by assuming that victims whose locations are within a certain threshold are not distinct. Sometimes,

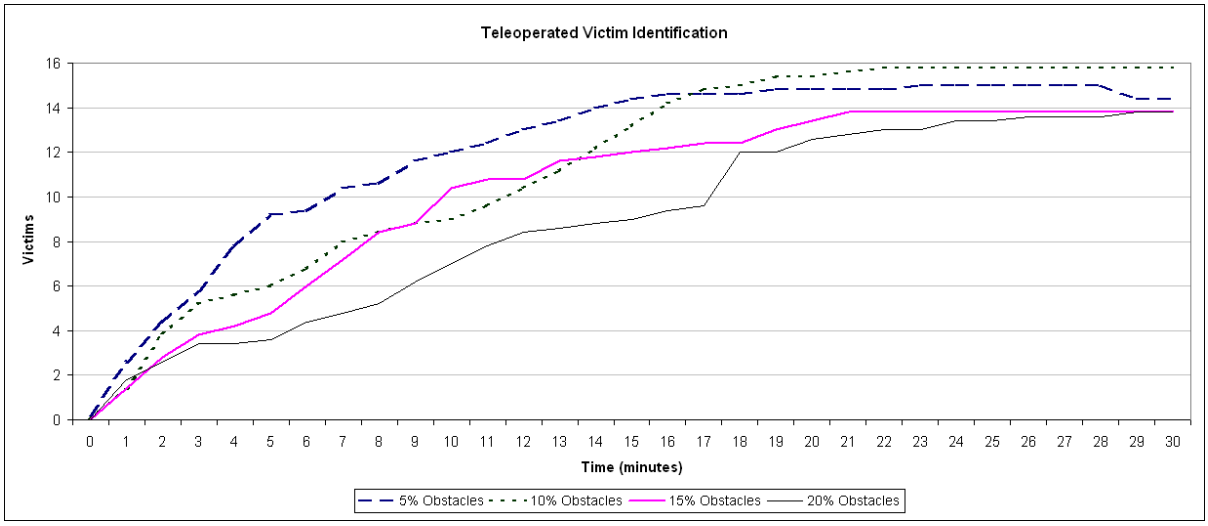

Figure 5.4: Average (n=5) number of victims identified by teleoperated agents in 5%, 10%, 15% and 20% obstacle coverage environments.

however, the distance between two victims is very close to that threshold, causing an agent to believe that there is one victim when there is in fact two, and the agent may flip back and forth for a short time, unsure of whether there is one or two victims. Additionally, in rare circumstances the sensor readings are so noisy that the agent may identify one victim as two distinct victims for a short period and then correct itself later when it discovers there is in fact only one victim. Both of the anomalies described above occur rarely, but an error of plus or minus one victim identified is a safe margin of error. The anomalies appear twice in Figure 5.4, near the end of the trials the average number of victims identified in the 5% obstacle coverage environments drops by half a victim and in the 10% obstacle coverage environments the average number of victims identified rises to 15.5 victims. The teleoperated agents located all the victims in both the 15% and 20% obstacle coverage environments, while missing

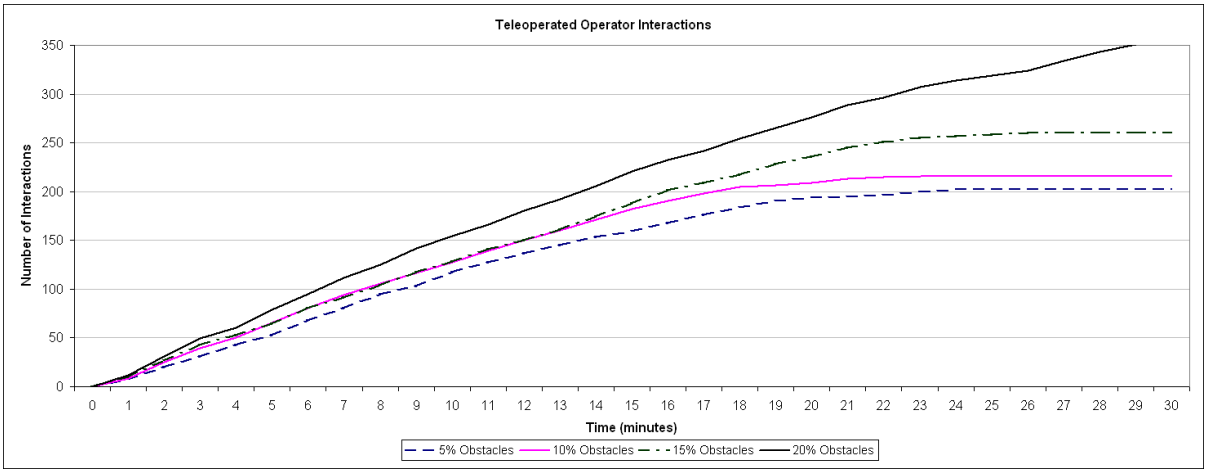

Figure 5.5: Average (n=5) number of interactions operators had with teleoperated agents in 5%, 10%, 15% and 20% obstacle coverage environments.

only one victim on average in the 5% and 10% obstacle coverage environments. There is only a slight degrade in effectiveness as the environments get more complex. Notice that in the 5% obstacle coverage environments, victims are located more quickly than in the 20% obstacle coverage environments. The speed with which victims are identified in the 10% and 15% obstacle coverage environments are almost equal. The degradation in performance as environments become more complex indicates that environments with higher obstacle coverage are more difficult to locate victims in, which is to be expected. However, the teleoperated agents generally maintain a high degree of victim identification.

Figure 5.5 shows the average number of interactions required between the operator and the agents to achieve the effectiveness described above with respect to the total environment coverage and the number of victims identified across all trials. Notice

that the number of interactions increased linearly as the trials progressed. In the 5%, 10% and 15% obstacle coverage categories, the increase in operator interactions levels off at a point late in the trial. This is because at that point in the trial, the agents had already reached their target environment coverage. Operator interactions increased slightly as the environment became more complex. Notice that the 5%, 10% and 15% obstacle coverage environments required roughly the same number of operator interactions, while the 20% obstacle coverage environment required slightly more. The rate of operator interactions for the 5% obstacle environments was roughly 10 interactions per minute. The rate of operator interactions for the 10% and 15% obstacle environments was roughly 12 interactions per minute. Finally, the rate of operator interactions for the 20% obstacle environments was roughly 14 interactions per minute.

5.6.2 Autonomous Results

Figure 5.6 shows the average total environment coverage achieved by the autonomous agents across all trials.

Of the three control mechanisms studied, the autonomous control system was affected the most by the increase in obstacle coverage. Agents were consistently able to reach their target coverage in trials where the obstacle coverage was 5%. As the obstacle coverage increased from 5% to 10%, performance degraded significantly and agents rarely achieved their target coverage. The performance degraded considerably

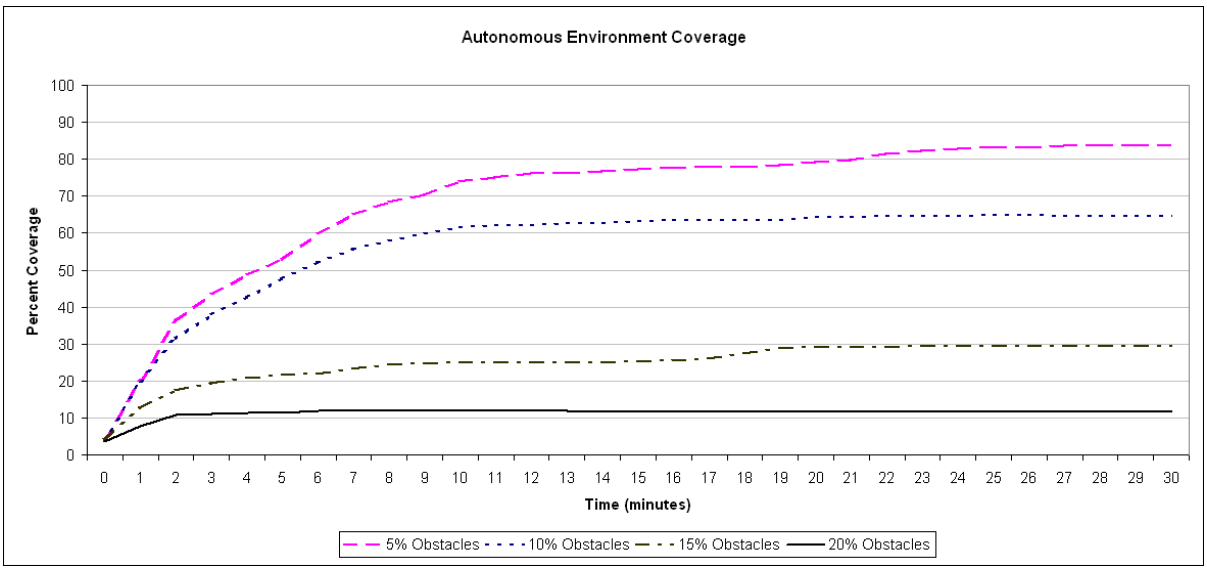

Figure 5.6: Average (n=5) environment coverage achieved by autonomous agents in 5%, 10%, 15% and 20% obstacle coverage environments.

when the obstacle coverage increased to 15%. In this case, agents could rarely achieve a total environment coverage of more than 30%. This performance degradation magnified as the obstacle coverage increased from 15% to 20%. In environments where the obstacle coverage was 20%, the agents were unable to explore the environment beyond a close vicinity to their starting location. Once the agents reached a local minimum, they would often be stuck for the duration of the trial.

Figure 5.7 shows the average effectiveness of the autonomous agents with respect to victim identification across all trials. In trials where the obstacle coverage was 5% and 10%, the autonomous control system performed very well for the majority of the simulation, identifying on average all but one victim. The agents identified victims that were in the open particularly efficiently. As the domain increased in complexity, however, the autonomous agents were more likely than the other agent types to fail

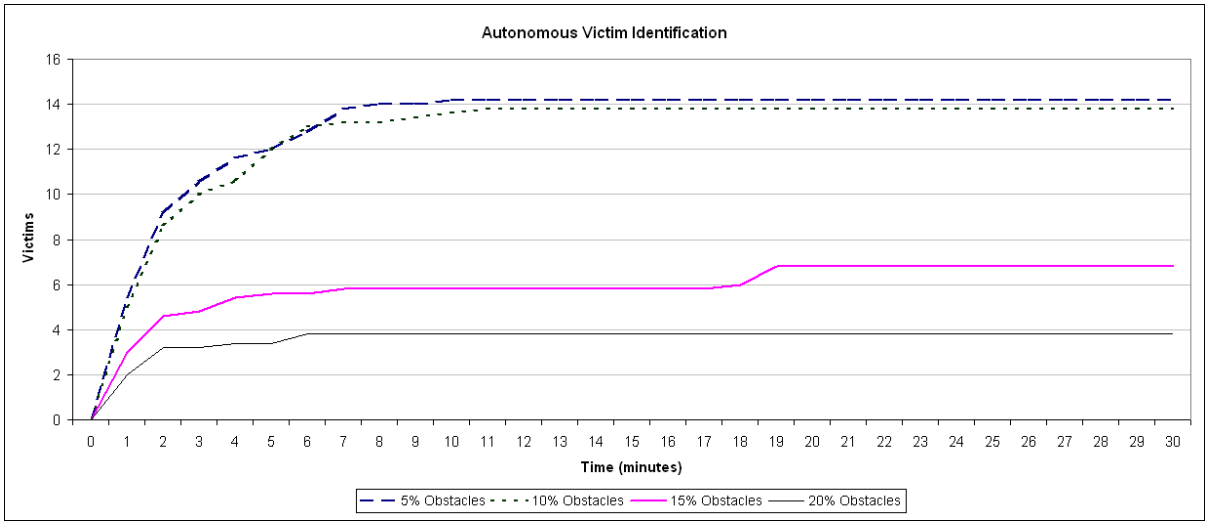

Figure 5.7: Average (n=5) number of victims identified by autonomous agents in 5%, 10%, 15% and 20% obstacle coverage environments.

to identify one or more victims. In fact, as the obstacle coverage increased, the performance of the autonomous control system degraded significantly. The sharp increase of victims found in the first three minutes of the simulation in both the 15% and 20% trials indicates that the control system was able to identify and categorize the victims that were close to where the agents began their search, but failed to explore enough area to find more victims. That is, the poor performance with respect to total area covered is reflected in the number of victims identified. After roughly five minutes of the simulation in the 15% and 20% obstacle coverage trials, the rate of victims discovered drops off to a point where the autonomous control system failed to find more than one additional victims.

Figure 5.8 shows the average of the total amount of time the autonomous agents spent immobile across all trials. As the complexity of the environment increased, the

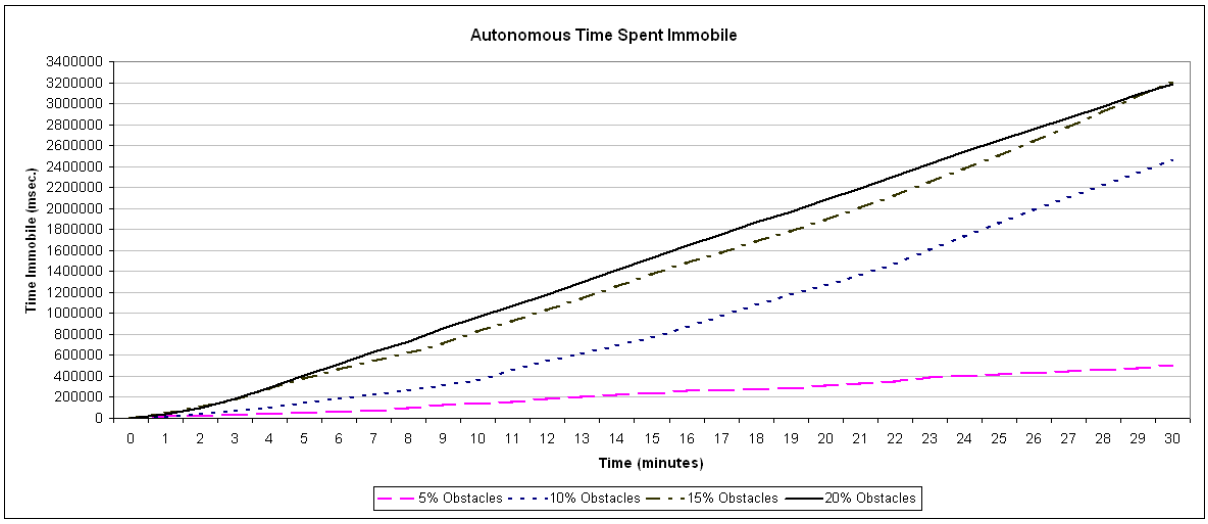

Figure 5.8: Average (n=5) time autonomous agents spent immobile in 5%, 10%, 15% and 20% obstacle coverage environments.

amount of time agents spent immobile also increased significantly. The difference is most pronounced between the 5% and 10% obstacle coverage environments. In all trials, once autonomous agents rendered themselves immobile by driving into a wall, or getting stuck in an opening too small to fit through, they were often unable to find the right combination of turn and reverse commands to get themselves mobile again. Autonomous agents unable to free themselves contributed to the high amounts of time spent immobile in the autonomous trials.

5.6.3 Blending

Figure 5.9 shows the percentage of the environment blending agents were able to cover, averaged across all trials. The blending control system achieved its target

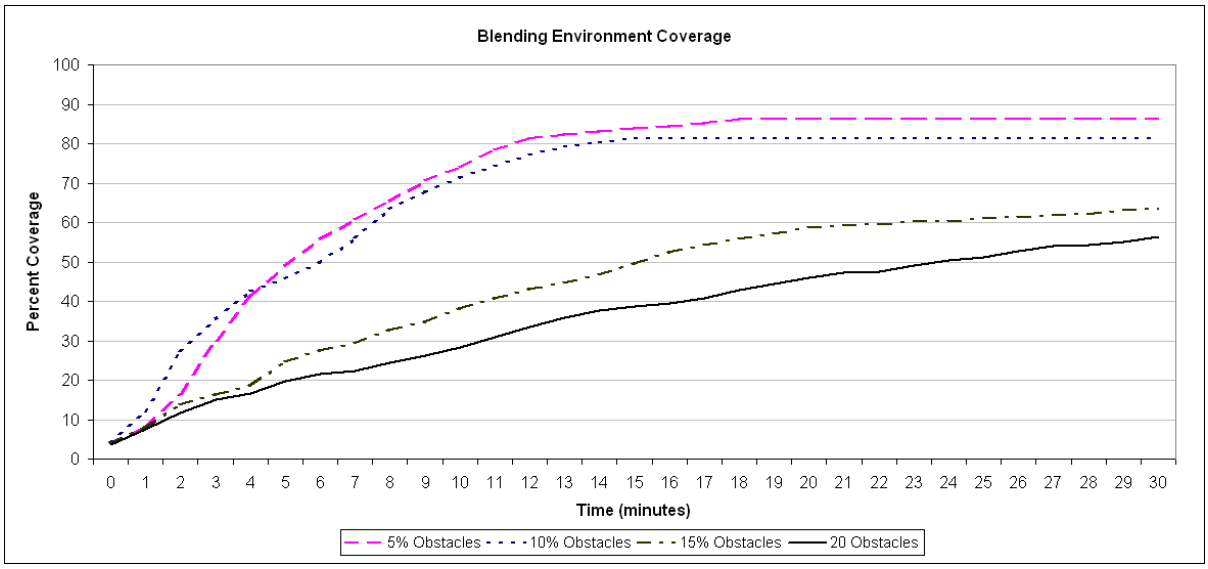

Figure 5.9: Average (n=5) environment coverage achieved by blending agents in 5%, 10%, 15% and 20% obstacle coverage environments.

coverage consistently for trials where the obstacle coverage was 5% and 10%. Often the total environment coverage reached 80% within 15 minutes of the start of the trial. In the 15% and 20% obstacle coverage environments, however, the total environment coverage decreased significantly. Agents were unable to get much more than 60% of the environment covered.

Figure 5.10 shows the average effectiveness of the blending agents with respect to victim identification across all trials. Early in the trials, blending agents located victims as quickly as autonomous agents and quicker than teleoperated agents. In the 5% and 10% obstacle coverage environments, 15 victims were located on average before 15 minutes of the trial passed. In the low obstacle coverage environments (5% to 10% obstacle coverage), the blending agents were very effective at locating victims. As the environment became more complex, the effectiveness of the blending

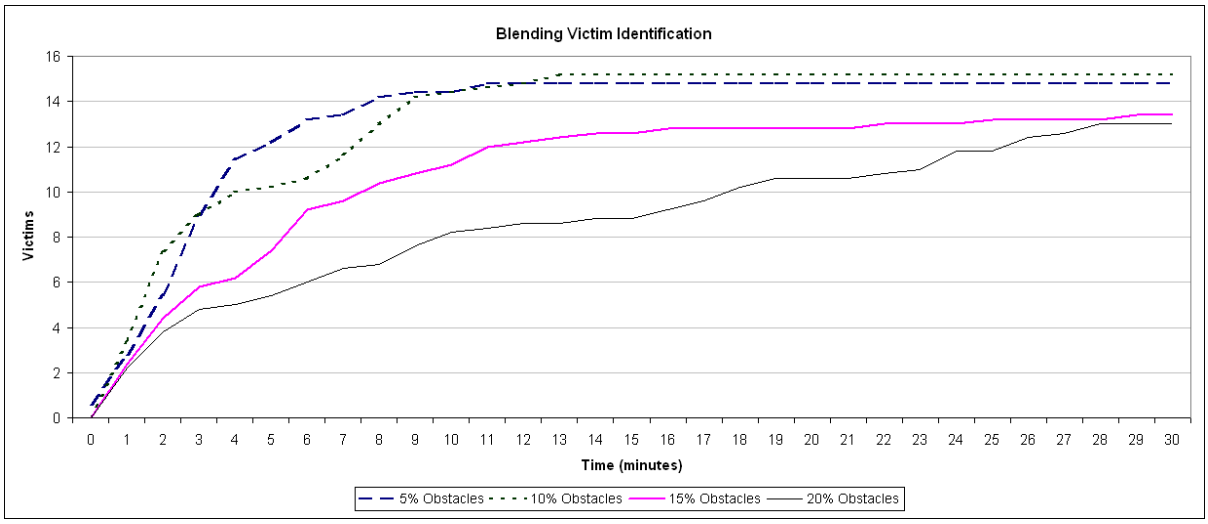

Figure 5.10: Average (n=5) number of victims identified by blending agents in 5%, 10%, 15% and 20% obstacle coverage environments.

agents degraded. In the 15% and 20% obstacle coverage environments victims were not located as quickly as the lower obstacle coverage environments, but by the end of the trial roughly 13 victims were located in all.

Figure 5.11 shows the average amount of time blending agents spent immobile over all trials. In the 5% and 10% obstacle coverage trials, the time blending agents spent immobile was negligible. When the obstacle coverage increased to 15%, the time spent immobile became more significant. The most significant degradation in performance occurred when the obstacle coverage was 20%. Agents spent upwards of 2 million milliseconds immobile. However, this is still roughly a third less than that of autonomous agents under the same conditions. Section 5.7 will present complete performance comparisons between approaches.

Figure 5.12 shows the average number of interactions required between the op-

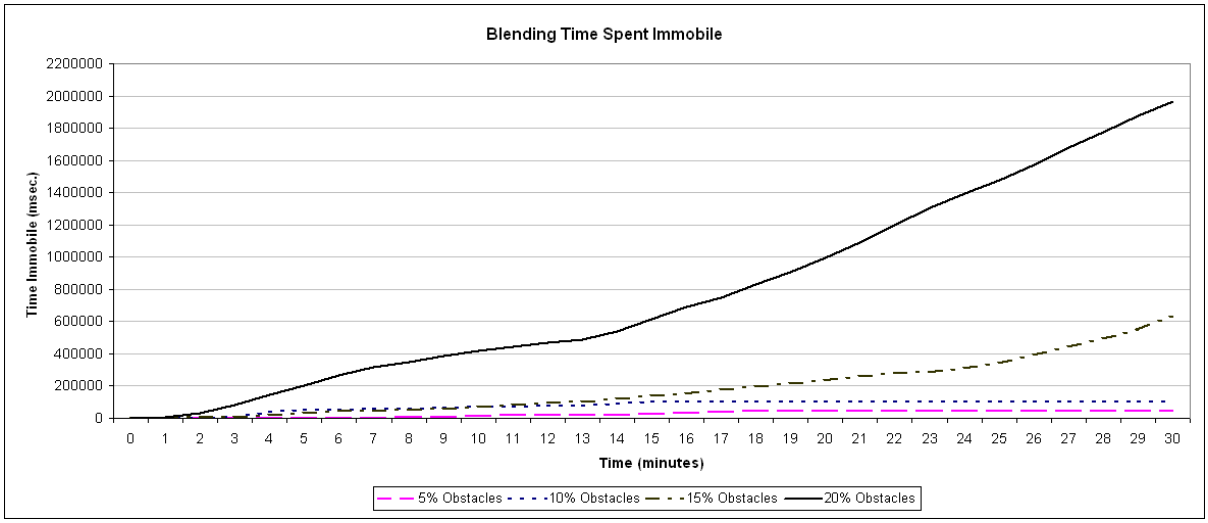

Figure 5.11: Average (n=5) time blending agents spent immobile in 5%, 10%, 15% and 20% obstacle coverage environments.

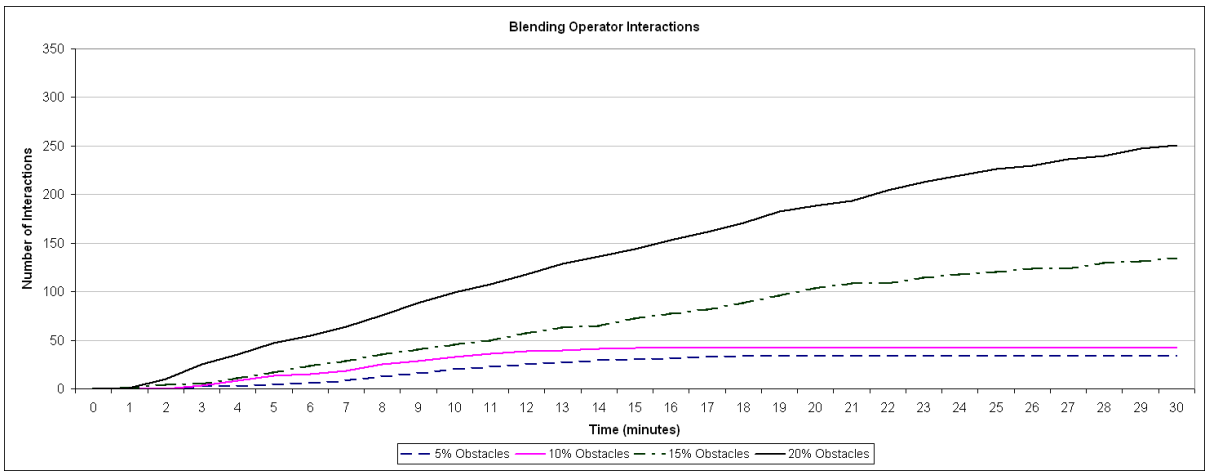

Figure 5.12: Number of interactions operators had with blending agents in 5%, 10%, 15% and 20% obstacle coverage environments.

erator and the agents to achieve the effectiveness described above with respect to the total environment coverage and the number of victims identified across all trials. In the 5% and 10% obstacle coverage environments, the number of operator interactions was extremely low, due somewhat to the speed with which the blending agents reached their target environment coverage. Interactions increased slowly from 0 to less than 50 in the first 15 minutes of the trials. After 15 minutes, the agents already reached their target environment coverage, so the interactions did not increase further. The 15% obstacle coverage environments required more operator interactions, since the agents did not reach their target environment coverage before 30 minutes. The rate that the operator interactions increased was slightly faster than the 5% and 10% environments. By the end of the trial, the blending agents required just under 150 operator interactions on average for the 15% obstacle coverage environments, indicating the number of operator interactions increasing at a rate of roughly 5 interactions per minute. The 20% obstacle coverage environments required many more interactions than the other environments, increasing at a rate of roughly 9.5 interactions per minute. By the end of the trials, the 20% obstacle coverage environments required an average of almost twice as many operator interactions as the 15% obstacle coverage environments.

5.7 Analysis

The results of the trials presented here provide evidence for the superiority of a control system that blends autonomy and teleoperation over either autonomy or teleoperation alone according to four measures of efficiency for robots operating in USAR: environment coverage, victim identification, operator interaction and time spent immobile.

Blending agents performed significantly better than autonomous agents in terms of area coverage, covering more area at a faster rate (see Figure 5.13). I attribute the performance to a human operator's ability to recognize unexplored areas of the environment quickly and guiding agents to unexplored areas more efficiently then the autonomous control system could. Some unexplored areas were unlikely to be found by the autonomous agents because of the unique obstacle configurations in those unexplored areas. That is, the obstacles may have been arranged such that the autonomous agent would have to move through a narrow gap and the agent's *avoid-static-obstacles* motor schema may not allow the agent to get close enough to obstacles to allow it access to that gap. The teleoperated agents performed slightly better than the blending agents, since although the operator could guide blending agents into unexplored areas, once an agent was neglected (i.e. the operator shifts his/her attention to another agent) the autonomous portion of the blending control system may guide the robot back to an explored area. The teleoperated agents only move when the operator commands them. In complex environments, the autonomous portion of the blending agents was more likely to hinder the agents if left unchecked

for too long, contributing to poorer obstacle coverage for blending agents.

With respect to victim identification, the blending of autonomy and teleoperation in the blending agents gave them an advantage over both the teleoperated agents and the autonomous agents (see Figure 5.14). Blending agents were able to take advantage of the autonomous control system to find victims quickly in the first portion of the trials. At least a few victims were easier to find, and the autonomous control portion would find those victims early. Therefore, early in the trials the operator was required to give the blending agents very little attention. Later on in the trials, when the victims in the open were all located, the blending agents performed better then the autonomous agents. This was because the operator could guide the agents through the more difficult areas of the environment, encouraging the agents to cover more area and discover more victims.

Even though the teleoperated agents often performed equally to the blending agents with respect to the number of victims found, the teleoperated agents required more attention, which is made obvious by the number of operator interactions required by the teleoperated agents in comparison to the blending agents (see Figure 5.15). To achieve comparable performance, teleoperated agents required many more operator interactions than did blending agents (see Figure 5.15). The autonomous control portion of the blending agents enabled the agents to perform the most simple portions of the USAR task themselves. In the early portions of the blending trials, the autonomous agents required little attention to perform well in exploring the area around them. Later in the trials, the agents required guidance, since they were unable get

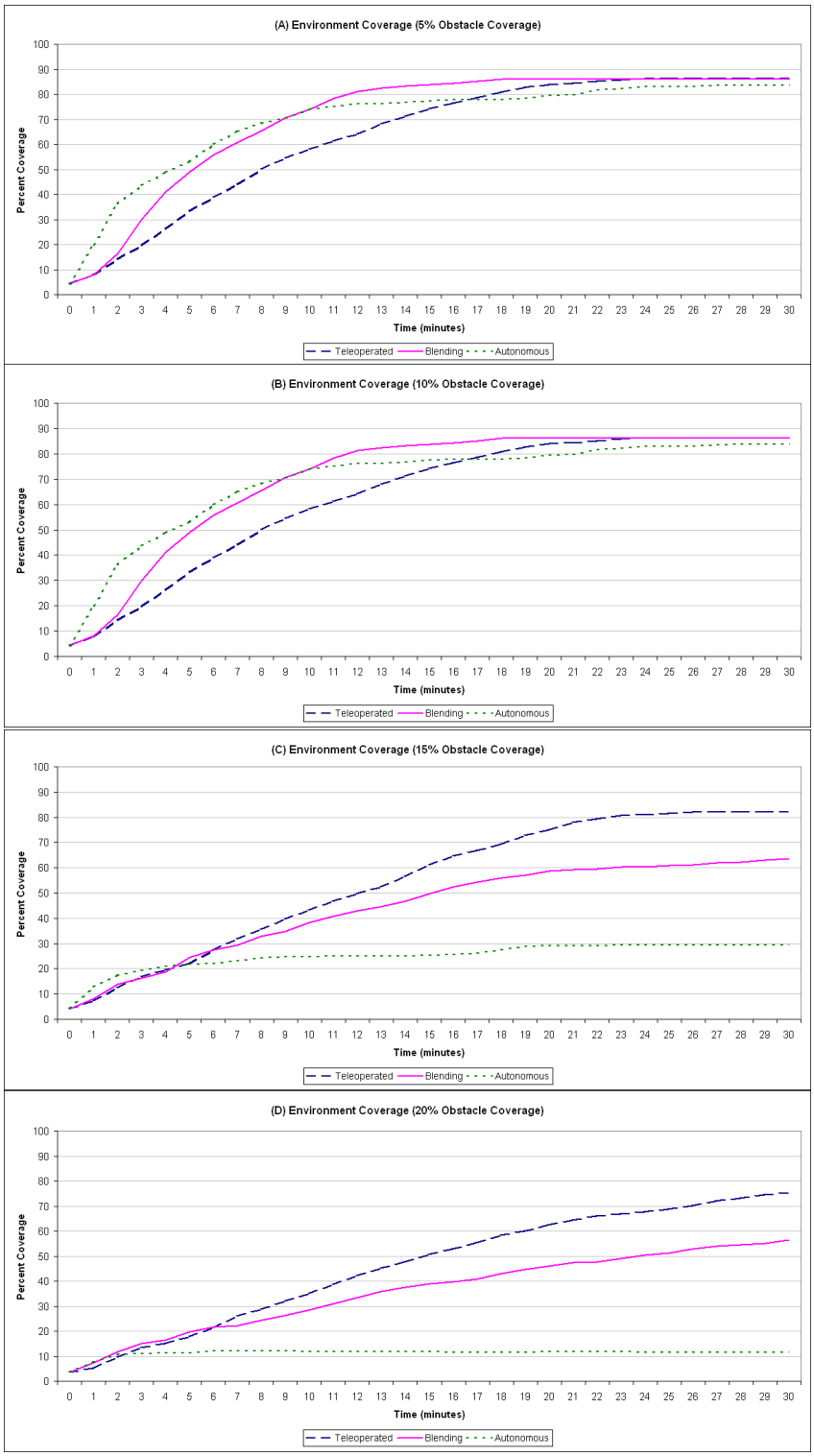

Figure 5.13: Comparison of environment coverage in teleoperated, autonomous, and blending experiments. All results are averages over 5 trials.

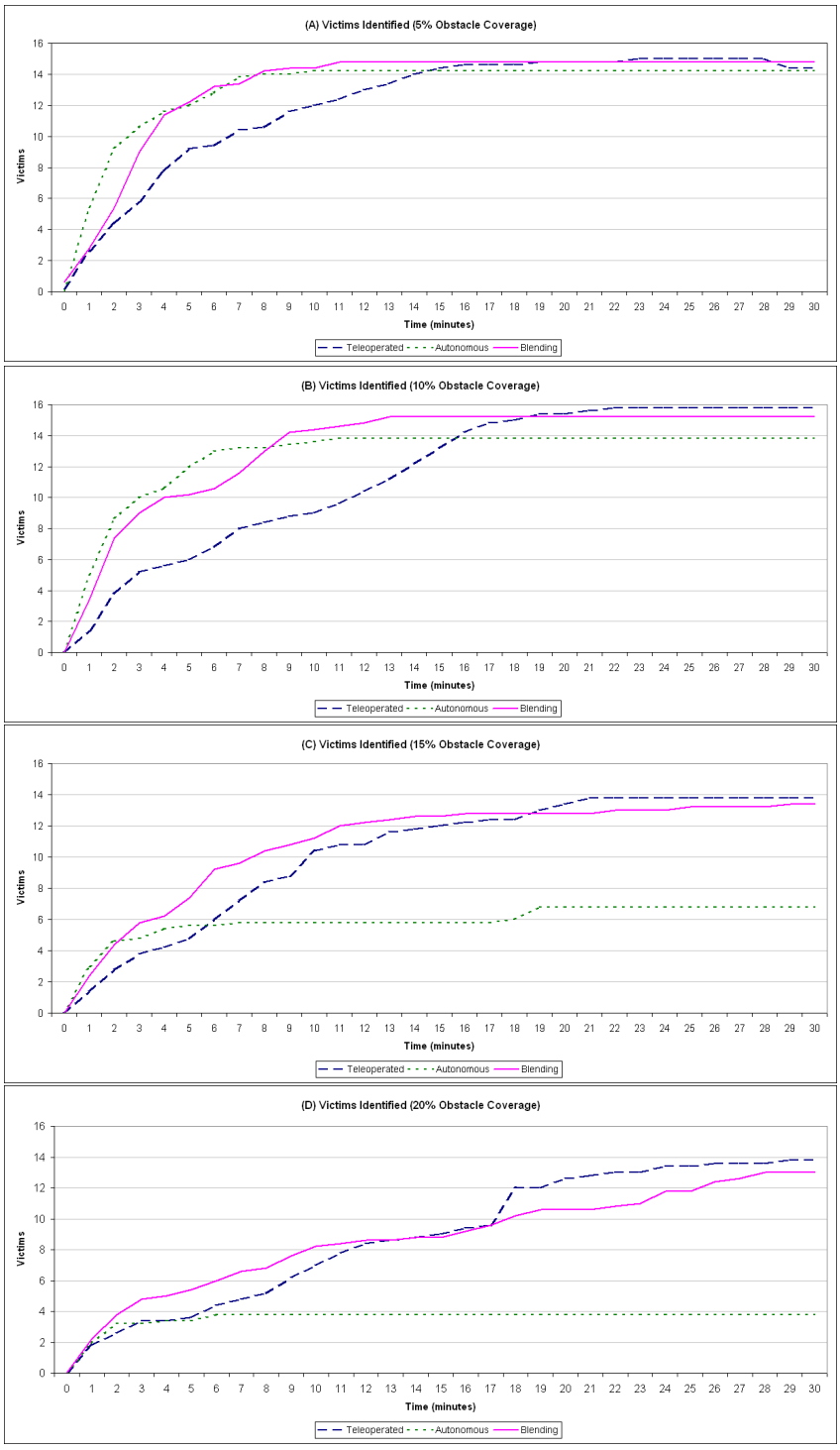

Figure 5.14: Comparison of number of victims identified in teleoperated, autonomous, and blending experiments. All results are averages over 5 trials.

to certain portions of the environment autonomously. These are important findings, since they highlight the increased productivity achievable by a simple autonomous agent given some intervention in the form external commands from an operator.

Throughout all trials performed, the teleoperated agents required many more interactions to complete their task. In trials where the obstacle coverage was 5%, the teleoperated control system needed an average of 5 times more interactions than the blending control system. As the obstacle coverage increased, both control systems required more interactions. The blending control system required fewer interactions in the more complex environments than the teleoperated control system (see Figure 5.15, graphs C and D). The ratio gradually decreases from 5, to 4.8, to 2.5, and finally to roughly 1.5 times the number of interactions. The more dense environments required more supervision from the operator, leading to more agent-operator interactions. Even with the additional attention required by the more dense environments, the blending control system required less attention from the operator, which contributed to a lower cognitive load.

The time each agent spent immobile with respect to autonomous versus blending agents is another indication of the gains associated with blending autonomy and teleoperation. Since the autonomous agents are behavior-based, they are susceptible to local minima (described in Section 5.3.3), often becoming stuck in difficult environments. When agents got stuck in autonomous trials, they would often remain stuck. In the blending trials, if an agent became stuck, the operator was often able to free the agent. Since the operator was notified by the intervention recognition system

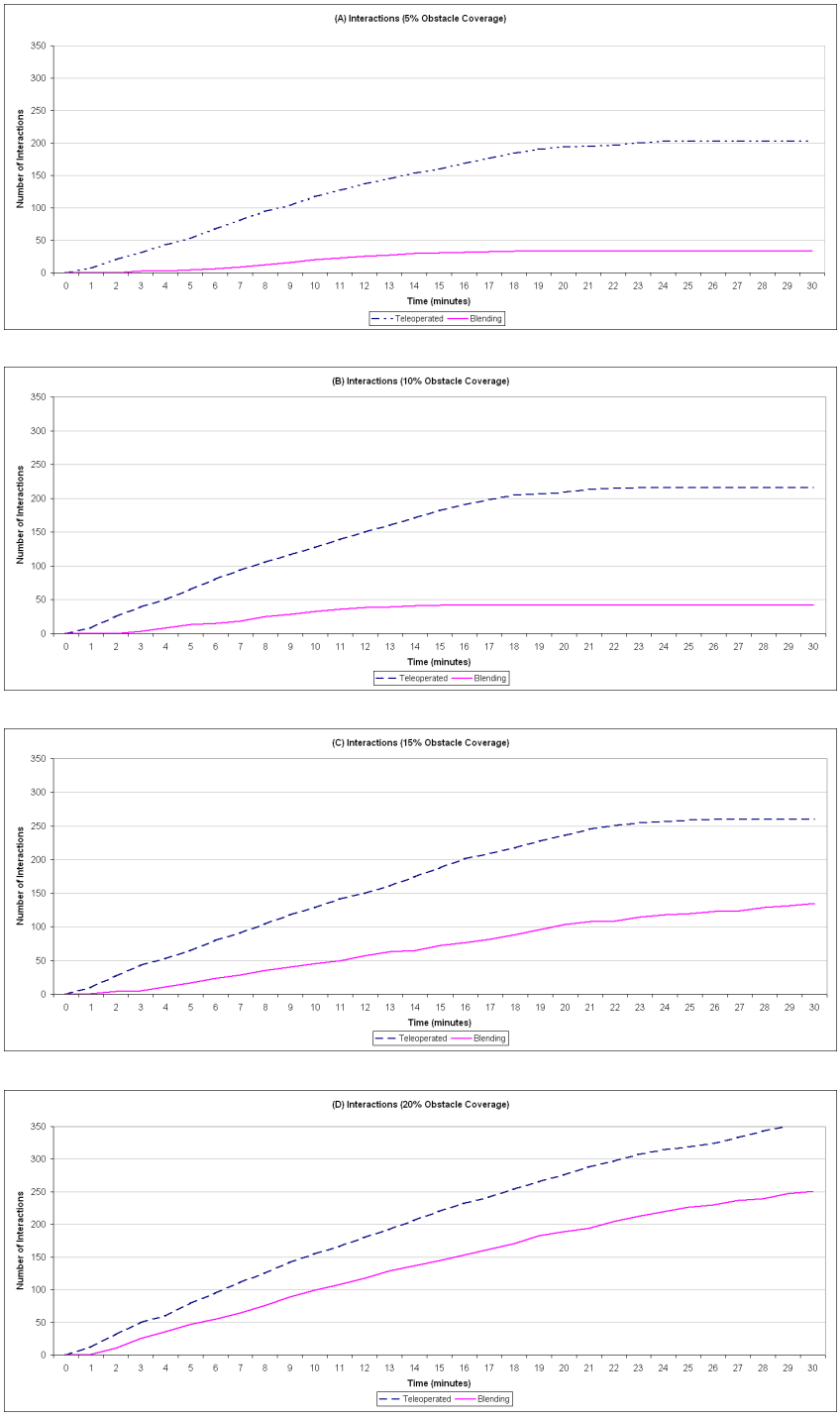

Figure 5.15: Comparison of agent-operator interactions in teleoperated and blending experiments. All results are average over 5 trials.

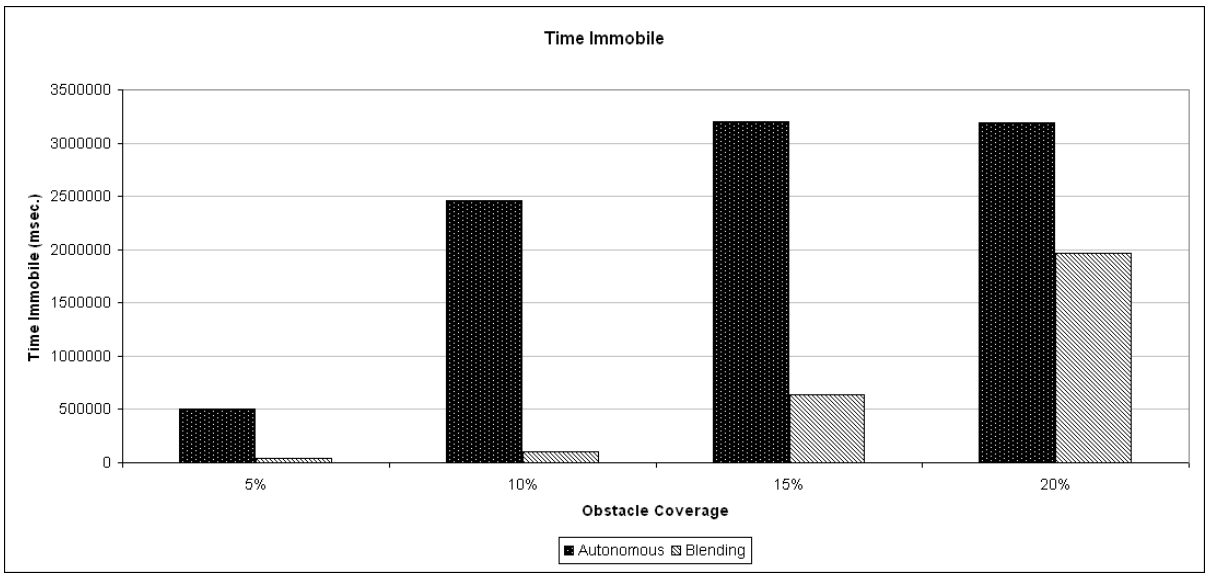

Figure 5.16: Average time in milliseconds spent immobile by environment difficulty, for blending and autonomous agents.

whenever an agent became stuck, the operator was often able to free the agent in a timely manner, reducing the amount of time any particular blending agent spent immobile. In the lower obstacle coverage trials (5% and 10% obstacle coverage), agents became stuck less overall. Moreover, when agents did get stuck, they tended to get stuck less severely, and therefore it was easy for the operator to get the agent mobile again. In trials with higher obstacle coverage, the agents would get themselves stuck in much more complex ways, making it more difficult for operators to release them. In trials where the obstacle coverage was 20%, the time spent stuck for the blending control system was much higher, since agents were often difficult to get mobile, leading to agents being abandoned. Blending operator instructions with the autonomous instructions contributes to a significant increase in effectiveness for agents, which can be observed by comparing the results of the autonomous trials and the blending trials.

5.8 Summary

The experiment described in this chapter demonstrates that the blending of teleoperation with existing autonomous agents can significantly improve the effectiveness of agents. It is also evident that the blending of autonomy and teleoperation does reduce the number of interactions between the operator and the agent, while still maintaining a comparable level of performance. Finally, the experiment demonstrates a practical balance between autonomy and teleoperation in complex dynamic environments. The next chapter will describe how this research as a whole addresses the research questions and discuss future work related to this research.

Chapter 6

Findings and Recommendations

6.1 Overview

This research was designed to answer the research questions in Chapter 1. However, in attempting to answer those questions I have also become aware of a wide variety of research that relates not only directly to blending the intentions of human operators with autonomous robots, but also related to the limitations of autonomous control, the difficulties related to applying the techniques described here in a physical domain, the scope of teleautonomy, and much more. In this chapter I will begin by discussing my findings and relating those findings back to the research questions in Chapter 1. Next, I will present the contributions that this work offers to the research community. Finally, I will discuss future work related to this research.

6.2 Findings and Analysis

Chapter 1 presented several research questions around which the work in this thesis is focused. I will reiterate those questions here and discuss the results in my experiment in light of these questions.

Research Question 1

Will the addition of teleoperation to autonomous agents increase their overall performance?

This research shows that agents designed to perform in a USAR domain perform better when there is some degree of intervention on the part of an operator. This intervention reflects the limitations of the autonomous agents. Where an autonomous agent is unable to complete a task on its own, the operator may provide insight and direction to the agent in order to guide it into a more advantageous position. It has also been shown that the intervention does not have to be in the form of unquestioned low level instructions: instead, more abstract levels of control can be blended with the agent's desires, resulting in a shared control environment where the desires of both the agent and the operator are blended.

However, there are several limitations to my approach that are worth mentioning. First, the autonomous agents designed for this research were very basic behavior-based agents. Since my focus was on studying the relationship between the human operator and autonomous agents, and seeing as I built the autonomous agent architecture based solely on theory, the agents used here do not reflect the full capabilities of

autonomous agents in general. By devoting more time to the construction of the autonomous architecture the autonomous agent performance may have been better. Beyond the results obtained here, however, one must also consider the state of the art demonstrated in USAR test beds today. In light of the fact that there are extremely few entrants into USAR competitions to date that are autonomous, it is not an exaggeration to say that it will be some time to come before the capabilities of purely autonomous agents allow success in a USAR domain.

Also, it is important to note that this research was carried out in simulation on a specific task, USAR. These results may not reflect results in entirely different domains. Despite the fact that the simulation tool used has been physically validated, there will undoubtedly be differences in a physical implementation of this domain from the simulation I have used. Physics in a simulator is never perfect, and my experiment also assume the ability to identify victims reasonably well - an orthogonal task to the concept of teleautonomy itself, but a very important one to the successful performance in USAR. These assumptions considered, however, I believe my results are still an indication that autonomous agents (given today's state of the art) in areas similar to this one will also benefit from the introduction and blending of human operator instructions.

The last issue I wish to note is the nature of the teleoperated instructions being blended. The focus of this research is on the design of an approach to teleautonomy and the performance of that design in a particular environment. I have not concentrated on human-computer interaction issues, in particular the ease with which one

individual as opposed to another can interact with such systems. Like any other talent, the ability to visualize the situation of a robot remotely (situational awareness) will vary from person to person, as will the amount of operator fatigue and other particular criteria related to the successful control of a group of mobile robots. In this research, I served as the sole human operator, and the results are thus to be interpreted in light of my abilities as an operator, which I believe are average. It would have been interesting, however, to see if having a variety of different human operators would have affected the results, and attempt to identify and quantify the affects of particular human skills on the results of teleoperation. These are all issues that are mentioned further in Section 6.4. Additionally, in general, the teleoperated instructions should not be limited to a human source. Since other agents have their own perspectives on the environment, it would be useful to attempt to relieve operator fatigue by allowing other agents to have input on a sliding scale to the same degree that human operators can in this approach. Again, this relates to future work that will be discussed later in this chapter.

Research Question 2

Can the introduction of autonomy reduce the number of interactions required between the agent and the operator while maintaining a comparable overall performance?

The experiment in Chapter 5 demonstrates that the addition of autonomy does reduce the number of interactions required between the agent and the operator. The reduction in the number of interactions indicates a decrease in the cognitive load of

the human operating the agents. A reduction in cognitive load is significant, since if a human can operate a single agent requiring only small amounts of intermittent attention, then presumably the operator could also control multiple agents without becoming overloaded. This research demonstrates a single operator operating three agents and requiring few interactions, compared to controlling three purely teleoperated agents.

There are some major assumptions required to support the answer to question 2. First, the simulated environment attempts to approximate the cognitive load that a human operator would expect in the real world USAR. Murphy et al. [2000a], however, note that real world USAR is more difficult for a human operator to perform than the testbeds used in USAR competitions. A simulator would presumably be significantly easier on a human, given solid obstacles rather than piles of random visually distracting debris, and a much more three-dimensional environment. There are numerous other differences in the simulator I have used: obstacles cannot be moved by the agent, and the obstacles representing debris also cannot be damaged. In the real world, a robot can get snagged on debris and drag it away, potentially causing further damage in the environment. As such, human operators in real USAR would likely perform much worse than the human operators in simulated USAR. So would autonomous agents, however, and given the state of the art in AI, this further emphasizes the need for human support and a teleautonomous mode of operation. Given all this, I still believe that these results strongly suggest that cognitive load can be reduced through the introduction of autonomy in agents, and I wish to validate

this theory on real robots in the physical USAR domain, which is discussed more in Section 6.4.

Beyond the issue of simulation, the measurement chosen for operator interaction should also be mentioned. The number of interactions with the robots is used in my experiment as a means to approximate the cognitive load on a human operator. More operator-agent interactions is indicative of higher amount of cognitive load. This measurement, however, does not take into account the amount of attention that the operator must pay to agents even when he or she is not directly interacting with an agent. An operator may be focusing all of his or her attention on observing a particular agent without having to send any instructions at all. Thus, while quantifying interactions is a useful measure of operator load, it is not the sole measure. Experiments using a variety of human operators could supplement this measure by videotaping the subject and attempting to note periods of attention during which agents were not being physically given instructions.

Finally, these findings are most relevant to operating in the USAR domain. Although in domains similar to USAR, operator-agent interactions will likely be impacted in a similar fashion by the introduction of autonomy to agents, the less similar the task is to USAR the less applicable these findings may be.

Subsidiary Question

Can an infrastructure be designed to support a practical balance between autonomy and teleoperation in complex dynamic environments?

The blending control system designed for this research supports a practical balance between autonomy and teleoperation. The blending control system enables the operator to interact with the agents at various levels without necessarily exerting direct control over the agent. One of the most important features of this research is the support of a balance, where the operator and the agent cooperate so that together they achieve higher effectiveness than either would alone. The effectiveness is attained by increasing the capabilities of the agent through the addition of external instructions, and by reducing the amount of cognitive load on operators, increasing the potential number of robots that can be operated, and reducing the onset of fatigue common to operators working in complex dynamic domains.

Supporting a practical balance between autonomy and teleoperation in a general sense is much more difficult than the work done here. The constraints of the simulated USAR domain allowed me to focus on certain aspects of teleautonomy and limit the involvement of the human operator to only three types of interactions: the joystick, the waypoint manager, and manual behavior control. There are a variety of other forms of teleoperation not covered in this work that would be desirable to have in a system that balances autonomy and teleoperation. The practicality of some of the methods of control require further validation in the real world, where the infrastructure may be much more difficult to design.

The work done here is only preliminary to the ultimate goal of my research, which is to have an infrastructure that supports a practical balance between autonomy and teleoperation in a variety of real world domains.

6.3 Contributions

This research is of interest to many research communities, including mobile robotics, teleautonomous robotics, user interfaces and distributed artificial intelligence. The contributions include:

1. The development and implementation of an approach to balancing autonomy and teleoperation in a complex domain.

2. An approach that has benefits over previous approaches and which is demonstrable in an approximation of a physical domain.

3. A methodology that allows agents with a limited degree of autonomy to be deployed in a domain that is far beyond the approaches currently used in autonomous systems.

4. A methodology that allows a human operator the ability to do more by distributing their control feasibly over a number of robots with minimal supervision for each.

5. Encouragement of further work in multi-agent systems in this area by keeping the costs of individual agents low, allowing the potential for larger numbers of them to be deployed.

The blending approach that I developed has several advantages over both autonomous approaches and teleoperated approaches deployed in similar environments.

The blending approach has more efficient victim identification over autonomous approaches due to operator intervention. Another advantage over strictly autonomous approaches is the ability of operators to help robots who have become stuck. A third advantage is the direction that operators can provide to reduce the amount of time agents spend doing pointless work. Additionally, the blending approach provides a reduction in cognitive load required in strictly teleoperated approaches through the automation of some of the tasks, and a further reduction in operator load through waypoint management. Finally, the approach supports the ability of the operator to adjust the behavior of agents on the fly to help the autonomous control to cope with specific situations.

6.4 Future Work

There are a number of directions that future work in this area can profitably take. One of the most obvious extensions to this work is the application of the blending control system on physical robots. Since this work was done using the Player/Stage application suite, all code written to control the simulated stage agents is directly compatible with physical Pioneer mobile robot platforms. However, the code used in this thesis was not verified on a set of physical robots. Extending the blending control system to work with other mobile robot platforms is another goal of future work in this area. There are several issues that have to be addressed if this system is going to be applied to physical robots. First, on real robots, perfect localization is no longer a

simple assumption. Odometry on real robots is likely to have at least some noise, and that noise will be cumulative. The application of vision and other sensor technology would have to be employed in order to have useful localization. Another assumption that has to be dealt with is the increase in sensor noise and environment complexity. Two sensors in particular will present the most challenging problems: vision and the laser range finder. Recognizing objects using vision will be a very significant problem in a real robot compared to a simulated one, and thus a more sophisticated method for handling errors will have to be developed. The laser range finder, which is free in simulation, would be expensive to purchase in the real world. The approach and much of the code written for the simulated blending system will be applicable on the physical robots, but the underlying infrastructure will require much additional work.

This research focuses on blending human operator instructions with agent desires. However, the blending control system was designed so that it could be extended to include instructions from peer agents instead of human operators. Research into how agents could send instructions to one another, and how those instructions could be blended with the intentions of the receiving agent could further reduce the cognitive load of human operators by allowing agents to be more helpful to one another.

The blending system described in this research decreases the amount of risk to the agent, but a persistent operator is still able to greatly effect the agent's resulting actions. If the agent could model the operator and develop a level of trust, the decision of whether commands are blended and to what degree could be influenced by how much trust the agent has with a particular operator. Consider an inexperienced

operator who unintentionally persists in instructing the agent to perform very risky tasks that are entirely avoidable, such as moving too close to unstable ground. If the agent "knew" that the operator was inexperienced, the agent might blend the inexperienced operator's instructions differently. Imagine now that instead of an inexperienced operator, a malicious operator may try to take advantage of agents and hinder their progress for their own means. In either situation, whether the operator is inexperienced or malicious, the ability to model operators may have advantages. This type of research could blossom into multi-agent research, where agents keep track of operator reputations and share their reputations to benefit all agents in a particular society.

Again, since I was the sole human operator for this research, there was little feedback from other potential operators. I would have like to perform a complete usability study for this control system. The usability study would involve recruiting a pool of subjects to use as operators, classified by experience with tools such as robots or remote-controlled vehicles (it would even be possible to employ a psychological spatial-visualization test as a tool for categorizing subjects by their skills at visualizing remote situations). Additional criteria for measuring cognitive load would also have to be devised to supplement to the heuristic measurement used in this research. A usability study could provide a large amount of insight into the relationship between the human operator and the agent. This could lead to improvements in blending by taking advantage of some of the natural ways that human operator interact with agents.

Another direction for this research is user interface development. A well designed user interface can encourage easier interaction between the operator and the agents. I would like to devote more time to designing a user interface that is intuitive for a variety of operators. The user interface employed here is sufficient for this research, but it could be improved to be more operator friendly. I would like to explore ways of improving the interactions between agents and human operators, such as introducing a more immersive user interface that allows the operator to feel more "in control". Traditional user interfaces often seem passive, where the operator is watching the agent perform instead of being immersed with the agent in the environment.

6.5 Conclusion

It is my hope that the success of this research encourages more research in blending autonomy and teleoperation so that its benefits can be applied to a number of domains where simple agents can perform the task only to a limited degree. I believe that agents will be unable to perform complex tasks autonomously in the near future, requiring aid from human operators to improve their effectiveness. This research has demonstrated that blending operator instructions with the autonomous desires of agents can benefit both human operators and agents alike, increasing effectiveness in a complex and dynamic environment. If agents are not going to be capable of performing complex tasks in the near future, then the interactions of human operators and robotic agents will only become more important as a subject for future research.

Bibliography

Philip Agre and David Chapman. What are plans for? In P. Maes, editor, *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, pages 17–34. MIT Press, March 1991.

Khaled S. Ali. *Muliagent Telerobotics: Matching Systems to Tasks*. PhD thesis, Georgia Tech, 1999. Computer Science and Engineering.

Khaled S. Ali and Ronald C. Arkin. Multiagent teleautonomous behavior control. *Machine Intelligence and Robotic Control*, 1(2):3–10, 2000.

John Anderson and Alfred Wurr. Dimensions of teleautonomy in mobile agents. In H Lueng, editor, *Proceedings of Artificial Intelligence and Soft Computing*, number 6, pages 1–6, Banff, Canada, July 2002. The International Association of Science and Technology for Development.

Ronald C. Arkin. Motor schema-based mobile robot navigation. *International Journal of Robotics Research*, 8(4):92–112, 1989.

Ronald C. Arkin. Behavior-based robot navigation for extended domains. *Adaptive Behavior*, 1(2):201–225, 1992.

Ronald C. Arkin. *Behavior-Based Robotics*. Cambridge: MIT Press, Cambridge, MA, 1998.

Ronald C. Arkin and Khaled S. Ali. Integration of reactive and telerobotic control in multiagent robotic systems. In *Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, pages 473–478, Brighton, England, August 1994.

Ronald C. Arkin and Tucker Balch. AuRa: Principles and practice in review. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(2):175–189, 1997.

Ronald C. Arkin and Tucker Balch. *Cooperative Multiagent Robotic Systems*, pages 278–296. Cambridge: MIT Press, Atlanta, Georgia, 1998.

Ronald C. Arkin, Thomas R. Collins, and Yoichiro Endo. Tactical mobile robot mission specification and execution. In *Proceedings of Mobile Robots XIV*, volume 3838, pages 150–163, Boston, Massachusetts, September 1999. International Society for Optical Engineering.

Tucker Balch. *Behavioral Diversity in Learning Robot Teams*. PhD thesis, Georgia Institute of Technology, December 1998a.

Tucker Balch. Integrating rl and behavior-based control for soccer. In *RoboCup-97:*

Proceedings of the First Robot World Cup Soccer Games and Conferences, Springer-Verlag, 1998b.

Tucker Balch and Ronald C. Arkin. Motor schema-based formation control for multiagent robot teams. In *Proceedings of the 1995 International Conference on Multiagent Systems*, pages 10–16, San Francisco, CA, 1995.

Jacky Baltes and John Anderson. A pragmatic approach to robotic rescue: The keystone fire brigade. In *Proceedings of the AAAI Mobile Robot Competition and Exhibition Workshop*, pages 38–43, Edmonton, Alberta, July 2002. AAAI.

Maxim Batalin. Java client for player/stage. Webpage, July 2003. http://www-robotics.usc.edu/ maxim/JavaClient/jc.htm.

Darrin C. Bentivegna, Khaled S. Ali, Ronald C. Arkin, and Tucker Balch. Design and implementation of a teleautonomous hummer. In *Proceedings of Mobile Robots XII*, pages 130–138, Pittsburgh, PA, October 1997. International Society for Optical Engineering.

Jeffrey M. Bradshaw. Introduction to software agents. In Jeffrey M. Bradshaw, editor, *Software Agents*, pages 3–46. The MIT Press, Menlo Park, CA, 1997.

Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, March 1986.

Rodney A. Brooks. A robot that walks: Emergent behaviors from a carefully evolved

network. In *Proceedings of the International Conference on Robotics and Automation*, pages 692–694. IEEE, May 1989.

Rodney A. Brooks. Elephants don't play chess. *Robotics and Autonomous Systems*, 6(1&2):3 – 15, June 1990.

Rodney A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47: 139–159, 1991a.

Rodney A. Brooks. New approaches to robotics. *Science*, 253:1227–1232, September 1991b.

Rodney A. Brooks, Cynthia Breazeal, Matthew Marjanovic, Brian Scassellati, and Matthew Williamson. *The Cog Project: Building a Humanoid Robot*, pages 52–87. Springer, New York, 1999.

Barry Brumitt, Anothony Stentz, Martial Hebert, and CMU UGV Group. Autonomous driving with concurrent goals and multiple vehicles: Experiments and mobility components. *Autonomous Robots*, 12(2):135–156, March 2002.

J. Cameron, D. MacKenzie, K. Ward, R. Arkin, and W. Book. Reactive control for mobile manipulation. In *Proceedings of the International Conference on Robotics and Automation*, pages 228–235, Atlanta, GA, 1993.

Yuni Cao, Tsu-Wei Chen, Martin D. Harris, Andrew B. Kahng, Anthony M. Lewis, and Andre Stechert. A remote robotics laboratory on the internet. In *Proceedings of the International Networking Conference 1995*, pages 64–72, Honolulu, June 1995.

Jennifer Casper. Human-robot interactions during the robot-assisted urban search and rescue response at the world trade center. Master's thesis, University of South Florida, April 2002. Computer Science and Engineering.

Jennifer Casper, Mark Micire, Jeff Hyams, and Robin R. Murphy. A case study of how mobile robot competitions promote future research. In *Proceedings of Robocup 2001*, pages 123–132, 2001.

Jennifer Casper and Robin Murphy. Workflow study on human-robot interaction in USAR. In *Proceedings of the International Conference on Robotics and Automation*, volume 2, pages 1997–2003, Washington, May 2002. IEEE.

Jennifer Casper, Robin Murphy, Mark Micire, and Jeff Hyams. Mixed-initiative control of multiple heterogeneous robots for urban search and rescue. Technical report, University of South Florida, 2000.

David Chapman. Penguins can make cake. *AI Magazine*, 10(4):45–50, 1989.

S. Coradeschi and A. Saffiotti. Anchoring symbols to sensor data: preliminary report. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI)*, pages 129–135, Menlo Park, CA, 2000. AAAI Press.

Jacob W. Crandall and Michael A. Goodrich. Experiments in adjustable autonomy. In *Proceedings of the International Conference of Systems, Man, and Cybernetics*, volume 3, pages 1624–1629. IEEE, 2001.

Tom Duckett and Alessandro Saffiotti. Building globally consistent gridmaps from topologies. In *Proceedings of the 6th Annual Symposium on Robot Control (SYROCO)*, pages 357–361, Vienna, Austria, September 2000. International Federation of Automatic Control.

E. Gat. Integrating planning and reaction in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI)*, pages 809–815, San Jose, CA, July 1992.

Brian P. Gerkey, Richard T. Vaughan, and Andrew Howard. *Player Version 1.3.2 User Manual*. Robotics Research Laboratory, University of Southern California, Los Angeles, CA, May 2003.

Brian P. Gerkey, Richard T. Vaughan, Kasper Stoy, Andrew Howard, Gaurav S. Sukhatme, and Maja J. Mataric. Most valuable player: A robot device server for distributed control. In *Proceedings of Intelligent Robots and Systems*, pages 1226–1231, Wailea, Hawaii, October 2001. IEEE.

Ken Goldberg. *The Robot in the Garden*. Cambridge: MIT Press, Massachusetts Institute of Technology, 2000.

Sean Graves and Richard Volz. Action selection in teleautonomous systems. In *Proceedings of the International Conference on Intelligent Robots and Systems*, pages 14–19. IEEE, 1995.

Stevan Harnad. The symbol grounding problem. *Physica D*, 42:335–346, 1990.

I. Horswill. Polly, a vision-based artificial agent. In *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI)*, pages 824–829, Washington, DC, July 1993. AAAI.

Wesley H. Huang and Eric P. Krotkov. Optimal stereo mast configuration for mobile robots. In *Proceedings of the International Conference on Robotics and Automation*, Albuquerque, NM, 1997. IEEE.

Adam Jacoff, Elena Messina, and John Evans. Experiences in deploying test arenas for autonomous mobile robots. In *Performance Metrics for Intelligent Systems Workshop*, Mexico City, Mexico, September 2001a. IEEE.

Adam Jacoff, Elena Messina, and Brian Weiss. Reference test arenas for autonomous mobile robots. In *The 14th International FLAIRS Conference*, Key West, Florida, May 2001b. AAAI.

O. Khatib. Real-time obstacle avoidance using harmonic potential functions. In *Proceedings of the Conference on Robotics and Automation*, pages 500–505, St. Louis, MO, 1985. IEEE.

Aaron Khoo, Robin Hunicke, Greg Dunham, Nick Trienens, and Muon Van. Flexbot, groo, patton and hamlet : Research using computer games as a platform. In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI)*, number 14, pages 1002–1003, Edmonton, Alberta, July 2002. AAAI.

David Kortenkamp and Terry Weymouth. Topological mapping for mobile robots using a combination of sonar and vision sensing. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 979–984, Seattle, Washington, 1994. AAAI.

B. Krogh. A generalized potential field approach to obstacle avoidance control. Technical Report 484, Society of Manufacturing Engineers, Dearborn, Michigan, 1984.

M. Krogseter, R. Oppermann, and C. Thomas. A user interface integrating adaptability and adaptivity. In R. Oppermann, editor, *Adaptive User Support: Ergonomic Design of Manually and Automatically Adaptable Software*, pages 97–125. Lawrence Erlbaum Associates, Hillsdale, NJ, 1994.

M. Krogseter and C. Thomas. Adaptivity: System-initiated individualism. In R. Oppermann, editor, *Adaptive User Support: Ergonomic Design of Manually and Automatically Adaptable Software*, pages 67–96. Lawrence Erlbaum Associates, Hillsdale, NJ, 1994.

J. Lee, M. Hubar, E. Durfee, and P. Kenny. Um-prs: An implementation of the procedural reasoning system for multirobot applications. In *Proceedings of the Conference on Intelligent Robotics in Field, Factory, and Space*, pages 842–849, Houston, TX, March 1994.

D. Lyons and A. Hendriks. Planning as incremental adaptation of a reactive system. *Robotics and Autonomous Systems*, 14(4):255–288, 1995.

M. Mataric. Integration of representation into goal driven behavior-based robots. *IEEE Transactions on Robotics and Automation*, 8(3):304–312, June 1992.

M. Mataric. Behavior-based control: Examples from navigation: Examples from navigation, learning, and group behavior. *Journal of Experimental and Theoretical Artificial Intelligence*, (2-3):323–336, 1997.

François Michaud and Serge Caron. Roball, the rolling robot. *Autonomous Robots*, 12(2):211–222, March 2002.

Michael Montemerlo, Joelle Pineau, Nicholas Roy, Sebastian Thrun, and Vandi Verma. Experiences with a mobile robotic guide for the elderly. In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI)*, number 14, pages 587–592, Edmonton, Alberta, July 2002. AAAI.

Robin Murphy, Jennifer Casper, Mark Micire, and Jeff Hyams. Assessment of the NIST standard test bed for urban search and rescue. In *Proceedings of AAAI Mobile Robotics Competition Workshop*, pages 11–16, Austin, TX, July 2000a.

Robin Murphy, Jennifer Casper, Mark Micire, Jeff Hyams, and Brian Minten. Mobility and sensing demands in USAR. In *Proceedings of the Industrial Electronics Society Conference*, Nagoya, Japan, October 2000b. session on rescue engineering.

Robin Murphy and J. Sprouse. Strategies for searching an area with semi-autonomous mobile robots. In *Proceedings of Robotics for Challenging Environments*, pages 15–21, Albuquerque, NM, June 1996. American Society of Civil Engineers.

Robin R. Murphy and David Hershberger. Classifying and recovering from sensing failures in autonomous mobile robots. *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI)*, 2:922–929, 1996.

Allen Newell and Herbert A. Simon. Computer science as empirical inquiry: Symbols and search. *Communications of the ACM*, 12:113–126, 1976.

Nils J. Nilsson. Shakey the robot. Technical Report 323, Artificial Intelligence Center, SRI International, Menlo Park, CA, 1984.

Eclipse Platform Technical Overview. Object Technology International, Inc. IBM Corporation, February 2003.

Stuart Russel and Peter Norvig. *Artificial Intelligence A Modern Approach*, pages 31–50. Alan Apt, Upper Saddle River, New Jersey, 1995.

Linda Strachan, John Anderson, Murray Sneesby, and Mark Evans. Minimalist user modelling in a complex commercial software system. *User Modelling and User-Adapted Interaction*, 10:109–145, 2000.

Andreas L. Symeonidis, Pericles A. Mitkas, and Dionisis D. Kechagias. Mining patterns and rules for improving agent intelligence through an integrated multi-agent platform. In *Proceedings of the IASTED International Conference on AI and Soft Computing*, pages 48–53, Banff, Canada, July 2002.

Mohan Trivedi, Brett Hall, Greg Kogut, and Steve Roche. Web-based teleautonomy and telepresence. In *Proceedings of the 45th Optical Science and Technology Con-*

ference, volume 4120, pages 81–85, San Diego, August 2000. International Society for Optical Engineering.