

**CONSTRAINT-BASED TASK SELECTION AND
CONFIGURATION FOR AUTONOMOUS MOBILE ROBOTS**

by
Richard S. Stansbury
University of Kansas, 2007

M.S. Computer Engineering, 2004
B.S. Computer Engineering, 2002

Submitted to the graduate degree program in Electrical Engineering and
Computer Science and the Faculty of the Graduate School of the University of
Kansas. In partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Dr. Arvin Agah (Chairperson)

Dr. Chris Allen

Dr. Prasad Gogineni

Dr. Carl Leuschen

Dr. David Braaten

Date Defended: _____

The Dissertation Committee for Richard S. Stansbury
certifies that this is the approved version of the following dissertation:

**CONSTRAINT-BASED TASK SELECTION AND
CONFIGURATION FOR AUTONOMOUS MOBILE ROBOTS**

Committee:

Dr. Arvin Agah (Chairperson)

Dr. Chris Allen

Dr. Prasad Gogineni

Dr. Carl Leuschen

Dr. David Braaten

Date Approved: _____

ABSTRACT

An autonomous mobile robot must be capable of rationally selecting its tasks in order to achieve some state or goal. Rule-based systems encode system knowledge into a set of rules that guide the robot. The dynamic nature of the world is seldom consistent enough to support the rigidity of rules. A new decision maker is needed that expresses the problem as intuitively as rule-based systems with flexibility, extensibility and generalizability.

The constraint-based methods can be used to model the problem of selecting and configuring tasks for mobile robots. Constraint Satisfaction Problems (CSPs) provides a framework in which multiple conflicting constraints upon the robot can be resolved in such a way that the robot not only performs correctly, but also meets or exceeds its performance requirements. Constraints also provide the intuitive means of specifying the decision model without the rigidity of rules. Performance objectives are incorporated into the constraint model so that the decision-making system is capable of rationally guiding the robot through actions that best meet its current needs and goals.

In this dissertation, a framework and decision maker for robot task selection and configuration is developed. Tasks under this framework are modeled as constraint satisfaction problems. A common software interface was used in order to support the uniform composition of the CSP models. The solution to the CSP

provides the selection of a task and its configuration. A utility function is used to select a single solution, if multiple solutions are generated. The framework is demonstrated for three unique robot scenarios: a delivery robot, a polar robot, and an urban search and rescue robot.

The delivery robot scenario models tasks for a mobile robot responsible for delivery of items within an office environment. It provides an initial proof-of-concept. Constraint models are constructed for a charge task, a pickup item task, and a deliver item task. The solution to the CSP configures the task execution by specifying the satisfying speed and path for the robot to follow. The performance of the system given the task load, which is directly proportional to the size of the CSP model, is evaluated. Results show a significant increase in computation time when the number of simultaneous delivery requests grow beyond five tasks; however, the delivery failure rate remained lower than a traditional rule-based approach as the load increased.

The polar robot scenario models a autonomous mobile robot for remote sensing of Polar Regions. The simulation of the polar robot includes a number of remote sensing instruments, including a synthetic aperture radar (SAR), accumulation radar, gravimeter, magnetometer, and IR spectrometer. The robot is also equipped with a solar and a wind generator. The challenge was to balance robot survival and data collection over a full Antarctic year. Constraint models for each instrument, generator, and task are implemented and evaluated versus a rule-based system. The constraint-based system produced significantly lower failure rates, 70% or lower, versus a near 100% failure rate for the rule-based system. The mean survival time using the constraint-based decision maker is greater than 250 days; and the rule-based systems mean survival is less than 200 days. The mean mission completeness of the constraint-based system is significantly greater than

the rule-based system, at a 95% confidence level, for four out of five experimental configurations of the polar scenario.

The urban search and rescue scenario (USAR) models a mobile robot for the mapping and exploration of collapsed buildings to locate victims and hazards. The robot and its environment are simulated based on works on robot-assisted search and rescue. Task models are constructed for searching, reporting results to the rescue party, obtaining repairs, and charging. Eight unique experiment configurations are developed with varying victim injury levels, number of blocked locations, and topologies (hospital and hotel). The constraint-based system performed statistically better than the rule-based system for two out of eight configurations for mean victims rescued; two out of eight for hazards localized; four out of eight for mean number of collisions; and four out of eight for locations mapped. The constraint-based framework meets or exceeds the performance of the rule-based system.

The new decision framework is capable of guiding a variety of mobile robots through rational decisions for task selection and configuration. This is demonstrated in this dissertation for three different scenarios. The framework is flexible to changes in the environment, mission, or tasks. It is also extensible as constraint models can be extended to develop models for new tasks. This work demonstrates that constraint-based decision making is a viable approach to robot task selection and configuration, and performs better than rule-based systems over a variety of applications.

ACKNOWLEDGEMENTS

Special thanks goes to my adviser and committee chair Professor Agah. His knowledge and support has been invaluable throughout my research. I would also like to thank my committee members Professor Chris Allen, Professor Prasad Gogineni, Professor Carl Leuschen, and Professor David Braaten for serving on my committee and providing periodic advice. Thanks must also go to all of those involved with the Polar Radar for Ice Sheet Measurement (PRISM) project and the Center for Remote Sensing of Ice Sheets (CReSIS).

I thank my wife Amy for her patience over the past five years as I have pursued a PhD. She has always given me the needed push when things were looking rough or when stress overwhelmed. My family has continued to be a great support.

Without the assistance of my co-workers/colleagues/friends, Eric Akers, Chris Gifford, and Hans Harmon, I would not have made it past many of my mental roadblocks. In addition, our work together has provided the solid foundation in polar robotics from which part of this research is derived.

This work was supported by the National Science Foundation (grant #OPP-0122520), the National Aeronautics and Space Administration (grants #NAG5-12659 and NAG5-12980), the Kansas Technology Enterprise Corporation, and the University of Kansas.

Contents

LIST OF FIGURES	xi
------------------------	-----------

LIST OF TABLES	xiii
-----------------------	-------------

1 INTRODUCTION	1
1.1 Problem Statement	3
1.2 Hypothesis	3
1.3 Approach Summary	4
1.4 Dissertation Structure	5
2 BACKGROUND	7
2.1 Robot Tasks and Behaviors	7
2.2 Constraint Satisfaction Problem	9
2.2.1 Requirements and Constraints	11
2.2.2 CSP Equivalence	11
2.2.3 Binary and N-Ary CSPs	12
2.2.4 CSP Consistency	13
2.2.5 CSP Solver Techniques	14
2.2.6 CSP Efficiency Enhancements	17
2.3 Modeling	19

2.3.1	Formal Specification Languages	19
2.3.2	Constraint Programming	21
2.4	Rule-based Systems	22
2.5	Resources	24
2.5.1	Koalog Constraint Solver	24
2.5.2	JavaCC Parser	25
2.5.3	Hypothesis Testing and Student's T-Test	26
3	RELATED WORK	29
3.1	ThingLab	29
3.2	Least Constraints Framework	30
3.3	Constraint Nets and Robotics	32
3.4	Behavior Coordination Mechanisms	34
3.5	Robot and Environment Simulation	35
4	RESEARCH APPROACH	36
4.1	Framework Overview	36
4.2	Performance Variables and Constraints	38
4.3	Constraint Language	39
4.4	Linguistic Data Representation	41
4.5	Tasks	42
4.5.1	Active and Inactive Tasks	42
4.5.2	Task Variables	42
4.5.3	Task Constraint Model	43
4.5.4	Task Execution	43
4.5.5	Task Utility	44
4.6	Task Composition	45

4.7	Task Selection	45
5	DELIVERY ROBOT	48
5.1	The Robot	49
5.2	Delivery Environment	50
5.3	Tasks and Task Constraints	51
5.3.1	Wait Task	53
5.3.2	Movement Task	53
5.3.3	Charge Task	56
5.3.4	Mobile Manipulator Task	56
5.4	Evaluation	57
5.4.1	Metrics	58
5.4.2	Data Collection	58
5.4.3	Robot Problems	59
5.4.4	Rule-based System	60
5.4.5	Computing Platform	60
5.5	Results and Analysis	60
5.5.1	Task Load	60
5.5.2	Failures	62
5.5.3	Collisions	64
5.5.4	Service Time and Resource Consumption	65
5.5.5	Discussion	66
6	POLAR ROBOT	71
6.1	Background	71
6.1.1	Applications and Examples	72
6.1.2	Long-term Survivability	73

6.2	Polar Environment Simulation	76
6.2.1	Climate Simulation	76
6.2.2	Terrain Simulation	76
6.3	The Robot	77
6.3.1	Remote Sensing Instruments	78
6.3.2	Power Systems	80
6.3.3	Percepts	81
6.3.4	Drive System	82
6.3.5	Internal Temperature	83
6.3.6	Damage and Failure Simulation	83
6.3.7	Assumptions	85
6.4	Constraint Model	85
6.4.1	Variables	86
6.4.2	Charge Task	88
6.4.3	Survey Task	91
6.5	Evaluation	100
6.5.1	Metrics	100
6.5.2	Experiment Configurations	101
6.5.3	Rule-based System	103
6.5.4	Experimental Procedure	104
6.5.5	Computing Platform	104
6.6	Results and Analysis	104
6.6.1	Failure Rate	105
6.6.2	Survival Time	106
6.6.3	Mission Completeness	107
6.6.4	Tasks	107

6.6.5	Discussion	109
7	URBAN SEARCH AND RESCUE ROBOT	111
7.1	Background	112
7.1.1	Urban Search and Rescue	112
7.1.2	Objectives	113
7.1.3	Levels of Autonomy	114
7.1.4	Locating and Classifying Victims	115
7.2	The Robot	116
7.2.1	Base Station	116
7.2.2	Percepts	116
7.2.3	Drive System	117
7.2.4	Power System	118
7.2.5	Internal memory	118
7.2.6	Internal Map	118
7.3	Constraint Model	119
7.3.1	Variables	120
7.3.2	Abstract Movement Task	121
7.3.3	Search Task	125
7.3.4	Charge Task	126
7.3.5	Upload Task	127
7.3.6	Repair Task	130
7.3.7	Wait Task	130
7.3.8	Utility Function	130
7.3.9	Performance Requirements	131
7.4	Evaluation	132

7.4.1	Metrics	132
7.4.2	Experimental Parameters	132
7.4.3	Rule-based System	136
7.4.4	Computing Platform	137
7.5	Results and Analysis	137
7.5.1	Scores	137
7.5.2	Rescues	138
7.5.3	Reported Hazards	139
7.5.4	Reported Map Locations	140
7.5.5	Collisions	141
7.5.6	Discussion	142
8	CONCLUSION	144
8.1	Contributions	144
8.2	Limitations	146
8.3	Future Work	147
	BIBLIOGRAPHY	160
	APPENDIX A: STUDENT'S T-TEST RESULTS	161

List of Figures

2.1	An example of a constraint satisfaction problem (upper-left) and its solution (lower-right).	10
4.1	Flow of robot autonomy with decision maker.	37
4.2	Composition of constraint data for decision maker.	46
5.1	Delivery scenario: rendering of the environment.	50
5.2	Delivery scenario: class hierarchy.	52
5.3	Delivery scenario: mean CPU execution time versus number of simultaneous tasks.	61
5.4	Delivery scenario: mean failure percentage time versus number of simultaneous tasks.	62
5.5	Delivery scenario: number of failures versus error mode (5000 requests and static obstruction).	63
5.6	Delivery scenario: number of failures versus error mode (5000 requests and dynamic obstruction).	64
5.7	Delivery scenario: number of collisions versus error mode (5000 requests and static obstruction).	65
5.8	Delivery scenario: number of collisions versus error mode (5000 requests and dynamic obstruction).	66

5.9	Delivery scenario: mean execution time per task versus error mode (static obstruction).	67
5.10	Delivery scenario: mean execution time per task versus error mode (dynamic obstruction).	68
5.11	Delivery scenario: mean battery usage per task versus error mode (static obstruction).	69
5.12	Delivery scenario: mean battery usage per task versus error mode (dynamic obstruction).	70
6.1	Polar scenario: failure rate versus configuration.	105
6.2	Polar scenario: survival time versus configuration	106
6.3	Polar scenario: completeness of total mission versus configuration.	108
6.4	Polar scenario: tasks started versus configuration.	109
6.5	Polar scenario: tasks completed versus configuration.	110
7.1	USAR Scenario: class hierarchy.	119
7.2	USAR scenario: the hospital topology.	133
7.3	USAR scenario: the hotel topology.	134
7.4	USAR scenario: performance score versus configuration.	138
7.5	USAR scenario: rescued victims versus configuration.	139
7.6	USAR scenario: hazards reported versus configuration.	140
7.7	USAR scenario: locations mapped versus configuration.	141
7.8	USAR scenario: obstacle collisions versus configuration.	142

List of Tables

6.1	Polar climate almanac for simulation.	77
6.2	Polar Scenario Configurations for Evaluation.	102
7.1	USAR Scenario: Configurations for Evaluation.	135
1	Delivery scenario: failure rate Student's T-test results	162
2	Delivery scenario: collisions Student's T-test results	163
3	Delivery scenario: execution time Student's T-test results	164
4	Delivery scenario: resource consumption Student's T-test results .	165
5	Polar scenario: Student's t-test results	166
6	Polar scenario: Student's t-test results (continued)	167
7	USAR scenario: Student's t-test results	168
8	USAR scenario: Student's t-test results (continued)	169

Chapter 1

INTRODUCTION

An autonomous mobile robot must be capable of rationally selecting and configuring tasks to perform. A task is any robotic action or series of actions to achieve some state or goal. The set of tasks to choose from may be derived from a variety of sources. A high-level planner may generate one or more new tasks at any given time. Alternatively, the tasks could be a queue of requests from users such as a delivery robot receiving pickup and delivery requests. The rational decision is based upon an interpretation of the state of the robot and its environment, the requirements or constraints of the robot's operational performance, and their relation to the task. Specification of the overlapping constraints upon the variables of the decision maker may become quite complex and challenging to specify.

System requirements may vary during operation in order to accommodate the robot's performance, environmental conditions, or user expectations. For instance, a mobile robot navigating through a congested environment ought to operate with greater caution rather than ensure timely delivery. The task selection requirements may be defined in terms of one or more performance objectives. Possible performance objectives may be the timeliness of the operation, risk

avoidance, and resource conservation as the robot carries out its task. These objectives could conflict with one another under many scenarios; thus it is a challenge to develop a decision making system capable of selecting a task, given multiple objectives. A task selection mechanism, such as the one presented in this dissertation, must be capable of adapting to changing requirements while continuing to ensure rationality. In addition, such controller must be capable of resolving conflicting solutions when multiple requirements attempt to influence the decision.

The constraint satisfaction problem (CSP) is a search-based problem domain from the Artificial Intelligence community where the variables of the problem are assigned values that satisfy the constraints set upon them. Constraint programming techniques derived from constraint satisfaction research express very complex relationships between interdependent variables.

CSPs have been applied to many application areas include scheduling, design, and image processing. However, they have not been utilized for robotic applications due to the computational complexity (NP-Complete) of many solution algorithms. As algorithms and heuristics for solving CSPs improve, CPU speeds increase, and software on-a-chip techniques become more prevalent, real-time constraint satisfaction will be possible within the next decade.

The use of constraint satisfaction techniques is a viable mechanism for making multi-objective decisions for robot task selection. A satisfying solution to the CSP model will be one such that the task selected will not only operate correctly, but also satisfy the current requirements set forth upon the robot. A decision maker using constraint satisfaction will provide additional adaptability and extensibility to the robot control system.

1.1 Problem Statement

In order to make rational decisions to guide an autonomous mobile robot, a decision maker must be developed that can handle a variety of unknown situations, conflicting expectations, and changes to the underlying system or set of tasks. Rule-based systems provide an intuitive means of encoding system knowledge into a set of rules that guide the robot. In situations in which complete state knowledge were available, rule-based systems would outperform any other approach.

However, the real world within which the robot operates is seldom static enough to support the rigidity of rules. Rules prescribe actions given predefined states. If a previously unforeseen state should arise, the rule-based decision maker would not be prepared to make a rational decision. Incorporation of changes to the problem domain such as new robot components, new percepts, or even new tasks would require extensive changes to the rule-based system in order to continue to operate successfully. Lastly, some decision makers are inflexible when problems present multiple conflicting objectives that are time variant.

A decision maker is needed that provides the same expressability of the problem model as rule-based systems, but provide flexibility, extensibility and generalizability.

1.2 Hypothesis

The constraint-based methods can be used to model the problem of selecting and configuring tasks for the operation of autonomous mobile robots. CSPs provide a framework in which multiple conflicting constraints upon the robot can be resolved in such a way that the robot will not only perform correctly, but also will meet or

exceed its performance requirements. Constraints also provide an intuitive means of specifying the decision model, but without the rigidity of rules. Given a state, a rule-based system would select an action, while a constraint-based system would narrow the set of possible actions that do not conflict with the system's goals. Furthermore, by incorporating performance objectives into the constraint model, the decision maker will be capable of rationally guiding the robot through actions that best meet current needs and goals.

1.3 Approach Summary

A framework for the constraint-based task selection was developed to support this research. The framework includes the definition of system performance requirements, a robust constraint language, interfaces for defining tasks, and the implementation of the task selector. As part of the constraint system, three variables for the requirements were defined. The variables dealt with the robot's acceptable levels of risk avoidance, timeliness, and resource conservation.

A constraint language was developed to support the specification of complex constraints. The language is based on first-order predicate calculus. It supports common constraints for logic, arithmetic, comparison, and sets. The language is written such that it can be utilized with a variety of constraint solvers. For this research, the Koalog Java Constraint Solver was used. The goal of the constraint language was to provide a robust, composable, and user-friendly language for constraint programming.

The task interfaces provide a general framework to which all tasks must adhere in order to work successfully with the decision maker. Each task is responsible for communication with the robot for control and state knowledge. Each task

generates a CSP model that represents its capabilities relative to the requirements variables. Each task is responsible for its own execution if it is selected by the solver.

The decision maker maintains a list of tasks. It queries each task to determine if it is active. If a task is active, it requests the task's CSP. The task CSPs are merged into a single CSP that represents the entire problem. The CSP is solved and its first N-solutions are extracted, where N is a runtime defined number. From these N-solutions, the best solution is selected, given some utility function. With the solution provided, the appropriate task is executed.

1.4 Dissertation Structure

Chapter 2 presents the relevant background and resources used for the development of the constraint-based decision maker for task selection and configuration. Work related to this research, such as constraint-based research related to robotics and other task selection mechanisms are presented in Chapter 3. Chapter 4 discusses the approach in developing the constraint-based decision maker's framework. In this chapter, the framework will be presented and each of its components discussed in greater detail. Chapters 5, 6, and 7 present application scenarios in which the framework was demonstrated using a simulated mobile robot. In Chapter 5, a proof-of-concept problem is explored by coordinating tasks for a delivery robot. This research is then linked to research conducted at the Center for Remote Sensing of Ice Sheets (CReSIS) in Chapter 6 by defining a model to ensure the success and long-term survivability of a polar mobile robot. Chapter 7 demonstrates the constraint-based system using a robotic urban search and rescue (USAR) scenario. In Chapter 8, the viability of constraint-based task

selection and configuration is discussed based on the results from each of the scenarios. The contributions that the research has made to both the robotics community and CReSIS are defined. The current limitations of the system are considered. Finally, the future work of this research is discussed.

Chapter 2

BACKGROUND

This chapter presents the background to the research conducted for this dissertation. An explanation of the similarities and differences between robot tasks and behaviors is provided. The constraint satisfaction problem and associated research are presented. Task modeling is discussed with a focus on constraint programming. Rule-based systems, which will be used as experimental control are described. The resources used to conduct this research are presented. Finally, an overview of statistical analysis using Student's T-Test is covered.

2.1 Robot Tasks and Behaviors

The American Heritage Dictionary provides a number of definitions for “tasks”. The two that best meet the goals of this research are “a piece of work assigned or done as part of one’s duties” and “a function to be performed; an objective” [3]. For this research, the task is then assigned to the robot and carried out by the lower-level control system.

Researchers from Carnegie Mellon University defined for their own research a

task description language. In their research, they defined a task as the control necessary for resource management and exception handling. In the three-layer robot control hierarchy, tasks and task selection reside in the middle layer between the Planning Layer and the Behavior Layer [69].

Finally, for multi-robot systems, the term task was simply defined as a subgoal that is necessary for achieving the overall goal of the system [32]. The tasks can be discrete such as delivering a package, or continuous such as monitoring a door.

For this research, all three definitions are applicable. The assumption is that tasks are derived from some high-level planner or from a user. These tasks represent one or more actions that the robot must perform at the lower layer of robot control. During execution, the tasks may react to exception states by adding or removing additional tasks.

As stated above, tasks are defined at a layer above robot behaviors. Behavior-based robotics defines the robot's control and decision making with respect to a set of behaviors. Behaviors may be loosely defined as actions used to achieve or maintain goals. Unlike tasks, these actions and goals are lower level and may be based on control-theory [50]. Behaviors may be coordinated through a variety of mechanisms including higher-levels of abstraction using tasks. Significant research has been conducted in order to determine how best to coordinate behaviors. Many of the techniques share similarity with the task coordination discussed within this dissertation. As a result, these techniques will be presented in much greater detail in the upcoming related work chapter. The constraint-based decision maker defined within this dissertation would be capable of coordinating behaviors. However, the complexity of the constraint models would increase greatly.

2.2 Constraint Satisfaction Problem

The constraint satisfaction problem (CSP) is a problem solving technique in Artificial Intelligence. In general, a problem is specified as a set of variables whose values are restricted by constraints. Search techniques determine a solution. A CSP's solution is a valid assignment of all variables such that no constraints on the variables fail. One or more solutions may exist [67].

A constraint satisfaction problem may be visualized and solved as a constraint graph, or a constraint network. Variables represent nodes within the graph. Constraints are represented by arcs that connect variables that are constrained together. Using a graph to represent the CSP does not only serve as a visualization tool, as techniques using graph theory have been developed to solve or optimize constraint graphs [25, 26, 27, 67].

One of the major advantages of the constraint satisfaction problem framework is that it provides a domain-independent representation of problems. Therefore, general search techniques and heuristics may be utilized. A variety of application areas use constraint satisfaction techniques. A classic problem is map coloring. Each region on the map may be represented as a variable whose domain is the set of available colors. Constraints exist between bordering regions specifying that they cannot have the same color. Search techniques can be utilized to find a valid color assignment for each region such that the constraints are not violated [67].

The constraint satisfaction problem approach has been used to solve several real-world problems. Scheduling and planning are common application domains for which CSP techniques are used. A few areas in which such systems have been applied include university course timetabling [28], scheduling tasks for manufacturing processes [81], and scheduling resources for a trucking company

[21]. Airline scheduling is another common application of CSPs [67].

Figure 2.1 provides an example of a constraint satisfaction problem (upper-left) and its solution (lower-right). For each, a constraint graph is provided. The nodes represent the unsolved variables. The domain for each respective variable is presented in curly-braces next to the node. The arcs represent the constraints between the connected variables. The solution contains a single value in the domain for each variable. None of the values within the curly braces for the solution violate the defined constraints.

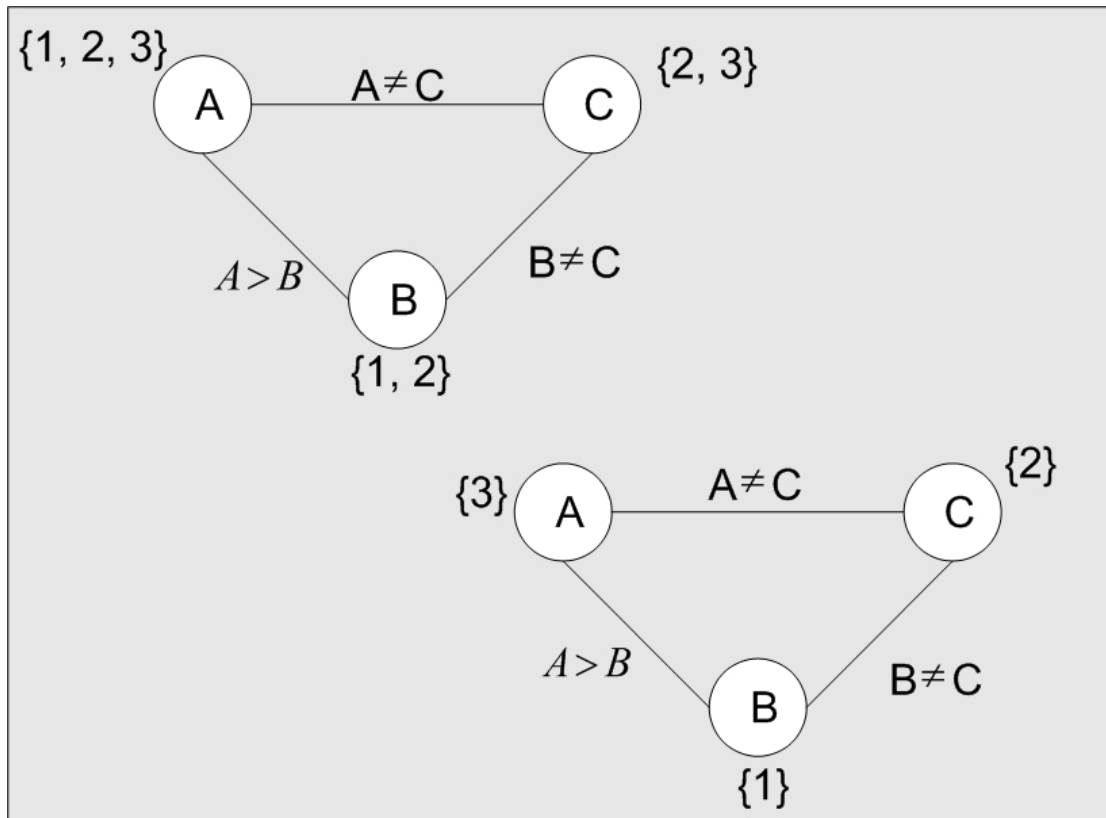


Figure 2.1: An example of a constraint satisfaction problem (upper-left) and its solution (lower-right).

In following sections, additional background regarding the constraint satisfaction problem and its formulations is presented. The differences between binary and

n-ary constraints are described, a variety of CSP solver techniques are discussed, and the equivalence property for comparison of two CSPs is presented.

2.2.1 Requirements and Constraints

Requirements are often set on a system to define the expectations of how the system should act while performing a task. The Webster-Merriam Dictionary [84] defines a requirement as “something wanted or needed”. In the robotics domain, a requirement may be that the robot travel at a maximum speed of five kilometers per hour.

Constraints enforce the requirements of the system. The Webster-Merriam Dictionary [84] defines a constraint as the “act of constraining”, where constrain is defined as “to force by impose structure, restriction, or limitation”. In a constraint satisfaction problem, a constraint limits the assignment of values to a variable. Constraints therefore enforce the requirements of the system by limiting specific elements of the system. For instance, to enforce the requirement of five kilometers per hour maximum speed, constraints on the left and right motor controlling the wheels limit their output to rates below five kilometers per hour.

2.2.2 CSP Equivalence

Empirically, two CSPs may be compared to determine if they follow the definition of equivalence. It is a system property that may be tested offline during the initial verification of the system.

The simplest and also one of the narrowest definitions of CSP equivalence is that two CSPs are equivalent if they possess the same variables (with identical domains) and constraints [66]. Determining equivalence with this definition

is simple, but it excludes many CSPs that are equivalent. For instance, two constraints between a pair of variables may be easily reduced into a single constraint between the pair of variables. The solution for the two CSPs would be identical, but by this definition they would not be equivalent [66].

A more common definition of CSP equivalence is that the pair of CSPs are equivalent if their solution sets are identical. Thus, the set of solutions of one CSP must be identical to the set of all solutions for the other CSP. This definition addresses redundant constraints, but does not take into account the possibility of redundant variables that may exist [66].

[66] presented an extended definition of CSP equivalence that considers both redundant constraints and variables. The definition states that two CSPs are equivalent if their solutions are mutually reducible. Reducibility implies that the solution of one CSP can be mapped, given a function, to the solution of the other CSP. This definition is transitive, reflexive, and symmetric.

2.2.3 Binary and N-Ary CSPs

A binary CSP is defined as a constraint satisfaction problem in which constraints are defined only between one or two variables (unary or binary). Unlike n-ary CSPs, more algorithms already exist and/or have been improved for solving binary constraint satisfaction problems. A binary CSP can be generated from n-ary constraints.

One technique to produce a binary constraint from an n-ary constraint is the addition of variables. These variables provide the link between the variables within the n-ary constraint. Based on the definition of equivalence provided by [66], such conversion will not produce equivalent CSPs. The solution of the n-ary

CSP may be reduced to the solution of the binary CSP, but the relationship is not bidirectional when going from the binary CSP to the n-ary CSP.

Another technique which does hold the equivalence between the n-ary and binary CSP is the dual-graph technique [66]. In the dual graph, nodes represent constraints and their domains are the legal combinations of values between the variables within the constraint. The arcs between these constraint nodes represent common variables within the constraints and enforce that the variables' instantiation are the same for both constraints. In the dual-graph form, there is a new and legal binary CSP that may utilize any binary CSP solution or reduction technique [66, 26].

2.2.4 CSP Consistency

Consistency is a property of a CSP that directly correlates to the effort required to solve a CSP. It relates to the presence of values within the variable domains that may conflict with a constraint. More formally, a CSP is K-consistent if for all sets of (K-1) nodes with consistent value assignments (no variable conflict), any Kth node that is added to the set will have a value within its domain that is consistent [67].

Node consistency (1-consistency) is the weakest consistency. A CSP is node consistent if there are not any nodes that violate unary constraints upon those nodes. Arc consistency (2-consistency) guarantees that for all pairs of variables within the CSP there are not any values within either variable's domain that may cause a conflict. Path consistency (3-consistency) ensures that for any three variables there are not any possible assignments that may result in a conflict for any constraints between the variables. A variety of algorithms exist to convert a

CSP to one of these three levels of consistency [45, 47, 67].

Strong K-consistency introduces additional restrictions. A CSP is strong K-consistent if it is (K-1)-consistent, (K-2)-consistent,..., 1-consistent. Obtaining strongly n-consistency where n is the number of variables in the CSP guarantees that no backtracking will be required to solve the CSP during search. Obtaining strong n-consistency is shown to be NP-hard [45, 47, 67]. An NP-Complete problem is a member of a class of problems that cannot be solved in polynomial time or better [67]. Stating a problem is NP-hard implies that it is NP-Complete or harder [55]. Constraint propagation refers to techniques that are used to improve consistency such as arc consistency algorithms [45, 67].

2.2.5 CSP Solver Techniques

Search is commonly used to solve CSPs. However, a variety of other techniques exist to solve CSPs. Selection of a particular solver can have a profound impact on the speed and consistency with which a solution can be obtained for a problem.

Backtracking Techniques

Backtracking search is quite similar to depth first search [67]. A node (variable) can be selected randomly, in order, or based on a particular heuristic. For each node, the variable is instantiated with a value from its domain (randomly or based on a heuristic). If the instantiated variable does not contradict any constraints, the algorithm instantiates the node. If a conflict exists, the next value within the variable's domain is tested. If no additional values exist, the algorithm backtracks to the previous variable and tries a new value. This algorithm requires worst-case exponential time [67].

Forward Checking

Traditionally, constraint propagation techniques are performed prior to solving the CSP with backtracking. Forward checking is a technique that can be utilized such that arc consistency is performed during the search [67].

Back Jumping

Back jumping makes the act of backtracking more intelligent within the CSP solver. Rather than backtracking to the previous node when a constraint conflict occurs, the algorithm backjumps along the search path to the node that is in conflict with the current node based on that constraint [67].

Repair Techniques

For particular CSP problems, a repair technique is capable of solving the problem much faster than using backtrack search techniques. Instead of searching for proper instantiations, the variables are randomly (or based on a heuristic) assigned values within the variables' domains. A repair algorithm finds conflicts and repairs them. The repair algorithm often relies on a heuristic such as min-conflicts. With min-conflicts, the value with the minimum number of resulting conflicts is used to replace the currently conflicting value [52].

The repair techniques perform relatively well. Their performance is often independent of domain size. They are known to work well for simple and difficult problems. For real-time applications, the technique works well since only minor changes may occur. Rather than solving the entire problem again, the conflicts from the update are repaired.

Dynamic CSP Techniques

Dynamic CSPs (DCSP) are those problems for which either constraints, variables, or allowable variable domains change for each iteration, or subsequent call of the solver, [25]. Rather than solve the entire CSP for each iteration, techniques exist that incorporate the information gained from the previous solution in order to solve the current problem faster [83]. Feedback of the previous solution is therefore required.

One of the most common DCSP techniques is to utilize the min-conflicts local search/repair algorithm [52, 83]. Rather than having a CSP that is randomly instantiated, the previous solution is used as the initial instantiation. Search is then used to repair the conflicts between the current problem and the previous problem's solution.

Other techniques have been developed to take advantage of DCSPs. For instance, Arc Consistency algorithms such as AC-4 (binary CSP) and GAC-4 (binary and non-binary CSPs) have been updated to utilize previous solution information [6, 7].

CBR + CSP Techniques

Another Dynamic CSP technique is the incorporation of Case-Base Reasoning (CBR) and the repair algorithm [61, 73]. For these problems, rather than using the previous solution as the instantiation of the variables, a case-base is consulted and the solution (or reduced CSP) that best matches the current problem is recalled. The repair algorithm then repairs the conflicts between the previous problem and the current problem [61, 62]. The CBR approach produces more overhead, but in problems where the CSP for two epochs are more distant, this

technique works because in the long run, a case will likely emerge more than once.

2.2.6 CSP Efficiency Enhancements

There are several properties that may describe a more efficient CSP, these include [82]:

1. Smaller domains for variables (reduces maximum domain size).
2. Reducing the number of variables within CSP.
3. Reducing the number of constraints (fewer comparisons during consistency checks).
4. Node ordering (order may reduce the number of backtracks).
5. Reducing the width of an ordered CSP.
6. Obtaining strong consistency:
 - (a) Node consistency: strong 1-consistency with linear time complexity.
 - (b) Arc consistency: strong 2-consistency with $O(ed^2)$ time complexity, where e is the number of constraints and d is the size of the largest domain.
 - (c) Path consistency: strong 3-consistency.
 - (d) Strong K-consistency: guarantees backtrack free, but NP-Complete.
 - (e) Strong $(w+1)$ -consistency: guarantees backtrack free search, where w is the width of constraint graph (see below).

Node Ordering and Width

The ordering of nodes and the width of a graph are often parameters that affect the performance of a particular CSP solution or reduction algorithm. They must be considered when constructing the CSP modules and building the complete CSP. Width of a CSP, given an ordering, is defined as the maximum nodal width within the network. The nodal width is defined as the number of previous nodes directly linked to the node via constraints [27, 31]. Selection of an optimal ordering (with minimal width) is an NP-hard problem [27]. Therefore, a heuristic technique must be utilized to order the nodes.

Width, Strong k-consistency, and Trees

Given an ordered CSP whose width is w , if the graph were to become strongly K-consistent through some algorithmic means, a solution may be obtainable linearly (search without any backtracking)[31]. Often however, strong K-consistency with $K > 3$ yields results with few backtracks.

Trees are ideal, but seldom obtain CSP graph structure. A tree has a width of one. Therefore, using a simple arc consistency algorithm, which yields a strong 2-consistency graph, it may be solved without backtracking.

Obtaining strong K-consistency often involves the addition of variables or constraints into the graph. With each additional node, the width may increase. Therefore, for many problems obtaining strong K-consistency for $K > w$ remains infeasible [45].

2.3 Modeling

For the proposed research, a model of a robotics problem must be transformed into a constraint satisfaction problem. In this section, background regarding a pair of modeling paradigms is discussed. Formal specification languages are often more general and provide some guarantees regarding performance based on mathematical analysis. Constraint programming languages are designed to represent CSPs, but lack the guarantee of correctness for the model.

2.3.1 Formal Specification Languages

Formal methods apply mathematical techniques for specification and/or validation of a system. Formal specification provides a description of how the system should perform without describing the actual implementation of the system. Verification of an implementation based on these specifications guarantees that the system will perform as specified. It does not however eliminate the need for testing as the specification itself could be incorrect. Formal methods are not limited to software design. Hardware verification and software/hardware co-design are also common uses of formal methods [11, 46].

A general procedure for using formal methods may be as follows. A specification model is written to define the required behaviors of the system. An implementation model is written to specify the structure of the system. Mathematical properties are defined between these two models to enforce various properties of the system. Finally, the system is verified [8]. A variety of verification techniques exist such as model checking, syntax and/or type checking, theorem proving, etc. Selection of a verification technique is highly domain-dependent.

Formal methods are quite common. Formal specification and verification is not

only necessary, but often required when developing critical systems. Such systems may possess particular safety or mission requirements that must be accomplished. Field and planetary robotics are particular examples of critical systems that could benefit from formal method techniques [30]. In situ repair of such systems could be quite costly, if not impossible, given a system design error.

Defining a specification language and a toolset for formal methods is non-trivial. When defining a specification, the balance between specificity and abstraction must be addressed. Insufficient abstraction results in weak specifications, but easier implementation. Insufficient expressivity lowers the learning curve, but limits the ability to specify more complex system properties [8]. Often it is much easier to utilize an existing specification toolset. The Z (pronounced “zed”) Notation [72] is one such software tool for formal system specification. VDM is another popular formal method tool [40]. VDM and Z provide similar methods and have similar abilities. As such, their similarities have been analyzed [34, 51].

Formal methods have been introduced into constraint research. Massachusetts Institute of Technology’s Software Design Group has developed Alloy, which is a specification language and a verifier for specifications of problems with relational properties such as the constraint satisfaction problem [37, 29]. Alloy is based on Z and the Object Constraint Language [64], an extension of UML, a graphics-based modeling and specification language for software. Because CSPs have a large state space, the verification system was only capable of verifying a user-selected subset of the specification [37].

2.3.2 Constraint Programming

A common semi-formal or informal methods of specifying a constraint satisfaction problem is employing constraint programming techniques [4]. Constraint programming is often declarative in which the system is modeled using either a special language or a library that is often integrated with a solver for the respective model. Constraint programming languages specify constraints that are commonly encountered when specifying a constraint satisfaction problem. Global constraints are also common where a single constraint can limit the domains of all variables within the CSP rather than specifying individual constraints for all variable combinations. For instance, the "all-different" constraint is commonly available to specify that no variables within the CSP may share the same value. Constraint programming is divided into two classes of constraint logic programming (CLP) and the more simple constraint programming [4, 78, 82].

Constraint logic programming is an extension of logical programming such as Prolog. Logic programming can be viewed as constraint logic programming in which only the equivalence constraint is available. Implementations of constraint logic programming vary dramatically. It is quite common for these languages to be built upon an implementation of Prolog. For instance, Prolog III [22] is a commercially available CLP language. CHIP is another popular commercial CLP language. It incorporates several constraint satisfaction solving techniques such as forward checking [23].

More general constraint programming offers a variety of methodologies and languages. It is not uncommon for a constraint system to require specification of the CSP using its own language. Others utilize a library for programming languages such as C/C++, Java, Smalltalk, LISP, Scheme, etc. Within a program,

using the data structures available within the library, a CSP is constructed which is solvable by the constraint system's solver engine. As with constraint logic programming, these systems vary greatly between implementations. Several non-commercial are available. However, many of these solvers lack the power and completeness of commercial alternatives. ILOG [36] and Koalog [42] are both commercially available libraries/engines for solving constraint satisfaction problems using constraint programming techniques.

2.4 Rule-based Systems

Rule-based systems define a set or class of software-based problem solvers from the Artificial Intelligence community. For rule-based systems, the problem solver utilizes a set of situation-action rules (if-then conditionals). Research in rule-based systems coincides with knowledge-based systems and expert systems in which the rules are accompanied by a set of known facts.

Rule-based systems are described as the encoding of human knowledge of a problem into a set of if-then rules [35]. The inference engine will select the relevant rules and combine the results. It will determine the best sequence of rules to execute. Lastly, its decision making is completely transparent and may be reviewed by the user.

As the rule-base increases in size, its level of skill increases. As the rule-base possesses rules for more specific situations, the overall success rate of the system will improve.

The use of a complex inference engines is not common in the domain of control and robotics when rule-based systems are applied. An example of a common rule-based system for robotics is the fuzzy controller.

In a fuzzy controller [79], inputs are fuzzified into linguistic values. These fuzzy membership values are then fed to the rule-base. From this point, a set of if-then statements will decide the robot's next actions. For example, one such rule could be "If speed is slow and time remaining is low, drive at high speed".

The system will not fuzzify its data, but it will translate continuous data into a smaller set of linguistic values. These values will then be fed to a set of rule which will select the task to perform and the desired configuration.

The difference between a rule and a constraint may be difficult to determine. Clearly, constraint-based systems and rule-based systems use different methods to generate their results. Furthermore, the constraint-based system may generate a number of valid results while some rule-based systems generate only one.

A rule from a rule-based system can be seen as prescriptive. If x and y , then z . Where a very explicit outcome is defined.

For CSPs, the constraints are restrictive instead of prescriptive. They define the bounds for which the variable may be capable of being assigned. If x and y the $z > \text{LOW}$.

It is easier to cover a broader range of states using constraints. For a rule-based system, every state must be defined with its own rule. If a rule does not match, the robot's action may be unpredictable. With the constraint-based system, a single result might not be generated, but it is known that all results are valid, given the current constraints upon the system.

One major limitation of CSPs when compared with rule-based systems is that it is not possible to provide a trace of the solver's decision making process

For the proposed research, rule-based systems are used for evaluation of the constraint model, serving as the control group.

2.5 Resources

In this section the resources utilized for the development of the constraint-based system and its scenarios are briefly discussed.

The Java Standard Edition Developer Kit (SDK) and Java Runtime Environment (JRE) [80] versions 1.4.2 and 1.6.0 were utilized for the development, implementation, and testing of the research discussed within this dissertation.

2.5.1 Koalog Constraint Solver

A variety of constraint programming packages with integrated solvers are available, both commercially and as open-source. The Koalog Java Constraint Solver [42] was selected for the proposed's research. Koalog has an available constraint language as specified in [41].

Koalog provides several solvers and built-in heuristics to decrease the computation time necessary to solve CSPs. In order to model a CSP, Koalog utilizes integer variables with a variety of domains. Variables using the Boolean domain can only be assigned the values one for true and zero for false. The MinMax domain limits the assignment of the integer variable to any integer value between a minimum and maximum value. The Sparse domain allows the user to specify a spare set of possible values. For instance, a variable may be assigned the sparse domain of even integer values from zero to nine by specifying an array $\{2, 4, 6, 8\}$.

Koalog's supported constraints are divided into three classifications. Arithmetic constraints define constraints that apply arithmetic operations to enforce an arithmetic relationship between one or more variables. For example, the add constraint enforces, given three integers, a , b , and c that $a = b + c$. The Less constraint applies the relation to two integer variables a and b such that

$a < b$. Boolean constraints contain common Boolean arithmetic operations such as And, Or, Not, If-Then, etc. New constraints can be constructed using existing constraints, by building a small CSP that enforces the entire relationship between the involved variables. CSPs may be combined, resulting in larger CSPs.

In order to solve the CSPs, several solution algorithms, variable heuristics, domain heuristics, and variable optimizers exist.. Many of the filters and optimizations for CSPs to reduce the computational complexity are not included in Koalog. Solvers available include backtracking search-based CSPs solvers and local search solvers. Only the backtracking solvers have been used for this research. Variable domains define the order in which variables are selected to be assigned values during the backtracking search. Domain heuristics define the order in which the domain space for each variable is attempted during backtracking search.

Since Koalog is written in Java, the software written that utilizes its libraries are portable.

2.5.2 JavaCC Parser

JavaCC [39] is a compiler designed to generate a program that can parse one or more strings, interpret the strings, and perform the actions requested by the strings. The application shares similarities with both Lex and Yacc. Similar to Lex, JavaCC contains a lexical analyzer that breaks the strings into a set of tokens that may be utilized by the parser. Similar to Yacc, JavaCC utilizes the tokens that are generated and a set of rules to produce a software process to perform the desired operations given the input tokens. As discussed later, JavaCC was used to generate a constraint specification level that used a syntax similar to first-order predicate calculus to abstract away from the Koalog API [41].

2.5.3 Hypothesis Testing and Student's T-Test

Statistical analysis for this research involved hypothesis testing with Student's T-Test.

For hypothesis testing, a null hypothesis is proven or disproven. Often, the null hypothesis is accompanied by an alternate hypothesis for which one hopes to prove. To prove or disprove the null hypothesis, a probability is calculated. If the probability of the null hypothesis is lower than the specified confidence level, commonly 95%, it is rejected and the alternative hypothesis is accepted. However, if the null hypothesis is not refuted, it cannot be concluded that the alternate hypothesis is invalid. Rather, at the specified confidence level, it cannot be supported.

One of example of using hypothesis testing is to prove that a variable's mean is not equal to a particular value. For example:

$$h_0 : \mu_0 = \mu$$

$$h_1 : \mu_0 \neq \mu$$

If μ_0 does not equal μ , the null hypothesis will be refuted and the alternative hypothesis will be said to be accepted within a 95% confidence interval.

In order to obtain the probabilities discussed, Student's T-Test can be applied. Student's T test is designed for data sets in which the variances are unknown and/or the sample size is quite small. For the T-Test, a t value is calculated and the number of degrees of freedom is equal to $n - 1$ where n is the sample size. In Equation 2.1, the t-test formula is given when comparing one random variable with a specified mean. Once a t-value is calculated, a table is used to look up the

appropriate probability given the t-value and the number of degrees of freedom.

$$t = \frac{\bar{x} - \mu}{\frac{s}{\sqrt{n}}} \quad (2.1)$$

For this research, when analyzing test results, often two random variables must be compared. These random variables are the results for a metric generated from the constraint-based system and the results for a competing rule-based system. One cannot examine the mean of each and immediately determine that one is better than the other. Using the t-test, the superiority of one system over another must be proven with some statistical significance, using hypothesis testing. Equation 2.2 may be used to compute the t-value assuming that the random variables do not have equivalent variances.

$$t = \frac{\bar{x} - \bar{y}}{\sqrt{\frac{s^2}{m} + \frac{s^2}{n}}} \quad (2.2)$$

An example of a hypothesis comparing two random variables is as follows:

$$h_0 : \mu_0 = \mu_1$$

$$h_1 : \mu_0 > \mu_1$$

This states that the null hypothesis is that the mean of both random variables is equivalent. The alternate hypothesis is that μ_0 is greater than μ_1 .

Depending upon the variables, it may be desirable to determine which yields the greater value such as the number of lives saved for a search and rescue operation. In other cases, smaller values imply better performance such as package delivery times. This may merely be defined through the alternative hypothesis.

For the alternative hypothesis $h1 : \mu_0 > \mu_1$, the right-tail test must be performed. This implies that the lookup table for the right-tail must be used given the calculated t-value. If $h1 : \mu_0 < \mu_1$, then the left-tail test must be used. If $h1 : \mu_0 \neq \mu_1$, then the two-tail test is performed.

To automate the data collection process, Matlab's Statistical Toolbox [49] was used. The *ttest2* function provided the functionality necessary to compare the two random variables. The experiment was performed checking both if $\mu_0 < \mu_1$ and $\mu_0 > \mu_1$. Therefore, it can be concluded with 95% confidence, which of the two is better or if a confident conclusion cannot be made.

Chapter 3

RELATED WORK

In this chapter, work related to the proposed research is discussed and evaluated.

3.1 ThingLab

ThingLab [10] was developed in the early 1980s as a software package that allowed users to graphically design and simulate a system. The tool could be used to simulate a circuit, mechanical linkages, geometries, and a variety of other objects. It employed constraint satisfaction techniques to describe the properties of the system and support simulation. ThingLab was built using Smalltalk, which was an early object-oriented programming language. ThingLab carried these object-oriented properties into its system.

Components within the simulator were represented as objects. These objects were comprised of a set of variables and constraints. Non-primitive objects were comprised of parts (other objects). These parts could be primitive objects, or comprised of parts themselves. For example, a resistor object was comprised of two variables representing the two end points. The property of resistance

was represented as a constraint of these two points. To create a circuit object comprised of two resistors in series, the object would have a merge between two resistor objects.

The merge concept introduced to constraint satisfaction problems was simple. The endpoint variables of each resistor were replaced by a single shared variable. The original objects retained information regarding the merge such that it could be undone if the configuration changed. This resulted in a constraint satisfaction problem in which the value difference between the one variable end of the circuit object and the second end of the circuit satisfied the constraints brought upon by Kirchoff’s voltage laws. Equivalence constraints could be used instead of a shared variable.

The concept of representing constraint satisfaction problems as a combination of smaller constraint satisfaction problems has inspired the work in this dissertation. Unlike the proposed research, ThingLab does not utilize a specification language that represents a constraint satisfaction problem. Rather, the CSP representing the simulated objects within ThingLab is built incrementally. The CSP remains as a data structure that can be stored and recalled.

3.2 Least Constraints Framework

In the late 1980s and early 1990s, D. K. Pai at Cornell University proposed a framework for robot programming using constraint satisfaction techniques [57, 58]. Rather than programming a robot using an arbitrary set of actions in order to perform a desired motion, a method was proposed for partially specifying the desired action of the robotic system using linear inequality constraints. Two special classes of variables were defined, input and output. Input variables

represented values that were received from an external agent. Output variables were readable by an external agent. Within the framework, the following process for specification was introduced. First, the user had to specify the variables and variable domains. Next, constraints were introduced to direct the system’s behavior.

The framework was later extended and named the “Least Constraints Framework” [58]. The extension to the framework introduced a few new concepts. Variables could be divided into specialized domains, representative of functionality within the system. This added some modularity to the specification. This division of variables provided additional organization for specification of constraints.

For constraint satisfaction, repair algorithms were utilized in which the previous solution acted as a starting point to the new solution process. This reduced the amount of change between states, as the robot would more likely take an action similar to its current action. This solution technique provided an anytime property as the system could always produce a solution that is close to the satisfying all constraints.

For simplicity, the least constraints framework is limited to only utilizing linear inequalities for constraints and numeric variables. In the proposed research, a variety of more complex constraints are incorporated, e.g., set theory constraints. Variables within the proposed system may be numeric or symbolic. By increasing the number of constraint and variable types within the system, more complex models of robot components can be constructed that utilize fewer constraints because of the added expressivity.

The concept of connections introduced in the proposed research extends Least Constraint’s variable groups. However, connections are used to specify modular robotic components while the variable groups were used to specify constraints

upon a group of related variables. Adding modularity reduces the complexity of specification writing, so that the designer can focus on a specific component rather than the entire system.

Constraints in the Least Constraint Framework are soft constraints. This means that a solution does not necessarily have to solve all constraints. However, the quality of a solution is determined by the number of constraints satisfied. Since a repair algorithm is used, this property provides an anytime property to the controller, such that a solution for the robot is always available. For the proposed system, a much different approach is used. Constraints in the proposed system are considered hard constraints that must be satisfied for the robot to act. This guarantees that the robot acts as desired. In cases in which the robot cannot satisfy all of its constraints, it is likely in a situation in which it ought not to act for safety or behavioral reasons. It is therefore better for the robot to not act until the input has changed such that it may act again. For instance, if an obstacle blocks the robot such that it can no longer move toward its goal, it is required that the robot does not proceed forward and collide with the obstacle.

3.3 Constraint Nets and Robotics

At the University of British Columbia, Zhang and Mackworth have developed a constraint-based system for the design of intelligent agents [48, 86, 87, 88]. The system was designed to support hybrid agents i.e., agents with both continuous and discrete interactions with their environment. The control systems could also be designed hierarchically such that higher-level logic such as planning would occur at the higher layers, while lower level control such as moving an actuator occurred at lower levels.

Zhang and Mackworth’s framework known as Constraint Nets, CN, represents a problem in a form similar to a control system. Constraint Nets are different from the phrase “constraint network”, which is used interchangeably with the “constraint graph” that represents a CSP. A CN is composed of a finite set of locations (variables), transductions (relation or causal mappings of input to output), and connections that tie locations to the ports of the transduction. Within such a model, hidden locations may represent internal state within the controller. The model can then be represented as a series of equations. The overall behavior of the system was represented by the solution of all equations defined within the CN [48, 86, 87].

With design of a robotic system, a model of the robot’s control, the plant (or representation of the robot itself), and the environment were constructed. A specification of the behavior was required for validation. A controller could be synthesized by generating a controller that drove the system toward satisfying the constraints of the constraint network models. If the system deviated from the constraints, its behavior was driven by the controller to a constraint satisfying state [48, 86].

They have utilized constraint nets to several application domains. Within the field of robotics, the most notable is for robot soccer. Using a soccer simulator, controllers were modeled using the constraint network framework in Java [88]. Using the University of British Columbia Dynamite Testbed, they have tested robot soccer control on a physical mobile robot [47].

Much of their work focused on integrating constraints into formal specification of robot controllers, and not into controlling the robot using the constraints themselves. The generated controller was designed such that it forced the robot’s actions toward satisfying constraints. Therefore, the system would

deviate from the specified path, but the deviation was guaranteed to be below a predefined level. The proposed research in this dissertation utilizes the constraint satisfaction problem as the controller itself. Using search techniques, the constraint satisfaction problem can be solved to produce robot control signals. The control signals generated are guaranteed to satisfy the robot’s specification as they satisfies all constraints.

3.4 Behavior Coordination Mechanisms

Behavior-based robotics is a paradigm for mobile robot control. Rather than focusing on complex computations to deliberate the robot’s next task, the robot takes a much more reactive approach. A behavior ties the robot’s control directly with percepts. Purely reactive systems lack the ability to direct the robot toward global goals. Behavior coordination and selection mechanisms have been studied extensively.

In [59], a taxonomy of behavior selection and coordination mechanisms is presented. The goal for such a mechanism is to activate the best behavior to suit the robot’s needs and goals. Within the taxonomy, state-based competitive coordination is discussed. Coordination mechanisms within this class base the decision of the robot’s next behavior or set of actions on the current state. The behaviors must compete against one another such that only one is selected.

One technique discussed is the Discrete Event System (DES). In [43] and [44], the DES system is considered for autonomous mobile agents. These agents are not limited to mobile robots, but also software agents. For the framework, composition operators are used to construct a nondeterministic finite state machine such that percepts and actions are sequenced together. Event drive transitions between

states such that the behaviors may be composed.

In [20], motivating factors were applied to the selection of behaviors for a mobile robot. The motivation was based on animal characteristics for contingency handling. It was assumed that given the robot’s emotional assessment of the state, it would be motivated to choose a particular task. Four emotions were included. Friendliness indicated how well the robot was currently interacting with its environment and collaborating with others. Fear indicated the level of concern the robot had regarding risks given the current state. Frustration was based on the robot’s difficulty in achieving a goal state. Fatigue was indicative of the robot’s current resource availability.

The constraint-based framework incorporates concepts derived from [20]. Rather than making the decision based on emotions, the task selection is guided by the desired levels of performance for the robot. Therefore, the constraint-based robot is provides a service to the user.

3.5 Robot and Environment Simulation

For each of the scenarios described within this research, custom simulations were developed to model the mobile robot. For robots of harsh environments, the use of simulation is an important task for validation of software techniques. Simulators are used to eliminate the complexity of working with physical hardware. In addition, for situations such as urban search and rescue or polar exploration, researchers are often separated from the test environment during research and development. Therefore, simulation is a viable technique that can be used for robotics research. The topic of simulation and the development of abstract software interfaces for software portability is discussed in [76].

Chapter 4

RESEARCH APPROACH

In this chapter, the research approach for developing a constraint-based decision maker for the selection and configuration of tasks for autonomous mobile robot is presented. First, an overview provides a top-level view of the approach. Next, the details of the employed framework are presented.

4.1 Framework Overview

The proposed decision making framework selects a task that will guide the mobile robot to perform some action(s) autonomously. It does not make the underlying low-level control decisions, which fall outside of the scope of this research. The decision maker lies above the low-level controller, but below an optional high-level mission planner.

Figure 4.1 demonstrates the relation of the decision maker with the overall autonomous control of the mobile robot. The decision maker for task selection is placed between the mission planner and the robot controller. The mission planner and decision maker communicate such that the decision maker passes up current

decision information and the mission planner passes down the current set of tasks, the addition of new tasks, or the performance requirements. The performance requirements and tasks may also be provided from an outside source. Once the task is selected, it is executed and its control signals are passed down to the robot controller, which controls the physical robot through its actuators. The robot controller receives data from the robot's sensors. This data may be shared with the decision maker. The robot's drive and actions affect the world state. The current world state is sensed by the robot's sensor sub-system.

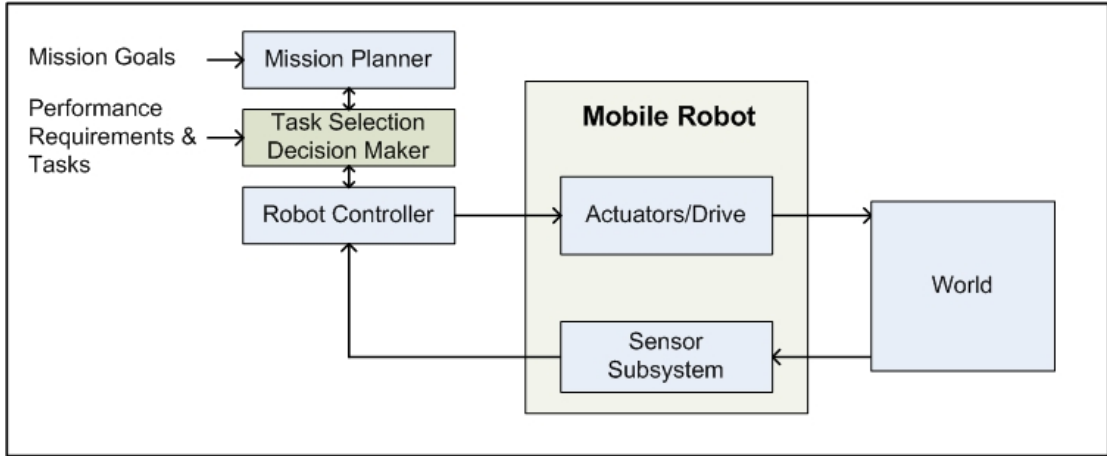


Figure 4.1: Flow of robot autonomy with decision maker.

A constraint satisfaction problem is generated that merge the constraint models for each potential task and the performance constraints set upon the system. The CSP may be solved to generate a set of one or more solutions. Each solution specifies one satisfying task and its constraint satisfying configuration. From this set of satisfying tasks, the best task is selected and executed, given the configuration of the CSP's solution as parameters.

Each task's model combines the constraints of the action(s) and the desired performance characteristics of the task's execution. CSP variables are defined that

represent the task’s percepts, control parameters, and any hidden state details. Percept variables are instantiated with the current perceptual data from the robot and cannot be modified by the CSP. Hidden variables define intermediate states by applying logic, arithmetic, and comparison constraints upon the percepts, and/or control variables. The control variables are used to configure and guide the task’s execution such as desired path, desired speed, etc.

4.2 Performance Variables and Constraints

System performance constraints define the acceptable performance which the selected task must be capable of satisfying. Each performance constraint is defined using a performance variable with a domain of possible values. For each variable, using a constraint, the minimum acceptable value for each performance requirement is defined. For each task, constraints define, relative to state variables and execution parameters, what the task’s performance can achieve. Therefore, the constraint solver must determine which tasks can satisfy both the tasks constraints and the performance constraints.

The *risk avoidance* performance variable defines the level of risk avoidance to be maintained while the robot executes the task. Risk is defined as the likelihood that an event will occur that will result in damage to the robot, damage to a human, or damage to property.

The *timeliness* performance variable defines the minimum acceptable timeliness for performing a task. Timeliness is defined as to how quickly the robot performs a task. Timeliness does not relate to the priority of the task.

The *resource conservation* performance variable defines the acceptable utilization of resources by the mobile robot. Resources are most often defined with

respect to mobile robots as energy usage (batter, fuel, etc). However, the term “resource” was selected in order to not limit this variable to one particular form of resource. A task must be capable of operating such that its resource consumption is reduced.

In addition to the three performance variables and associated constraints described above, additional performance variables can be defined as needed depending on the robotics problem being addressed. Quality is one such performance variable that has been included in two of the three experimental scenarios that this dissertation presents. Quality is a highly subjective term and will therefore be clearly defined for each scenario as necessary.

4.3 Constraint Language

Koalog provides a set of predefined constraints. Defining a system manually using the available Java function calls would be quite tedious and non-expressive. By nesting one or more constraints together, complex constraints can be defined. However, using the Koalog Application Programmer Interface (API) , the expression of constraints may become difficult to read and syntactically complex to specify.

A constraint programming language was defined and implemented for the task selection system. The language allows complex nested constraints to be defined. These constraints resemble first order predicate calculus statements with prefix notation. A parser was written in JavaCC [39] to parse the language statement and generate the appropriate Koalog constraints.

An example of a constraint defined using this technique is follows:

$$(=> (== \text{taskVariable deliveryTaskID})(== \text{Risk deliveryTaskRisk})) \quad (4.1)$$

This constraint states that the task selected variable equaling the delivery task's ID implies the risk avoidance performance variable must equal the delivery task's risk avoidance variable.

Comparison constraints define the relationship between two or more integers using the relationships: equals, not equals, greater than, greater than or equals, less than, and less than or equals. These constraints can be stand alone assertions or Boolean comparison. For instance, $(> a b)$ asserts that the variable a must be greater than b . The constraint $(== c (> a b))$ states that c is equal to the Boolean result of the comparison $(> a b)$.

Logic constraints define common logical relationships including: and, or, implies, if and only if, and not. These constraints can be defined as assertions, or as part of a nested constraint. For instance, $(=> a b)$ asserts that a implies b . The constraint $(=> a (\&\& b c))$ is a nested constraint in which the “and” constraint's result is utilized as part of the implication constraint.

Arithmetic constraints such as add, subtract, multiply, and divide are implemented in order to compare either three variables or compare two variables and a constant. For instance, the following constraint $(== a (+ b c))$ states a is equal to the sum of b and c , and $(== a (+ b 3))$ states that a is equal to b plus three.

Relations are also constraints that are supported. A relation defines, for a given set of two or more variables, the valid set of assignments relative to one another. For instance, $(== \{a, b\} \{\{1,2\}, \{3,4\}, \{5,6\}\})$ state that the only

valid co-assignments for variables a and b is the given choices.

4.4 Linguistic Data Representation

A constraint satisfaction problem's computational complexity increases dramatically as the size of the CSP grows. Reduction of the variable domains within the CSP has a significant impact on the solution time. Algorithms to filter and reduce the domains are often as computationally inefficient as the backtracking search algorithm itself. Therefore, to address this issue, all data used within the framework is linguistically specified or enumerated rather than given a continuous integer domain.

Assigning linguistic values to continuous variables is quite common in fuzzy logic and fuzzy sets. Since fuzzy CSPs are outside the scope of this research, the constraint variables are crisp instead of fuzzy. The mapping of continuous variables to linguistic variables are done explicitly for the task, within the Java class definition or a configuration file, or they are done using a simple mapping function. For example, given a known or specified minimum and maximum range for a particular variable, its data range is divided into n evenly spaced sets where n is the number of linguistic values to be assigned. For each continuous value, the appropriate set is looked up and its linguistic value's enumeration is returned. Most often a set of five linguistic values is used: {LOW, MED_LOW, MED, MED_HIGH, HIGH}, which are enumerated {-2, -1, 0, 1, 2}. The enumerations not only represent the linguistic value within the CSP, mathematical relationships can be developed using the enumerated value of a linguistic variable. For conversion of linguistic values for control, such as speed, an inverse mapping can also be implemented as a factor, or it can be explicitly defined.

4.5 Tasks

A software application programmer interface (API) was developed to produce a common data structure for representing the tasks and their constraint models that worked together with the decision maker. It provides a bridge between the decision maker and the mobile robot.

4.5.1 Active and Inactive Tasks

Tasks can be designated as active or inactive, depending on the current state of the system. For instance, if the robot’s manipulator is disabled, a “retrieve item” task should not be included in the set of potential tasks. Each task declaration implements a Boolean function *isActive*, which is true if the task is active and should therefore be considered by the decision maker. If the task is not active, its constraint model will not be built.

4.5.2 Task Variables

Each task is assigned a taskID variable, which is instantiated with a unique identifier for the task.

In addition, local instances of each performance variable are provided: TaskRisk, TaskResource, and TaskTimeliness. If the task is selected, each global performance variable must be equal to the local performance variable. This relation will be defined as a constraint within the constraint model. Control, percept, and hidden state variables can also be defined for each task.

Each task must implement the function *intializeVariables*. This function is called at every iteration. It redefines every CSP variable in the constraint model. At this time, percept variables are assigned immutable linguistic values equal to

the robot’s actual percepts.

4.5.3 Task Constraint Model

Each task must implement the *buildCSP* function, which generates a constraint satisfaction problem that contains all constraints. Each constraint is defined using the constraint language defined above. By default, the model contains the following constraints for each performance variable:

```
(=>  (==  taskVariable taskID)
      (==  taskRiskAvoidance RiskAvoidance))

(=>  (==  taskVariable taskID)
      (==  taskResourceConservation ResourceConservation))

(=>  (==  taskVariable taskID)
      (==  taskTimeliness Timeliness))
```

The constraints are often grouped into additional functions that specify the constraints for each of the respective performance variables. For each of the scenarios presented in this dissertation, the functions *addResourceConstraints*, *addTimelinessConstraints*, and *addTimelinessConstraints* were defined, among others.

4.5.4 Task Execution

Three functions are required by the Task API for the execution of a task.

The first function, *executeTask*, takes one parameter, i.e., the CSP’s solution.

The function must extract the relevant configuration data from the solution provided by the Koalog solver. The function then directs the robot and/or its components such that the task is correctly executed.

If the task is completed or safely interrupted, the second function, *handleSuccess*, is called. This function is responsible for performing necessary updates to the decision maker such as adding a followup task, or removing the task from the decision maker.

If the task terminates while incomplete, or under some error condition, the *handleFailure* function performs the necessary operations. The operations performed include adding followup tasks, or directing the robot to perform some corrective action.

4.5.5 Task Utility

Depending upon the constraint model and the constraints upon the performance variables, the constraint solver can derive multiple satisfying solutions. As with a variety of engineering problems, there could be several valid solutions, but the “best” or a “better” solution must be selected, if possible.

For each task, a utility function must be defined. Given the robot application, all possible tasks can have the same utility function. For other applications, each task may have its own unique utility function.

For each demonstration scenario in this dissertation, the associated utility functions are defined.

4.6 Task Composition

Figure 4.2 presents the relationship between various system components with respect to the composition of the constraint satisfaction problem used by the decision maker. The arcs in the graph represent the sharing of CSP variables and constraints between system components. The task selection CSP will be derived from the constraint models of the N associated tasks. The constraint models are tied through constraints and variables as shown by the arcs to the mobile robot (and/or its lower-level control system) and the global CSP variables. The global CSP variables are comprised of the *Task* variable and the set of performance variables *Timeliness*, *RiskAvoidance*, and *ResourceConservation*.

For each iteration, active tasks will construct a constraint model, given the current state, as described above. Performance variables, the task variable, and the robot's control and percept variables are shared between all models. When the models are merged, a new CSP is built. The variables for each model are added to the new CSP. Then, each of the constraints is assigned to the variables of the complete CSP. The resulting CSP contains all relations and all variables that were contained within the individual task models, and the constraints set forth by the user upon the performance variables.

4.7 Task Selection

The task selection module is responsible for coordinating the entire operation. It stores instances of each requirement variable, the constraint solver, and each task as shown in Figure 4.2.

The following is the algorithm carried out by the task selection module:

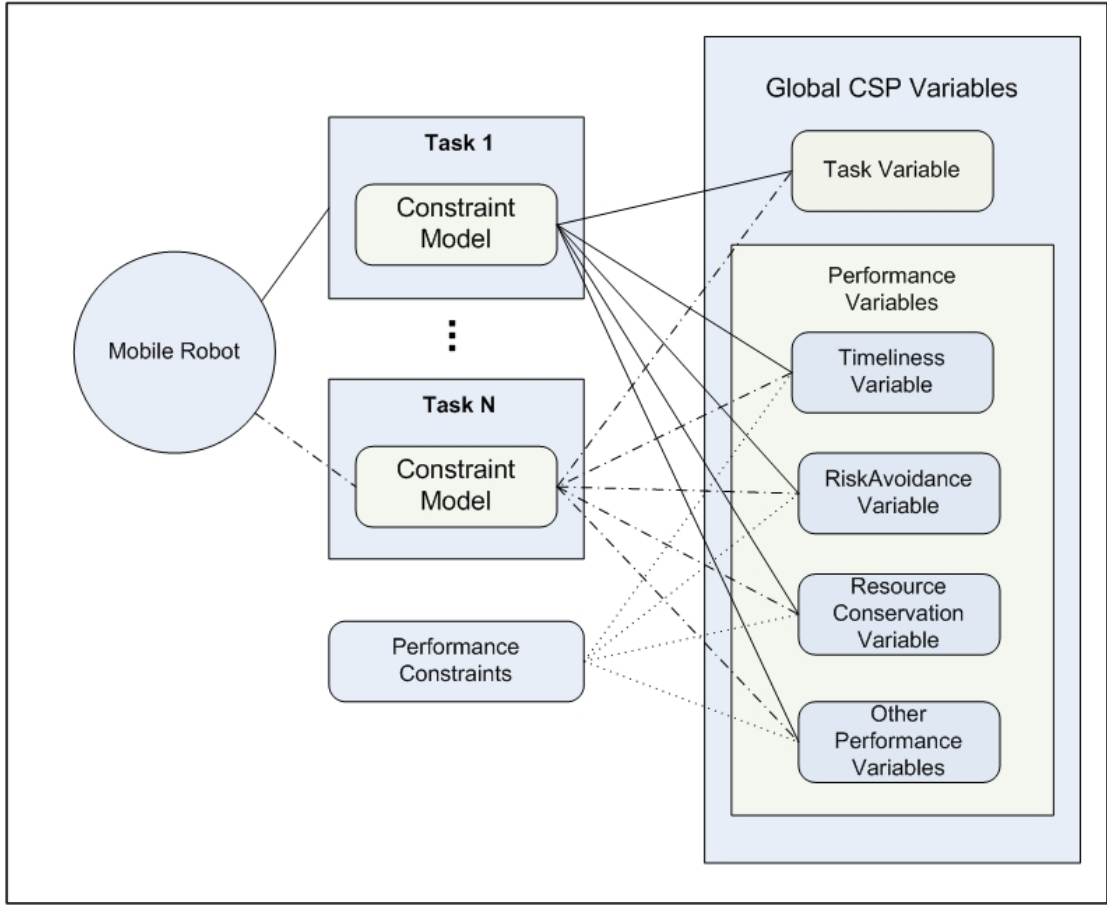


Figure 4.2: Composition of constraint data for decision maker.

1. Generate a list of active tasks.
2. Build the CSP from the set of active tasks. For each task, the task's CSP is obtained and added to the task selector's CSP. Once all task CSPs have been merged, the final CSP contains all of the variables and constraints of each task. If the same variable is used in multiple task CSPs, only one instance of the variable is added to the merged CSP.
3. Solve the problem for the first N solutions. Pick the "best" solution given some utility function.

4. If solution exists, extract the solution and execute task. Else, call the no solution handler, which responds accordingly.
5. If the execution completes successfully, the `handleSuccess` function is executed. Otherwise, the `handleFailure` function is executed.

Chapter 5

DELIVERY ROBOT

In order to demonstrate the constraint-based task selection framework a delivery robot scenario was developed and simulated. The goal of the delivery robot is to service requests to pickup items from one location and deliver them to another. A mobile robot simulation was developed based on the Pioneer 3AT mobile robot [65] equipped with a single manipulator and a payload carrier. It operates within a multi-room environment in which one or more paths exist between locations. One or more charge stations are distributed about the environment.

The robot must choose between several tasks during the delivery scenario. Each incoming request from a user is modeled as a pickup task, which is assigned to the robot's set of current tasks. Once the item is retrieved, a delivery task is assigned to the robot. The robot may also choose to perform a charge task or wait for more tasks.

5.1 The Robot

A Pioneer 3AT mobile robot, a small indoor/outdoor mobile robot, was simulated to support the delivery scenario demonstration. It is equipped with a holonomic drive with a maximum speed of 0.7 meters per second. The simulated robot possesses a single manipulator and an onboard payload carrier that supports up to three items. If the carrier is overfilled, the item in the carrier that was present the longest will be lost.

Numerous generalizations of the mobile robot are made for the demonstration and simulation. Because the tasks represent high-level control decisions, the robot is modeled at a higher-level. The robot can move from one location to another. The power consumption for a transition is static at 0.66% of the onboard battery. For charge tasks, 300 time units are required to charge or swap batteries at a charge station. The time required to move between locations is determined by travel distance and the robot's speed, discretized into five possible speeds from Low to High, over the range of 0.2 meters per second to 0.7 meters per second.

While traveling along its path, there is a probability at each location that the robot will collide with an obstacle. This probability is based on the speed of the robot and the obstruction level of the environment. For example, in a high obstruction environment, the probability of a collision is much higher for the robot travels at a high speed rather than if the robot travels at a low speed.

The robot maintains an internal map of the environment. The initial (default) state of the environment is provided at run-time. The robot can update its internal map of the the state of each location including the obstruction level, and blocked/unblocked value. Every 300 time units, the map is degraded toward the default state. This ensures that the robot's decisions are based on more recently

obtained state data. For instance, if a hallway is blocked by a mail cart, it will avoid the path temporarily, but eventually it will reconsider the path as viable.

5.2 Delivery Environment

The world for the delivery scenario is an environment similar to an office building or a hospital. The environment is represented by a topological map representing the connections between rooms and hallway locations across a single floor. Each location within the topological map is either an office or a hallway. Office locations may or may not have a charger. Rooms and hallways can be designated as blocked (inaccessible), which represents locations where the robot previously collided with an obstacle or a door that is closed, preventing access to the robot.

A rendering of the environment's topological map over its floor plan is provided in Figure 5.1. Each location is assigned a unique identifier (number). The bi-directional arcs between locations indicate valid transitions for the mobile robot. A charger is located only at location zero.

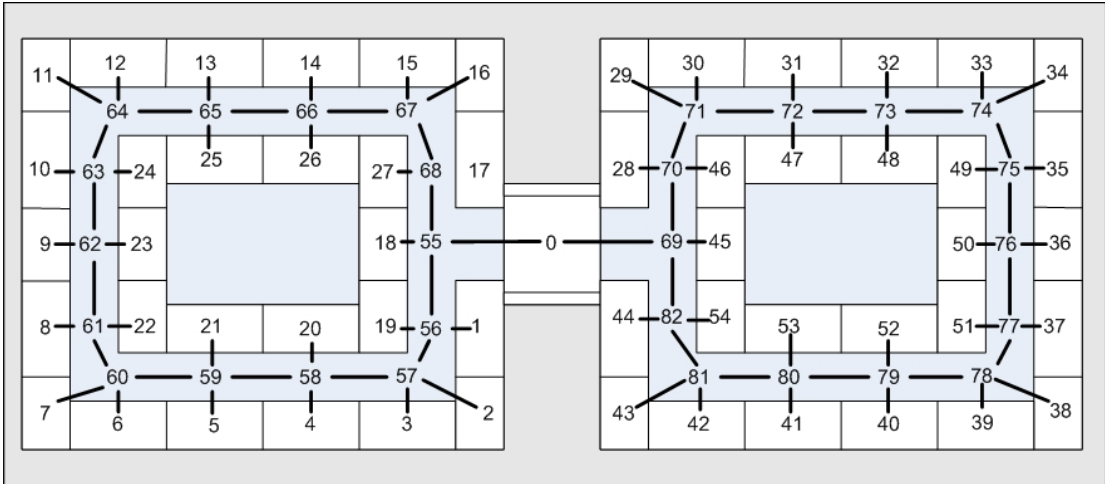


Figure 5.1: Delivery scenario: rendering of the environment.

Each location is defined with a static obstruction level (e.g., furniture) and a dynamic obstruction level (e.g., carts, humans, etc). Initially, the dynamic obstruction level is considered medium at all locations. Over time, the dynamic obstruction may increase or decrease, simulating changes within a real human occupied building. The initial map state is known by the robot at startup. The up to date state is sensed by the robot in order to build a more current map. When sensed, the obstruction level (static or dynamic) with the greatest value is returned.

A path represents a valid sequence of transitions to move from one particular location to a destination. Given the robot's current map, the obstruction level of each available path can be estimated. A path's obstruction level is the average obstruction level over all locations. Each path's obstruction level is used by the constraint-based decision maker.

5.3 Tasks and Task Constraints

Decomposing the delivery scenario, the robot must be capable of picking up an item, delivering an item, recharging itself when it runs low on power, and waiting for a new task when no tasks are active. Using object-oriented design, several layers of abstraction can be observed. At the highest level is the task interface, which defines the requirements of all tasks. The *Wait Task* implements this interface and waits for a set period of time. At the next level, the robot must move from one location to another for recharging, picking up, and delivering. The abstract task *Movement Task* defines the CSP variables and constraints associated with movement for the constraint model. The *Charge Task* directly extends the Movement Task. Pickup and delivery require the manipulator to retrieve or place

items. Therefore, extending the Movement Task, an abstract *Mobile Manipulator Task* is defined, which uses all of the variables and constraints of the movement task as well as new constraints and variables relating to the use of the manipulator. Finally, from the mobile manipulator task, the *Pickup Item Task* and the *Deliver Item Task* are derived. Figure 7.1 presents the class hierarchy graphically.

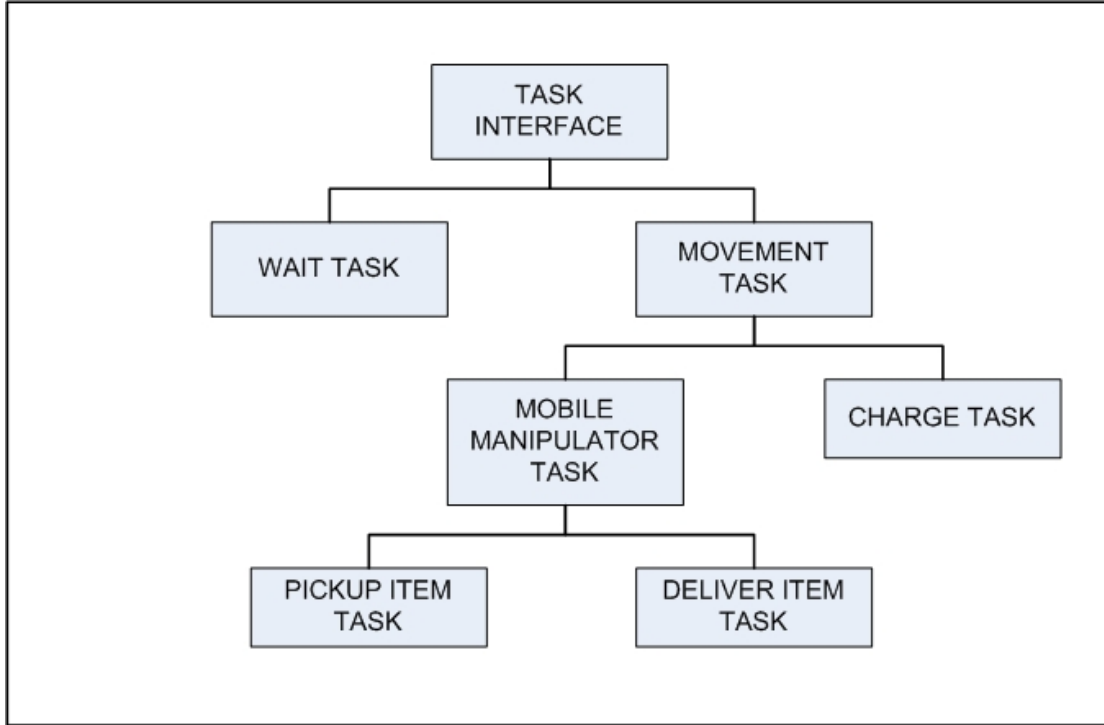


Figure 5.2: Delivery scenario: class hierarchy.

Charge and wait tasks are added to the decision module at start up. Pickup Item Tasks are added manually by the users. When adding a pickup task, the users provide a pickup location, destination, and a deadline. Deliver Item Tasks are derived from successfully executing Pickup Item Tasks. If a task has not begun execution by its deadline, it is cancelled and removed from the set of potential tasks.

5.3.1 Wait Task

The *Wait Task* is a no-op (no operation) task developed to handle situations in which either no other tasks are currently active or no other tasks satisfy the constraints of the system. When the Wait Task is executed, the system will pause for 30 time units (simulated seconds).

5.3.2 Movement Task

With the exception of the Wait Task, all other tasks are derived from the *Movement Task*. It represents an abstract task that defines the variables, and constraints for all tasks that require movement of the mobile robot. The Charge Task is directly derived from the Movement Task.

In addition to the task performance variables, several additional CSP variables are defined. The variables *PathVariable* and *SpeedVariable* represent control parameters for the mobile robot, indicating the path to be taken and the movement speed if the task is executed. Perceptual variables also exist including: *PathLengthVariable* (storing the length of the selected path), *ObstructionVariable* (representing the obstruction level of the selected path), and *DynamicObstructionVariable* (representing the activity level of the entire building environment). Two robot error modes are also represented as boolean CSP variables: *avoidanceFaultVariable* and *driveFaultVariable*. The robot's current battery charge level is stored in the *BatteryLevelVariable*.

Numerous constraints are required to model the acceptable movements of the mobile robot. These constraints are divided into the following groups: perceptual constraints, timeline constraints, risk constraints, and resource constraints.

Perceptual constraints are utilized to set task specific variables based on robot's

state data. To prevent the robot from operating in a state in which its battery may fail, the first constraint states that if the battery is low, then the movement task cannot be selected by the solver. For each path, constraints are also utilized to handle whether a path is traversable, each path's obstruction level, and each path's length. The following constraints define these basic properties. These constraints are as follows:

1. $(\Rightarrow (== \text{BatteryLevel } LOW) (! = \text{taskVariable } curTaskID))$ (Overridden by Charge Task)
2. For each path, if $(\text{path.traversable} == \text{false})$, $(! = \text{PathVariable } path)$
3. For each path, $(\Rightarrow (== \text{PathVariable } path) (== \text{PathLengthVariable } path.length))$
4. For each path, $(\Rightarrow (== \text{PathVariable } path) (== \text{ObstructionVariable } path.obstruction))$

Timeliness constraints define how the robot must operate given the specified timeliness performance criteria. The constraints are based on the robot's speed and the level of obstruction for the chosen path. Timeliness constraints are influenced by the speed of the robot and the length of the path. These constraints are as follows:

1. $\Rightarrow (== \text{taskTimeliness } MED_LOW)$
 $(|| (> \text{speedVariable } LOW) (< \text{pathLengthVariable } HIGH))$
2. $\Rightarrow (== \text{taskTimeliness } MED)$
 $(|| (> \text{speedVariable } MED_LOW) (< \text{pathLengthVariable } MED_HIGH))$

3. $\Rightarrow (== \text{taskTimeliness MED_HIGH})$
 $(\parallel (> \text{speedVariable MED}) (< \text{pathLengthVariable MED}))$
4. $\Rightarrow (== \text{taskTimeliness HIGH})$
 $(\parallel (> \text{speedVariable MED_HIGH}) (< \text{pathLengthVariable MED_LOW}))$

Risk constraints define how the robot must operate given the risk avoidance performance criteria. The constraints are based on the robot's speed and the level of obstruction for the chosen path. Two constraints exist to handle failure modes. These constraints are as follows:

1. $\Rightarrow (== \text{taskRisk MED_LOW})$
 $(\parallel (< \text{speedVariable HIGH}) (< \text{obstructionVariable HIGH}))$
2. $\Rightarrow (== \text{taskRisk MED})$
 $(\parallel (< \text{speedVariable MED_HIGH}) (< \text{obstructionVariable MED_HIGH}))$
3. $\Rightarrow (== \text{taskRisk MED_HIGH})$
 $(\parallel (< \text{speedVariable MED}) (< \text{obstructionVariable MED}))$
4. $\Rightarrow (== \text{taskRisk HIGH})$
 $(\parallel (< \text{speedVariable MED_LOW}) (< \text{obstructionVariable MED_LOW}))$
5. $\Rightarrow (== \text{avoidanceFaultVariable true}) (< \text{speedVariable MED_HIGH})$
6. $\Rightarrow (== \text{dynamicObstructionVariable true}) (== \text{speedVariable LOW})$

Resource conservation constraints define the acceptable path length for which the robot may travel to maintain a desired level of resource conservation. These constraints are as follows:

1. $\Rightarrow (== \text{taskResources MED_LOW}) (< \text{PathLengthVariable HIGH})$

2. $\Rightarrow (== \text{taskResources MED}) (< \text{PathLengthVariable MED_HIGH})$
3. $\Rightarrow (== \text{taskResources MED_HIGH}) (< \text{PathLengthVariable MED})$
4. $\Rightarrow (== \text{taskResources HIGH}) (< \text{PathLengthVariable MED_LOW})$

5.3.3 Charge Task

The *Charge Task* is responsible for recharging the robot once its power-level falls below a critical level. The task has no deadline. Rather than have a single destination, it has multiple possible destinations so the set of paths now includes every path from the robot's current location to each charging station. The following constraint exists for the Charge Task:

1. $(\Rightarrow (> \text{BatteryLevel MED_LOW}) (! = \text{taskVariable chargeTaskID}))$

5.3.4 Mobile Manipulator Task

The *Mobile Manipulator Task* is a second abstract task that extends the Movement Task. Both the Pickup Item and Deliver Item Tasks extend the Mobile Manipulator Task. No additional constraints are added within this class.

The Pickup Item Task directs the robot toward the pickup location of an item. One additional CSP variable is added for the pickup task. The PayloadAvailable variable is a Boolean CSP variable indicating whether or not the robot has space in its payload to handle the addition of one or more items. The constraint is as follows:

1. $(\Rightarrow (== \text{PayloadAvailable false}) (! = \text{TaskVariable curTaskID}))$

If the task is selected, the robot will move to the destination and then retrieve the item. If the pickup succeeds, a new delivery task is added. If the pickup fails, the task remains. If a timeout occurs, the task is removed.

The *Deliver Item Task* also extends the Mobile Manipulator Task in order to move an item to a destination, and then uses a manipulator to place the item at that location. Constraints for this task are solely derived from the Movement Task.

5.4 Evaluation

In order to demonstrate the constraint-based decision maker’s application to the robot delivery, two experiments were performed.

The first experiment compares system performance versus task load. The system is given N delivery requests, and the mean CPU time for solving the CSPs is recorded. The number of failed delivery requests over the N requests is also determined. The number of initial delivery requests varies over 1, 2, 5, 10, 15, and 20 tasks. By varying the load of initial tasks, the impact of larger CSPs (composed of more tasks) is observed.

The second experiment tests the robot’s performance under a variety of error conditions (described below). These experiments demonstrate the CSPs resilience to change and its ability to outperform a static rule-base decision maker. Tasks for this experiment are scheduled pseudorandomly with a maximum time between tasks of 400 time units.

5.4.1 Metrics

The following metrics will be utilized when analyzing the performance of the system and rule-based systems.

- *CSP solve time / CPU time:* sum of processing time required for task selections until no further tasks remain.
- *Task execution time:* mean time (in simulated system time units) for the robot to complete each task.
- *Resources consumed:* resources consumed by the robot while executing each task (in simulated battery units).
- *Number of collisions:* number of collisions occurring for all tasks.
- *Number of failures:* number of tasks of of all tasks that fail as a result of the deadline being reached before the robot has begun a task, or a robot being unable to deliver an item due to it being lost in transit.

5.4.2 Data Collection

For each configuration, a consistent data collection process is performed. The robot simulation is carried out ten times. For each run, a task schedule is randomly generated with N tasks (where N is the number of tasks dependent upon the experiment being performed). Each task's pickup location is location zero. The delivery location and the task deadline are generated randomly.

After all ten runs are completed, using Student's t-test, the system with the best performance for each metric is determined, using 95% confidence values.

5.4.3 Robot Problems

The following problems are defined to examine the task selection mechanism's performance under a variety of unique condition.

- *Dynamic obstacles:* By default, the robot operates within an environment in which the obstruction level at each location is fixed. In this configuration mode, every 300 time units, there is a 5% probability that the obstruction level either increases or decreases by one linguistic level (e.g., from low to medium low).
- *Payload system failure:* By default, the robot can carry as many as three items at a time. In this mode, the manageable payload is reduced to one.
- *Drive system Failure:* In this state, the robot's drive speed output is reduced by 25% and the resource consumption is increased by 25%. The adjustment of 25% for each of these penalties was selected arbitrarily to represent a reasonable degradation of performance given an error.
- *Avoidance Sensor Degradation:* The avoidance sensor still works, but not as reliably. Therefore, the probability at a given location that a collision occurs is increased by a factor of 5. This factor was selected arbitrarily to represent a significant increase in the probability of a collision given the robot's inability to sense its environment.

The CSP is configured to run at medium-high for all performance objectives. Empirical tests showed that this configuration provided the lowest number of delivery failures over most configurations of the robot task.

5.4.4 Rule-based System

The constraint-based system is evaluated against a rule-based task planner (control group). This task planner bases its decision on the current state and a fixed set of rules.

1. Task selection: Tasks are selected from the order in which they are scheduled.
2. Path selection: The shortest path is selected.
3. Speed: To avoid collision, the robot will always drive at medium-low speed.

5.4.5 Computing Platform

These experiments were run on a MacBook Pro with a 1.6Ghz Intel Core Duo processor and 512MB of RAM. The platform ran Java version 1.4.2.

5.5 Results and Analysis

This section presents the experimental results and analysis when applying the decision maker to the selection and configuration of tasks for a delivery robot.

5.5.1 Task Load

One issue with using a CSPs as a robot decision maker is its computational inefficiency as a problem scales in size. Figure 5.3 confirms these results by showing the mean decision time for the CSP and rule-based systems versus the number of simultaneous initial delivery requests. As the number of simultaneous requests

increases, the scale of the CSP problem increases. The rule-based system's computation time is consistently low near the x-axis of the plot.

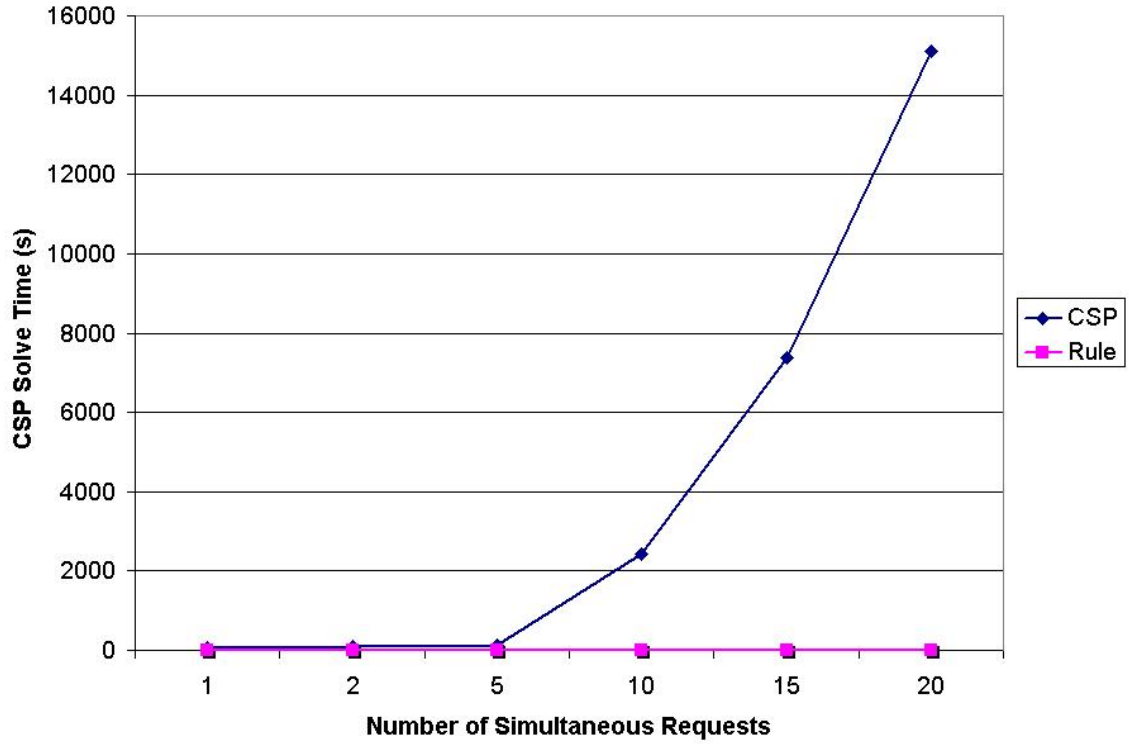


Figure 5.3: Delivery scenario: mean CPU execution time versus number of simultaneous tasks.

Contrary to these results, however, the number of failures of the constraint-based system was much smaller than the rule-based system under these conditions. Figure 5.4 illustrates the percentage of task failures while satisfying all of the initial requests. The results clearly show that the constraint-based system was more capable of selecting and configuring tasks, such that a greater number of tasks were completed prior to their deadline. Statistical analysis verified at at 95% confidence that the CSP had fewer failures when the number of tasks was greater than two.

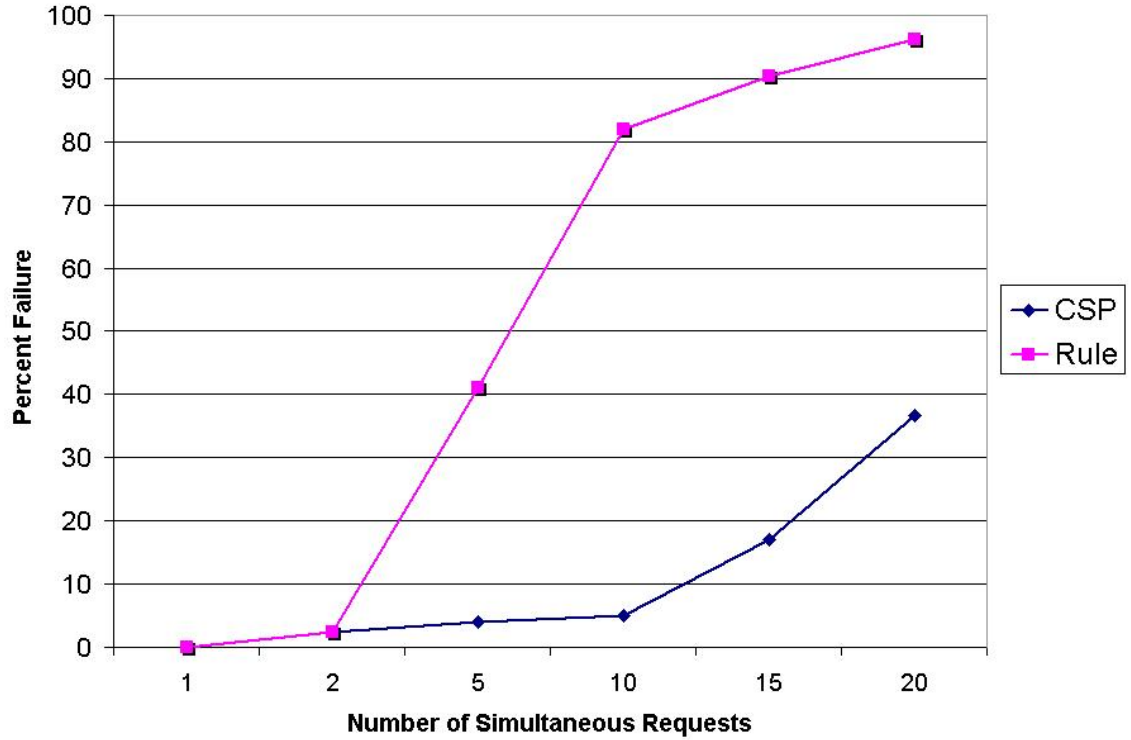


Figure 5.4: Delivery scenario: mean failure percentage time versus number of simultaneous tasks.

Clearly, a trade-off exists with respect to computational complexity and performance. As computer processors increase in speed and constraint satisfaction algorithms improve in efficiency, the computational penalty of a constraint-based system will be eliminated and its virtue will become more apparent.

5.5.2 Failures

Failures are caused by the robot being unable to deliver an item by its deadline, or the item being lost during transit. Figures 5.5 and 5.6 present the comparison of the number of failures for the rule-based and constraint-based systems with static obstacles and dynamic obstacles respectively. For each problem, the left

column presents the performance of the constraint-based decision maker and the right column presents the result for the rule-based system. Experiments were first run using the static obstruction in order to obtain a baseline over a variety of problem modes. Once completed, a smaller set of problem modes was used for the dynamic constraints. Between the dynamic and static obstruction modes, the ratio between the constraint-based and rule-based systems remain consistent.

These results were statistically analyzed and at a 95% confidence level. Examining the error modes, the failure rate was highest when a payload failure occurred. This was an example in which a single failure resulted in a significant failure of the overall system. It would be more difficult to concisely capture and handle these error states in a large-scale mobile robotic system.

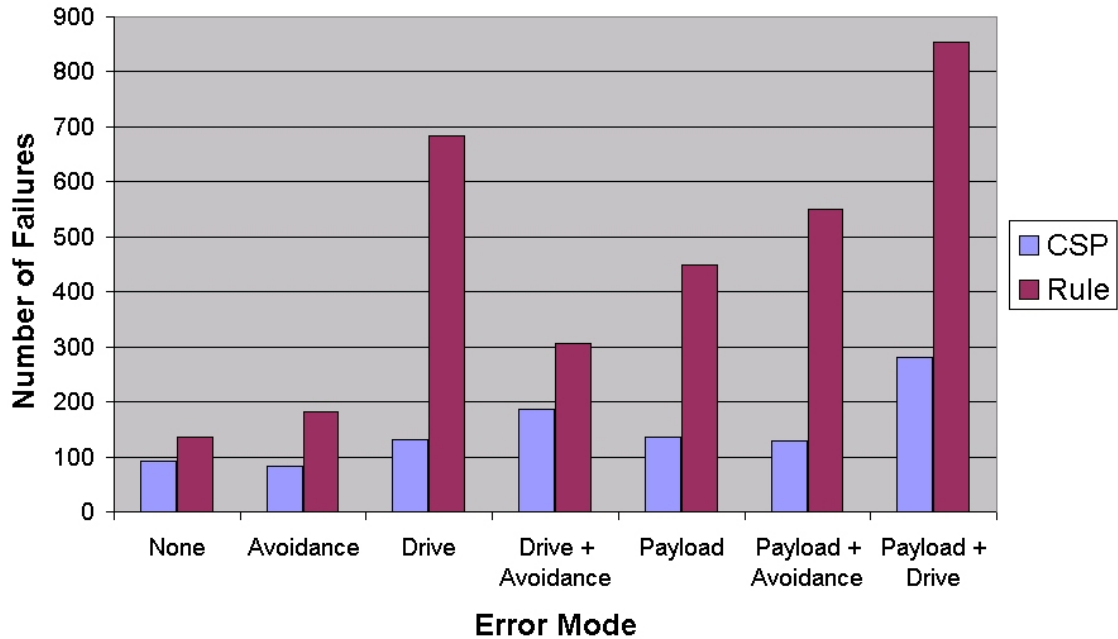


Figure 5.5: Delivery scenario: number of failures versus error mode (5000 requests and static obstruction).

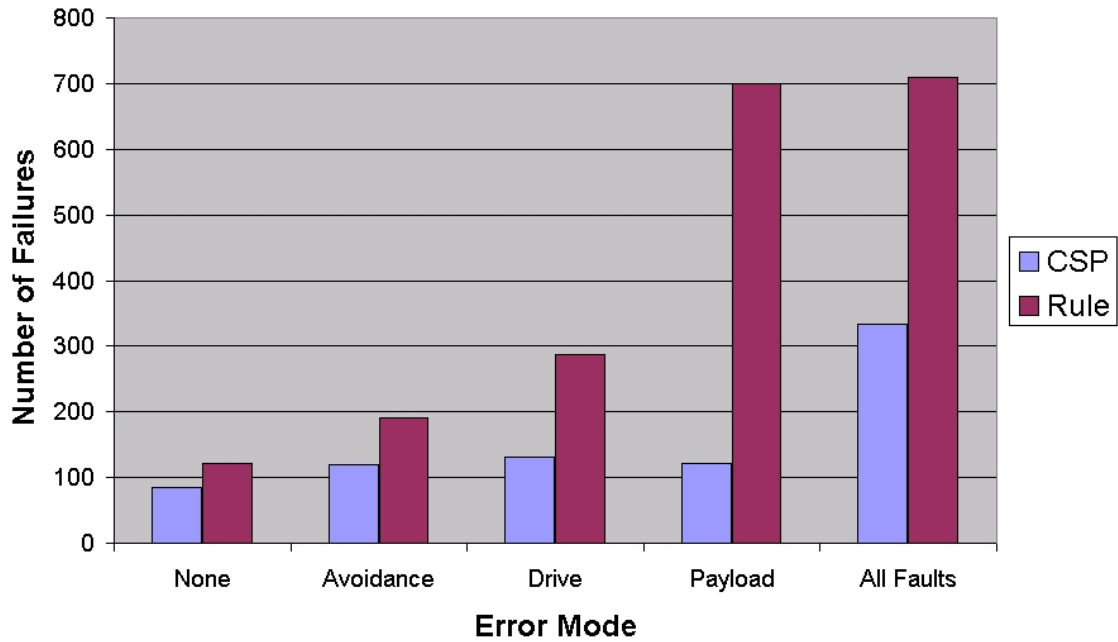


Figure 5.6: Delivery scenario: number of failures versus error mode (5000 requests and dynamic obstruction).

5.5.3 Collisions

Figures 5.7 and 5.8 present the comparison of constraint-based versus rule-based with respect to the number of collisions that occur given a variety of error modes for static and dynamic obstruction respectively. The mean number of collisions per request reflects the effectiveness of a system to avoid collisions. At a 95% confidence level, the constraint-based system outperformed the rule-based system. Often the rule-based system had over 50% more collisions. Both systems did their worse when the avoidance sensor's accuracy was diminished due to some failure.

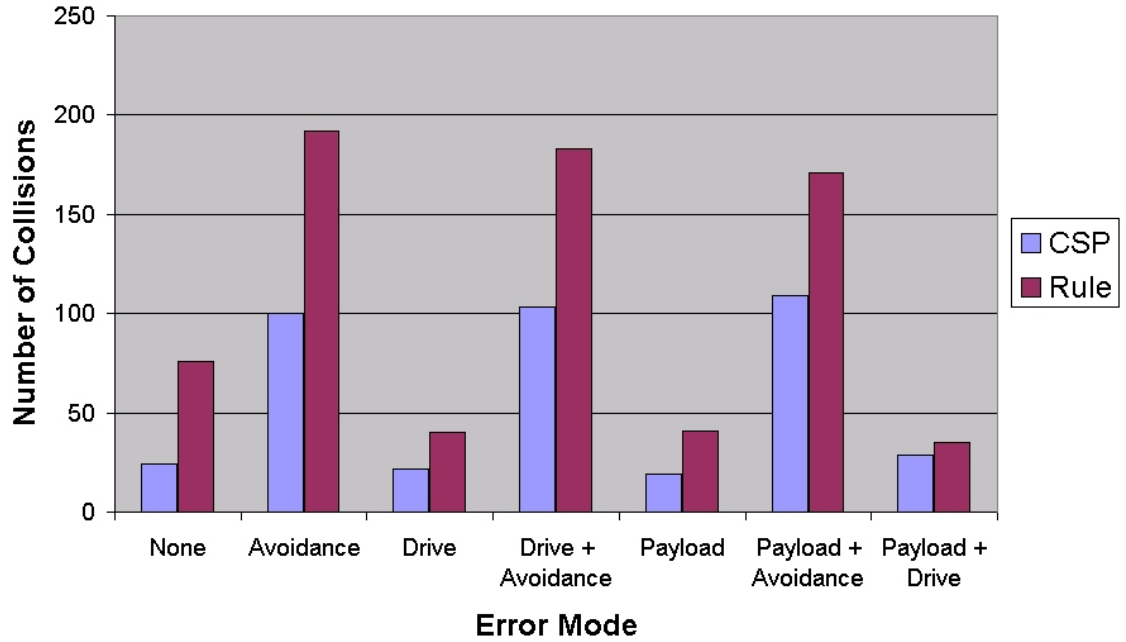


Figure 5.7: Delivery scenario: number of collisions versus error mode (5000 requests and static obstruction).

5.5.4 Service Time and Resource Consumption

The resource consumption results versus error mode for the rule-based and constraint-based systems are shown in Figures 5.11 and 5.12. The execution time results are included in Figures 5.9 and 5.10. The tradeoff of low collisions is that the timeliness and resource consumption for the constraint-based system is greater than that of the rule-based system under all configurations. While collisions do present a time penalty, the frequency of such collisions occurring is quickly averaged out over time. The rule-based system always picks the shortest path available for the task it executes. Therefore, the resource consumption levels and the execution time is at its lowest. The constraint-based system chooses tasks and its path over a variety of constraints, which seldom lead to the shortest path.

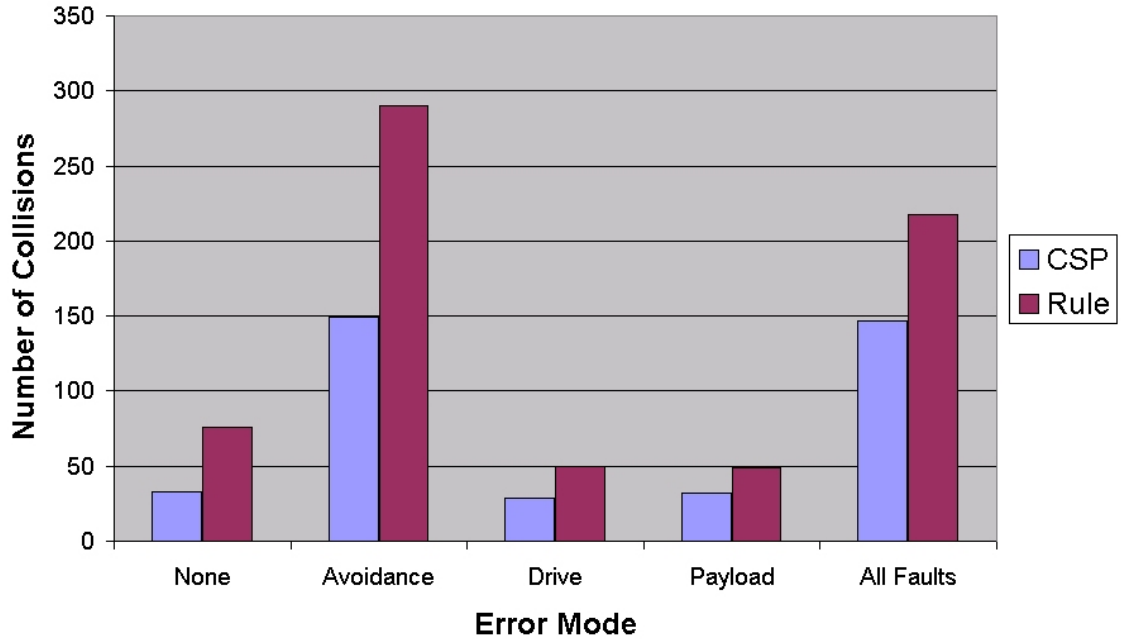


Figure 5.8: Delivery scenario: number of collisions versus error mode (5000 requests and dynamic obstruction).

The resource consumption difference between rule-based and constraint-based, while statistically significant, is still quite small at less than 1.5% of the battery difference.

5.5.5 Discussion

The constraint-based task selection mechanism was implemented for a delivery robot, and its performance was compared with a rule-based task selector. System quality was determined by the number of failures occurring due to missed deadlines or items lost in transit. The mean execution time, mean number of collisions, and the mean resource consumption were also measured and compared. The constraint-based approach generated significantly fewer task failures than the rule-

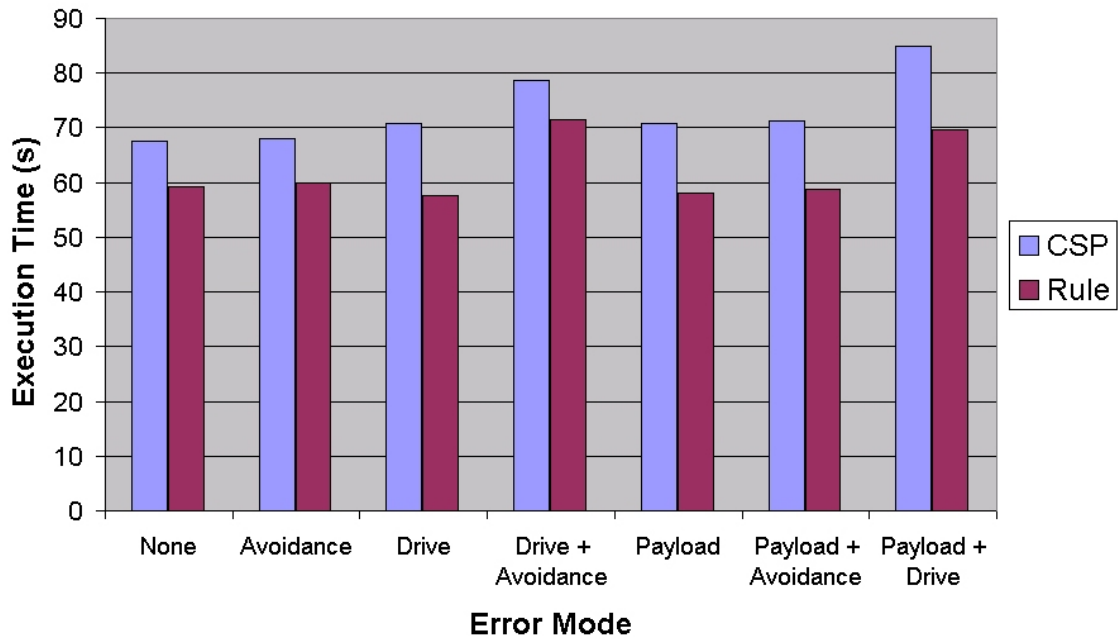


Figure 5.9: Delivery scenario: mean execution time per task versus error mode (static obstruction).

based system. It also generated fewer collisions under a variety of configurations.

When operating with a higher task volume, the constraint-based system consistently had fewer failures, but also required significantly more time to make each decision. This demonstrates one of the major tradeoffs between using the constraint satisfaction problem versus rule-based decision making. The rule-based system was capable of executing its tasks faster and utilizing fewer resources. The constraint-based system was better capable of making decisions such that time and resource usage were tradedoff in order to minimize the number of collisions and provide a higher success rate.

In rule-based systems, the performance may vary dramatically between implementations. Rule-bases that encompass all problem domain knowledge will

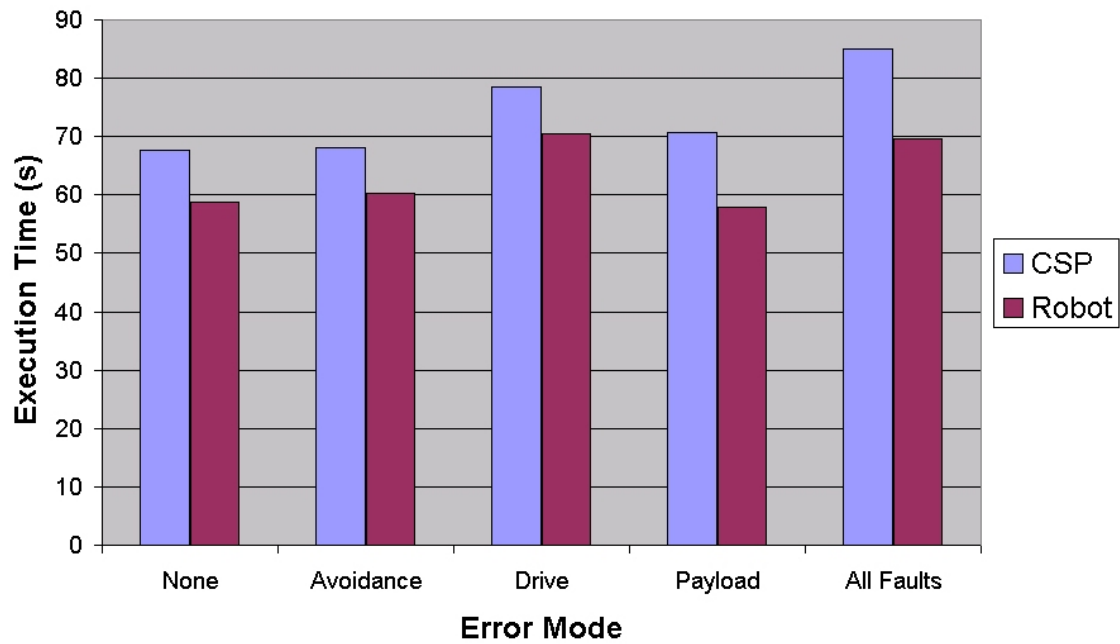


Figure 5.10: Delivery scenario: mean execution time per task versus error mode (dynamic obstruction).

out perform many competing decision makers with much less computational intensity. For this scenario, the rule-base was implemented such that under default conditions it would possess enough state knowledge to provide competitive performance with the constraint-base. Throughout this research, a best effort was made to creating a strong control group.

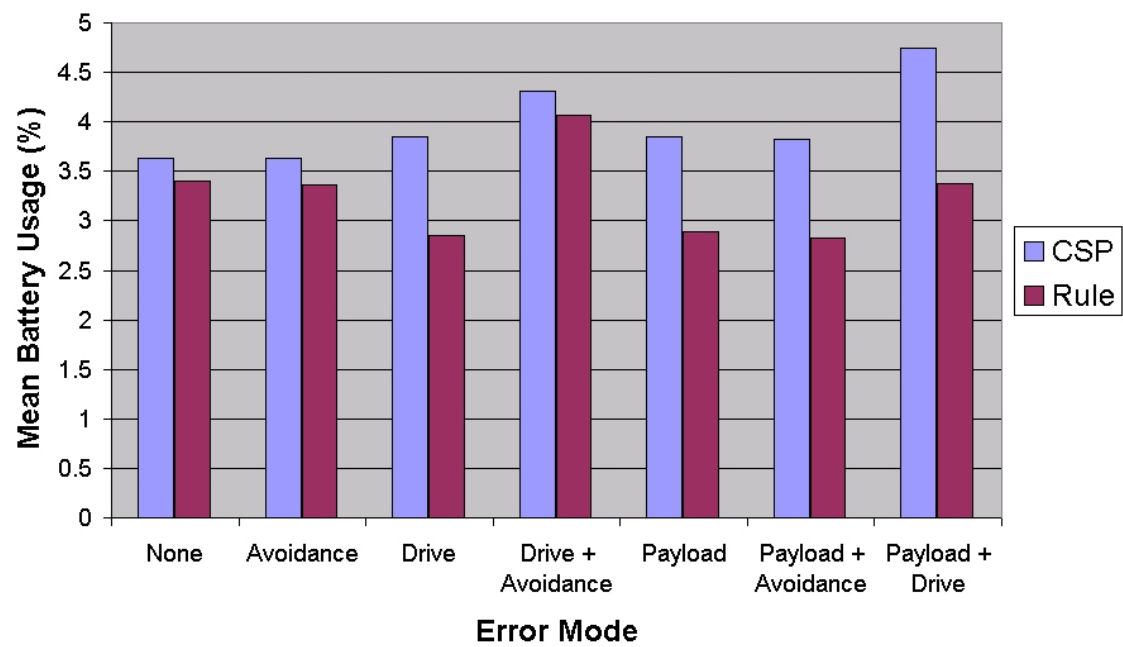


Figure 5.11: Delivery scenario: mean battery usage per task versus error mode (static obstruction).

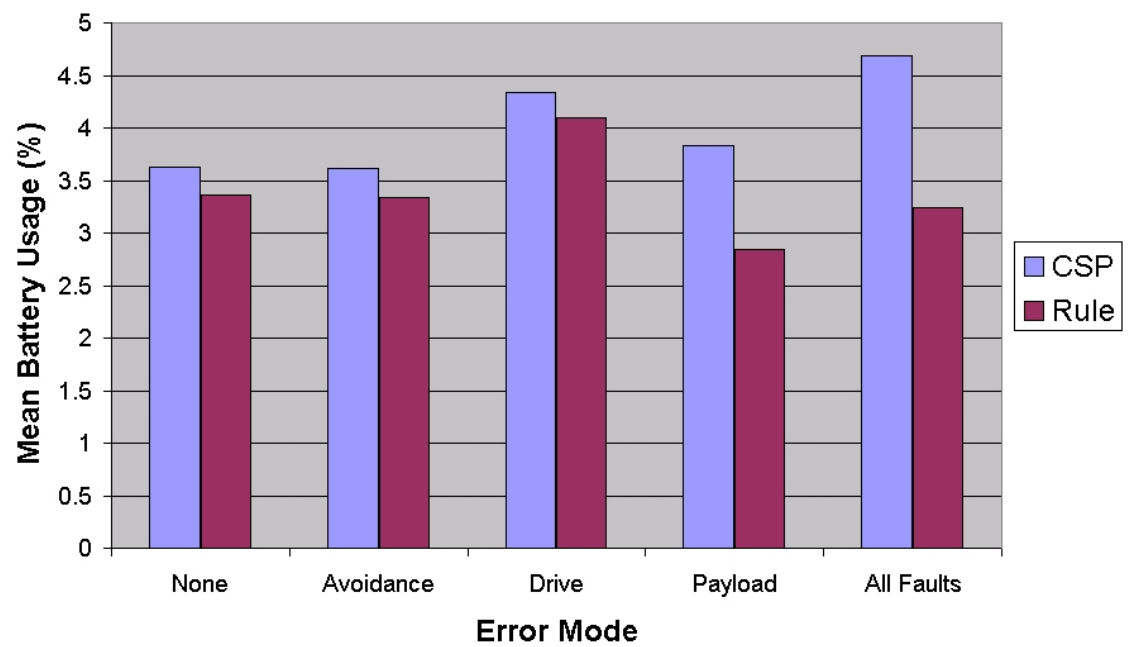


Figure 5.12: Delivery scenario: mean battery usage per task versus error mode (dynamic obstruction).

Chapter 6

POLAR ROBOT

Mobile robots are becoming more common for autonomous activities in polar regions. These polar robots face challenges that are not common for conventional mobile robots. As the threat of global warming becomes a greater concern, there is an increased need to perform research in the polar regions. Robots are therefore necessary to lessen the burden, improve the safety, and decrease the human footprint of polar field expeditions [14]. The challenge of long-term survival is possibly the greatest issue of polar robotics that must be addressed. The constraint-based task selection mechanism can be a unique method for choosing which task for the robot to execute at a given moment and the best configuration for its execution in order to ensure that the robot is capable of completing a majority of the assigned tasks possible for given conditions.

6.1 Background

In this section, background related to applications of mobile robots for polar research and the associated challenges are presented.

6.1.1 Applications and Examples

There are numerous polar research activities that can benefit from the utilization of one or more mobile robots. A workshop conducted by leading polar and robotics researchers enumerated several such potential applications [14]:

1. Traversal along a very detailed, hazardous, precise, or tedious route.
2. Research conducted outside of the typical field season during the polar night.
3. Data collection for instruments that require Low traversal speeds.
4. Augmentation to manned mission for parallel data collection activities.
5. Collection of clean zone measurements in which human presence must be limited or nonexistent.

The Polar Radar for Ice Sheet Measurement (PRISM) [60] project and the Center for Remote Sensing of Ice Sheets (CReSIS) [24] at the University of Kansas are conducting research into robots for polar traversal [77]. For PRISM, two mobile robots were developed by automating existing mobile platforms. The robots were designed to tow radar systems across the Greenland and Antarctic ice sheets along precise and tedious paths in order to obtain a variety of images of the ice sheet and its basal conditions. Under CReSIS, the deployment of one or more mobile robots will be used to carry out an autonomous seismic survey [33].

Carnegie Mellon University (CMU) constructed Nomad [53] for the Atacama Desert in 1997. It was then winterized and deployed in Antarctica to assist in the collection of meteorites on the polar surface. Such an activity is quite tedious for humans and also analogous to activities that a Mars rover might perform.

Hyperion, constructed by CMU, was a robot designed to operate solely on solar energy [85]. Its body was light weight and equipped with a large solar panel. Its movement was synchronized with the sun in order to maintain continuous operation. Continuous operation without use of a combustion engine is a major challenge for operating within the polar environment autonomously for an extended period of time. Hyperion was tested in the Canadian Arctic on Devon Island, which is considered similar to the Mars surface.

The Robot Antartico di Superficie (RAS) [9] was developed by the Italian National Agency for New Technology, Energy and Environment (ENEA) by automating a Snowcat tracked vehicle. Traversal between coastal base camps and field camps can be dangerous due to crevasses. The robot was equipped with a ground penetrating radar for crevasse detection, computer vision for path following, GPS for precise navigation, etc. It could operate as a stand along vehicle, or follow a lead vehicle.

Dartmouth constructed the Cool Robot with United States Army Corp. of Engineers [63]. This mobile robot was equipped with a solar panel covered shell that provided the robot with the power necessary to operate. Research was conducted regarding the kinematics and dynamics of operating on the polar terrain. Additional research was based on the development and analysis of the solar power system. The Cool Robot may be utilized for autonomous data collection with limited to no supervision.

6.1.2 Long-term Survivability

In [1], the challenges that exist regarding the autonomous long-term survivability of a polar mobile robot were discussed. The constraint-based task selector is used

in order to address several of these issues. These issues also define requirements for the simulation of the mobile robot within a polar environment for this research.

Mobility: It is important that the robot remain mobile at all times. Due to the High level of blowing snow, the robot moves periodically even when no task is assigned, in order to avoid becoming buried in a snow drift. Surface characteristics may vary dramatically throughout the season and from one location to another. During warm summer months, the robot may encounter softer snow as a result of surface warming. Throughout the season, snow drifts may also yield softer snow. Erosion from the wind may generate sastrugis, or hard dune-like obstacles in the snow that can result in rough terrain. It is essential that the robot not only remain mobile, but also to consider the surface conditions that can influence its mobility. If the robot gets stuck in the snow, it must be capable of moving itself out of it.

Temperature control: Internal temperature regulation is essential to maintaining continued operation of a polar robot and all its on-board components. As with the Mars rovers, a heated instruments box must be included that is thermostat controlled. Failure to heat non-hardened equipment may result in failure. For instance, a radar systems on the PRISM polar robot failed to operate successfully in Antarctica because it was not adequately heated within its enclosure [1].

Power management: Adequate renewable power is essential for autonomous long-term survivability. This work specifically excludes using gas or diesel generators because of the mechanical and logistical difficulties of autonomous refueling. The battery of the robot must be heated to a suitable temperature

to prevent accelerated discharging or damaged as the result of the cold climate.

Robots such as Hyperion [85] or Dartmouth's Cool Robot [63] utilize solar energy to sustain the mobile robot for operation. Hyperion utilized a light weight platform and large sail-like solar panels and drove such that it received maximum solar energy along its desired route. Cool Robot was designed with a box-like shell covered with solar panels so that it receives energy from the sun as well as reflected energy from the surface. The greatest limitation of using solar energy for the long-term survival of a polar robot is the polar night in which absolutely no sunlight is available for months. Redundant power sources are essential.

The intelligent coordination of robot activities with its available power is essential for success and long-term survival. If the robot becomes immobile or unable to warm its internal components, it may not be able to complete its mission even if suitable power became available at a later time.

Survival during periods of in-operability: Conditions exist such that the polar robot cannot safely operate given the current power availability, weather or terrain. The robot must be capable of surviving during these periods. At all times the robot must be able to avoid drifting snow, conserve power, and continue to heat its components.

Mechanical issues: For autonomous long-term survival, the mobile robot must be capable of continuous operation without maintenance over the period of many months. Therefore, autonomous operation must be considerate of mechanical limitation that exist and attempt to avoid conditions that place excessive wear and tear on the robot's mechanical components. For instance, operation at higher speeds over rough terrain will likely increase the wear on engines and

transmissions. Should the bearings receive excessive damage, the robot could become inoperable or its mobility may be impaired.

6.2 Polar Environment Simulation

The polar environment was simulated as accurately as possible in order to recreate some of the long-term survivability challenges described previously.

6.2.1 Climate Simulation

Throughout the simulation, the date and time are updated. The current month is used in order to simulate the current climate conditions. A maximum and minimum value exists for each simulated climate condition, namely, wind speed, temperature, and solar irradiance. Table 6.1 is the almanac used for this polar scenario based on [16] and [63]. Using this almanac, a floating point number is randomly generated for each of the climate conditions within the boundaries. The minimum wind speed is zero meters per second and not included in Table 6.1.

Visibility cannot be clearly modeled. Since the thermal spectrometer does not require visible light, the primary issue of visibility is blowing snow. The current visibility is randomly generated at some linguistic value from Low to High.

6.2.2 Terrain Simulation

Due to a variety of conditions, the terrain of the polar ice sheet can be a firm smooth surface, a rough surface, or a soft surface. Rough surfaces often result from higher levels of wind erosion to form sastrugi, which can have heights of up to one meter. Softer surfaces exist in conditions in which a large amount of snow

Month	Max Wind (m/s)	Min Temp. (c)	Max Temp. (c)	Min Solar (W/m^2)	Max Solar (W/m^2)
January	9	-38	-20	255	460
February	9	-52	-26	120	390
March	9	-62	-42	0	180
April	11	-79	-46	0	0
May	14	-75	-35	0	0
June	10	-69	-37	0	0
July	14	-74	-44	0	0
August	12	-73	-47	0	0
September	11	-75	-46	0	55
October	10	-66	-39	55	390
November	14	-48	-28	200	438
December	9	-34	-21	296	480

Table 6.1: Polar climate almanac for simulation.

has formed drifts, or during warmer days in which the surface softens due to the increase in temperature. To simulate these changes, one of the three conditions is selected at random to represent the terrain that lies ahead of the robot for its traverse. The robot must always make rational decisions regarding how to handle the current terrain.

6.3 The Robot

This section describes the simulated polar robot. The robot's onboard components such as scientific instruments, the drive system and power systems are defined so that they can be modeled. At a higher abstraction level, the robot's percepts are defined, but not associated to a particular hardware component or sensor.

6.3.1 Remote Sensing Instruments

A variety of scientific instruments for remote sensing have been included as part of the simulated polar robot. Several are listed as scientific sensors for the Center for Remote Sensing of Ice Sheet’s science and technology requirements [17] and [18].

Synthetic Aperture Radar: A synthetic aperture radar (SAR) and depth sounding radar package is included in the simulated mobile robot. These sensors respectively map the basal conditions and internal layers of the polar ice sheet. Surveys follow a long-distance grid pattern to collect a two dimensional image of the ice sheet. The instrument package developed for the CReSIS Uncrewed Aerial Vehicle is the reference for this sensor [2]. Measurements are collected along track every five meters. Precise driving is not required so long as the robot operates within a five meter tolerance of the desired path. The antennas for these radars are towed on a sled behind the mobile robot, which limits the robot’s mobility. The sensor package is always active, consuming up to 300 Watts of power [2].

Accumulation Radar: The accumulation radar is a Low-power radar that measures shallow layer structures of the polar ice sheet. It consumes nominally 20 Watts, including computer power usage. Typically, the unit follows similar traversal areas as the SAR. The sensor is modeled off of the CReSIS accumulation radar [2].

Magnetometer: The magnetometer is used to measure magnetic anomalies of the measurement environment for which the robot is operating. Due to the sensitivity of the device and electromagnetic emissions generated by the mobile

robot, this sensor is placed on a deployable boom. The boom cannot be deployed during high wind conditions due to the risk of damage. The instrument's power consumption is approximated to 5 Watts.

Gravimeter: The gravimeter measures gravitational anomalies in the robot's operating environment. The sensor can be placed on the same boom as the magnetometer. The robot and the sensor must be stationary while the sensor collects data [2]. Therefore, for such traversal, the robot must stop and collect a measurement with an average sample time of 60 seconds allotted for stopping and allowing vibrations to dissipate. Vibration from wind may diminish the quality of the data generated by the gravimeter. The Scintrex CG-5 gravity meter [68] has been selected by the CReSIS group. When active, it consumes 4.5 Watts of power.

IR Spectrometer: Infrared (IR)spectrometers have been utilized to generate an image of the IR emission for an area. The resulting data can yield a variety of details, including surface temperature and surface composition. One potential application would be meteorite discovery. The sensor requires the robot to hold its position or greatly reduce its speed for at least 5 seconds. The Miniature Thermal Emission Spectrometer used on NASA's Spirit and Opportunity Mars rovers were referenced for the model [5]. This instrument uses 5.4 Watts of power when active. Since the sensor was placed in the Mars rover's "Warm Electronics Box," it is assumed that an additional 10 Watts will be devoted to heating the instrument in order to protect and defog the instrument's optics. The quality of the instrument's data directly correlates with the level of visibility in the environment. Therefore, it is not suitable for blowing snow conditions.

6.3.2 Power Systems

The robot's power system will be divided between a rechargeable battery; two alternative power sources, a deployable wind turbine generator and a deployable solar panel; and an emergency backup battery. When one or more of the power generators are active, the robot may charge its battery or use the additional power to reduce the load placed upon the battery.

Battery: The robot is equipped with a single on-board battery or a collection of batteries that are stored within the robot's heated chamber. The battery is initially charged and may be recharged using the surplus power provided by the onboard generators. It has a capacity of 1200 Amp-hours.

Wind Generator: A wind generator is on-board the mobile robot. Due to safety issues, a locking mechanism must be in place when the generator is not deployed. When deployed, the generator will be raised such that its turbine's center is located approximately 1.5 meters above the robot. Due to the generator's deployment and the risk of damaging the generator, the unit is not able to operate at high wind speeds. The Southwest Wind Power AIR-X [71] wind turbine is used as the reference for this component. It may generate up to 400 watts at its peak speed of 12.5 meters/second. The generator does not begin generating power unless sustained winds are greater than 3 meters per second. A non-linear equation is used to approximate power output based on information provided in [56] as:

$$p = 0.205 * WindSpeed^3$$

where p is the output power in Watts and wind speed is in meters per second.

Solar Generator: Solar panels can be quite inefficient for their size. The power output of a solar panel is the product of the percent efficiency of the solar cell and the available solar irradiance in Watts per meter squared. Therefore, the solar panels are built such that they can operate within a deployed mode to obtain one square meter of coverage, or may be closed reducing the surface area by 75%. Due to the fragile nature of solar cells and the greater potential for damage from wind while deployed, the solar panel cannot operate in conditions with greater than moderate wind speeds. An array of 25% efficient solar cells are modelled. To simplify the simulation, additional solar irradiance from reflection off of the ice sheet's surface and the variation in solar panel output due to ambient temperature are not considered

Backup Battery: A low capacity backup battery is available for one time use in order to sustain the robot's heater and internal components in the event that the power available from the primary battery is insufficient.

6.3.3 Percepts

The robot is capable of sensing a variety of internal and external environmental conditions. These values are linguistically discretized in order to reduce the state space and increase the level of abstraction. The discretization is: Low, Medium-Low, Medium, Medium-High, and High.

Climate: Wind speed, temperature, and solar irradiance are sensed by the simulated robot and used in the task selection and configuration process. The

values are discretized uniformly between the annual minimum and maximum values for each climate condition as specified in the almanac.

Terrain conditions: It is assumed that the robot can determine through a variety of sensors, the approximate terrain condition. The conditions are enumerated as: rough, smooth, or soft.

Battery Charge: The battery's current charge is provided as a linguistic percept from Low to High Wattage, divided evenly over the battery's charge range from empty to fully charged.

Generator Output: To maintain a common frame of reference for generated power, a common range from Low to High is defined. Given a generator's current output wattage, a discretized value is perceived.

6.3.4 Drive System

The simulated polar robot utilizes an electromechanical drive system. The decision maker assigns a target linguistic speed from Low to High. A lookup table is utilized to determine the actual speed and consumed power, given the linguistic value. The speed ranges from 0.16 m/s at Low to 0.8 m/s at High.

Given terrain conditions, load factors are multiplied by the power consumed because of the excess power required to maintain the desired speed. If operating in soft terrain, the robot experiences greater surface resistance and consumes 50% more power. If operating over rough terrain, the robot experiences periodic conditions in which it must boost its power by 25% in order to overcome a surface obstacle.

If the robot is damaged as described in the following section, the robot's actual speed for each value is reduced by 25%. Power consumption for a damaged drive system is increased by 50%.

The terrain's resistance load is also available and can be used to handle failure conditions. When operating under rough conditions, the load may vary from Medium to High. When operating under soft conditions, the load may vary from Medium-High to High. When operating under normal conditions, the load varies from Low to Medium.

6.3.5 Internal Temperature

In the simulation it is assumed that many of the robot's computing components are placed within a heated compartment within the robot. To maintain the internal temperature, the power consumed by the heater adjusts to counter the outside outside temperatures. A simple mapping of perceived outside temperature to heater power is defined for the simulation.

6.3.6 Damage and Failure Simulation

The robot's components and drive system are susceptible to damage. For each, if a component operated outside of its specified constraints, it receives damage. Damage is tracked and once the damage exceeds a particular threshold, the component enters a failure mode. A component's failure could result in the failure of one or more tasks during the polar field season.

Drive system: The robot's drive system may receive damage if forced to operate at high speeds under high loads (from rough terrain or soft terrain).

A motor's performance decreases through continued operation at high loads. Once the damage exceeds the motor's threshold, its performance is decreased as described above.

Magnetometer and Gravimeter: Both the gravimeter and the magnetometer are placed on a deployable boom. The boom is damaged if deployed in high winds. If enough damage is encountered, the boom becomes inoperable and neither sensor can operate.

Wind generator: The wind generator is placed on a deployable mast that will raises it to a safe level before operation. If the mast fails, the generator cannot deploy. Damage is received whenever the generator is deployed during high wind conditions that exceed the robot's peak operating range.

Solar Panel: If deployed under high wind conditions, the solar panel is damaged. If the damage exceeds the threshold, it will be unable to deploy and thus generate only 25% of the maximum possible output.

Synthetic Aperture Radar: Caution must be taken when operating with the SAR. Since the radar's antenna sled is quite heavy, is towed using cables, and has a data cable tethered to the robot, there are a number of possible points of failure. If operating under high load, strain is put on each of these components, if enough damage is sustained, the sensor will be unavailable. Therefore, under a failure condition, all synthetic aperture radar experiments would fail.

6.3.7 Assumptions

It is assumed that a physical platform exists that is capable of supporting the payload described, i.e., the mass of the onboard equipment was not considered. The power levels for the drive system were selected on an ad hoc basis. Damage tolerance before failure for each component was developed as a means of determining if and when the robot violates any of the constraints specified and provides a suitable penalty for doing so. Solar energy was derived solely from the sun and not reflections off of the polar surface. The incidence angle of the sun above the horizon was also ignored.

6.4 Constraint Model

The CSP model of the polar robot given the simulated environment and the robot was developed.. The variables are first defined for the system. Next, each task's details are presented with its constraints, execution, and utility function.

Two task classes are defined for the polar scenario. The *Charge Task* has only one instance within the CSP and is used as the default task if no other solution is found. The *Survey Task* is a general purpose task for all surveys. At startup of the simulator, a list of one or more survey tasks with a desired instrument, survey path length, and expected data quality is generated. If available, the CSP solver produces a satisfying survey task. However, if no survey tasks are possible, the CSP chooses the charge task.

6.4.1 Variables

The variables are divided into four groups: performance variables, percept variables, control variables, and hidden variables.

Performance Variables: The task's performance variables *TaskRiskAvoidance*, *TaskResourceConservation*, and *taskTimeliness* were previously described. A fourth performance variable, *TaskQuality*, is used to indicate how well the robot can perform the selected task. For Survey Tasks, these are specified a priori indicating the level of quality expected by the users. For the Charge Task, quality indicates how well the robot is capable of recharging itself.

Percept Variables: The percept variables directly correlate to the robot's precepts. All percepts have been discretized to the linguistic values Low, Medium-Low, Medium, Medium-High, and High, with the exception of terrain:

- *WindSpeedVariable* stores linguistic value of the current wind speed.
- *SolarVariable* stores linguistic value of the current solar irradiance.
- *TemperatureVariable* stores linguistic value of the current temperature.
- *VisibilityVariable* stores linguistic value of the current visibility (Low implies closer while High implies further).
- *BatteryVariable* stores linguistic value of the current battery charge level.
- *TerrainVariable* stores current terrain type of Rough, Firm, or Soft.
- *SolarPowerVariable* stores linguistic value of the current solar generator output, given the current solar irradiance.

- *WindPowerVariable* stores linguistic value of the current wind generator output, given the wind speed.

Control Variables: The control variables correspond to the task execution parameters for the survey and charge tasks:

- *ChargeVariable* is a Boolean variable such that if the solution value is true the robot must charge.
- *GeneratorVariable* is an integer variable whose value corresponds to the generator that will be deployed using the enumerated order: None, Solar, Wind, and Backup, where backup corresponds to emergency backup battery.
- *HeaterVariable* is an integer variable discretized from Low to High corresponding to the thermostat setting for the robot's internal heater.
- *InstrumentVariable* is an integer variable corresponding to the instrument that will be utilized. The variable's values are enumerated as: No-Instrument, Gravimeter, Magnetometer, SAR, Accumulation, and IR-Spectrometer.
- *SpeedVariable* is an integer variable discretized from Low to High corresponding to the robot's drive speed.

Hidden Variables: The hidden variables are not derived from percepts or control parameters. These variables are used to link multiple variables together in order to reduce the complexity of the specified variables:

- *InstrumentPowerVariable* stores the power usage of that instrument linguistically from Low to High, given the instrument selected.

- *InstrumentTimelinessVariable* stores the timeliness of data collection using the instrument, given a selected instrument.
- *InstrumentRiskVariable* stores the level of risk involved with using the instrument, given a selected instrument and current percepts.
- *SpeedPowerVariable* this stores the potential power usage level while driving at that speed, given the selected speed for the robot.
- *SpeedTimelinessVariable* stores the timeliness of driving while driving at that speed, given the selected speed.
- *SpeedRiskVariable* stores the risk associated with driving at that speed, given the selected speed and current percepts.
- *GeneratorRiskVariable* specifies the risk associated with using that generator, given the selected generator and current percepts.
- *GeneratorPowerOutput* specifies the expected power output of that generator, given the selected generator and the current percepts.

6.4.2 Charge Task

The charge task contains the constraint model for charging the mobile robot. In this section, the constraints for each generator are described. The task's performance is solely defined by quality, which is directly correlated with the power output level of the generator selected.

Constraints

For the Charge Task, only the quality performance criteria is utilized. The constraints are defined to ensure that the robot enters only safe and valid hardware states when using the power generators, in order to avoid any damage.

The first constraint states that the power output is zero, which is linguistically captured by Low. The second constraint states that the task's quality for the Charge Task is the generator's output power level. A configuration that generates more power will have higher quality:

1. $(\Rightarrow (== \textit{GeneratorVariable NONE})(== \textit{GeneratorPowerOutput Low}))$
2. $(\Rightarrow (== \textit{Task chargeTask})(== \textit{TaskQuality GeneratorPowerOutput}))$

Solar Generator: If the solar generator is not available, the task cannot utilize it. If the wind speed exceeds Medium, the robot cannot deploy the solar panels. If the solar generator is used, the generator's output power for the charge task is equal to the solar generator's available output:

1. if (SolarGenerator not available), $(! = \textit{GeneratorVariable SOLAR})$
2. $(\Rightarrow (> \textit{WindSpeedVariable Med})(! = \textit{GeneratorVariable SOLAR}))$
3. $(\Rightarrow (== \textit{GeneratorVariable SOLAR})$
 $(== \textit{GeneratorPowerOutput SolarPowerVariable}))$

Wind Generator: If the wind generator is not available, the task cannot utilize it. If the wind speed is High, the wind turbine generator cannot be deployed. If the wind generator is used, the task's generator output power is equal to the wind generator's available output:

1. if (WindGenerator not available), (! = *GeneratorVariable Wind*)
2. (\Rightarrow ($==$ *WindSpeedVariable High*)(! = *GeneratorVariable Wind*))
3. (\Rightarrow ($==$ *GeneratorVariable WIND*)
($==$ *GeneratorPowerOutput WindPowerVariable*))

Backup Battery: If the emergency backup battery is not available, the task cannot utilize it. If the primary battery's charge level is not Low, the backup battery may not be used as it is for emergency use only. If the backup battery is used, the task's generator output power is equal to the Med:

1. if (Backup not available), (! = *GeneratorVariable Backup*)
2. (\Rightarrow ($>$ *BatteryVariable Low*)(! = *GeneratorVariable Backup*))
3. (\Rightarrow ($==$ *GeneratorVariable Backup*)
($==$ *GeneratorPowerOutput Med*))

Heater Constraints: For each temperature, a corresponding heater level exists. Therefore, at colder temperatures, the heater operates at a higher output.

1. (\Rightarrow ($==$ *TemperatureVariable Low*)($==$ *HeaterVariable High*))
2. (\Rightarrow ($==$ *TemperatureVariable Med_Low*)($==$ *HeaterVariable Med_High*))
3. (\Rightarrow ($==$ *TemperatureVariable Med*)($==$ *HeaterVariable Med*))
4. (\Rightarrow ($==$ *TemperatureVariable Med_High*)($==$ *HeaterVariable Med_Low*))
5. (\Rightarrow ($==$ *TemperatureVariable High*)($==$ *HeaterVariable Low*))

Other Constraints: A constraint exists to enforce that no instrument is selected for the charge task, if no data collection is conducted.

1. $(\Rightarrow (== Task\ chargeTask)(== InstrumentVariable\ NO_INSTRUMENT))$

Execution

If the charge task is selected, the robot performs the following procedure:

1. The generator is deployed and/or activated as the active power source.
2. The robot charges for 12 hours. The robot moves forward for 30 seconds during the 12 hour period in order to avoid becoming buried within a snow drift.
3. The generator is redeployed and/or deactivated.

Utility Function

The CSP may generate one or more solutions. Each solution is assigned a utility. The utility of a charge task is equal to the charge solution's quality level.

6.4.3 Survey Task

The survey task's constraints are divided into several categories: drive constraints, scientific instrument constraints, and constraints that tie the model to the performance criteria. For the generators and the heater, the survey task utilizes the same constraints listed previously for the charge task.

The Survey Task's model possesses all of the Solar Generator, Wind Generator, Backup Battery, and Heater constraints defined for the Charge Task.

Drive Constraints

The drive constraints define the level of risk, timeliness, and power usage of the drive system, given the speed and terrain conditions.

Timeliness relates to the robot's speed at which the robot travels. If operating on soft snow at speeds greater than Medium, the risk associated with the speed is High. If operating on Rough terrain at speeds greater than Medium, the risk associated with the speed is Medium-High. If operating on Firm terrain at speeds greater or equal to Medium the power used by the drive system's linguistic value is equal to the speed's linguistic value. For example, if the robot is driving at High speed, the power consumption level is also be High. If the surface is not Firm and the speed is greater than Medium, the drive system consumes a High quantity of power. If the surface is not Firm and the drive speed is Medium, the power used by the drive system is Medium-High. If the surface is not Firm and the drive speed is Medium-Low, the power used by the drive system is Medium. If the surface is not Firm and the drive speed is Low, the power used by the drive system is Medium-Low:

1. $(= > (== TaskVar curTaskID) (== SpeedTimelinessSpeedVariable))$
2. $(= > (\&\&(== TaskVar curTaskID) (\&\&(> SpeedVariable Med) (== TerrainVariable Terrain.Soft)))) (== SpeedRisk High))$
3. $(= > (\&\&(== TaskVar curTaskID) (\&\&(> SpeedVariable Med) (== TerrainVariable Terrain.Rough)))) (== SpeedRisk Med_High))$
4. $(= > (\&\&(== TaskVar curTaskID) (\&\&(>= SpeedVariable Med) (== TerrainVariable Terrain.Firm)))) (== SpeedPowerUsage SpeedVariable))$

5. (\Rightarrow ($\&\&(\text{== TaskVar curTaskID})(\&\&(> \text{SpeedVariable Med})$
 $(! = \text{TerrainVariable Terrain.Firm}))$
 $(\text{== SpeedPowerUsage High})$)
6. (\Rightarrow ($\&\&(\text{== TaskVar curTaskID})(\&\&(\text{== SpeedVariable Med})$
 $(! = \text{TerrainVariable Terrain.Firm}))$
 $(\text{== SpeedPowerUsage Med_High})$)
7. (\Rightarrow ($\&\&(\text{== TaskVar curTaskID})(\&\&(\text{== SpeedVariable Med_Low})$
 $(! = \text{TerrainVariable Terrain.Firm}))$
 $(\text{== SpeedPowerUsage Med})$)
8. (\Rightarrow ($\&\&(\text{== TaskVar curTaskID})(\&\&(\text{== SpeedVariable Low})$
 $(! = \text{TerrainVariable Terrain.Firm}))$
 $(\text{== SpeedPowerUsage Med_Low})$)

Remote Sensing Instrument Constraints

For each survey task, a particular remote sensing instrument is specified a priori. Therefore, the instrument variable is immediately constrained to the selected instrument. Likewise, the task's CSP can only contain the constraints specified to the selected task.

This constraint states that the instrument variable is set to the enumerated value of the instrument, as specified by the user for the survey.

1. (\Rightarrow ($\text{== TaskVar curTaskID}(\text{== Instrument curTaskInstrument})$)

Gravimeter Constraints: The gravimeter has a Low instrument timeliness because the robot must pause along its track to collect vibration free measurements. Instrument power usage is equal to the gravimeter's power usage. The

risk of using the gravimeter directly correlates to the wind speed as it is placed on a boom. If the wind speed is Medium-High, the quality of the data collected from the gravimeter is Low. If the wind speed is High, the quality of the data collected for the gravimeter is Low:

1. (\Rightarrow ($==$ *TaskVar curTaskID*) ($==$ *InstrumentTimeliness Low*))
2. (\Rightarrow ($==$ *TaskVar curTaskID*)
($==$ *InstrumentPowerUsage GravimeterPowerUsage*))
3. (\Rightarrow ($==$ *TaskVar curTaskID*)
($==$ *instrumentRisk WindSpeedVariable*))
4. (\Rightarrow ($\&\&$ ($==$ *TaskVar curTaskID*)
($==$ *WindSpeedVariable Med_High*)
($==$ *InstrumentQuality Med_Low*))
5. (\Rightarrow ($\&\&$ ($==$ *TaskVar curTaskID*)
($==$ *WindSpeedVariable High*)
($==$ *InstrumentQuality Low*))

Magnetometer Constraints: The magnetometer has a High instrument timeliness. Instrument power usage is equal to the magnetometer's power usage. The risk of using the magnetometer is equal to the current wind speed:

1. (\Rightarrow ($==$ *TaskVar curTaskID*)
($==$ *InstrumentTimeliness High*))
2. (\Rightarrow ($==$ *TaskVar curTaskID*)
($==$ *InstrumentPowerUsage MagnetomterPowerUsage*))

3. (\Rightarrow ($==$ *TaskVar curTaskID*)
($==$ *InstrumentRisk WindSpeedVariable*))

Accumulation Radar Constraints: The accumulation radar has a High instrument timeliness. Instrument power usage is equal to the accumulation radar's power usage. If the robot's speed is Medium-High, the quality of the collected data is less than Medium-High. If the robot's speed is High, the quality of the data collected is less than medium:

1. (\Rightarrow ($==$ *TaskVar curTaskID*)
($==$ *InstrumentTimeliness High*))
2. (\Rightarrow ($==$ *TaskVar curTaskID*)
($==$ *InstrumentPowerUsage AccumulationPowerUsage*))
3. (\Rightarrow ($\&\&$ ($==$ *TaskVar curTaskID*)
($==$ *SpeedVariable Med_High*)
($<$ *InstrumentQuality Med_High*))
4. (\Rightarrow ($\&\&$ ($==$ *TaskVar curTaskID*)
($==$ *SpeedVariable High*)
($==$ *InstrumentQuality Med*))

Synthetic Aperture Radar Constraints: The SAR has a High instrument timeliness. Instrument power usage is equal to the SAR's power usage. If the terrain is not Firm, the instrument risk is equivalent to the speed at which the robot travels. If the robot's speed is Medium-High, the quality of the data collected is less than Medium-High. If the robot's speed is High, the quality of

the data collected is less than Medium. If the terrain is Rough, the quality of the collected data is less than Medium:

1. (\Rightarrow ($==$ *TaskVar curTaskID*)
($==$ *InstrumentTimeliness High*))
2. (\Rightarrow ($==$ *TaskVar curTaskID*)
($==$ *InstrumentPowerUsage SARPowerUsage*))
3. (\Rightarrow ($\&\&$ ($==$ *TaskVar curTaskID*)
($!=$ *TerrainVariable Terrain.Firm*))
($==$ *InstrumentRisk SpeedVariable*))
4. (\Rightarrow ($\&\&$ ($==$ *TaskVar curTaskID*)
($==$ *SpeedVariable Med_High*))
($<$ *InstrumentQuality Med_High*))
5. (\Rightarrow ($\&\&$ ($==$ *TaskVar curTaskID*)
($==$ *SpeedVariable High*))
($==$ *InstrumentQuality Med*))
6. (\Rightarrow ($\&\&$ ($==$ *TaskVar curTaskID*)
($==$ *TerrainVariable Terrain.Rough*))
($<$ *InstrumentQuality Med*))

IR Spectrometer Constraints: The IR spectrometer has a Medium instrument timeliness because it must pause briefly to take a scan of the area. Instrument power usage is equal to the spectrometer's power usage. The quality of the data collected from the IR spectrometer is equal to the current visibility:

1. (\Rightarrow ($==$ *TaskVar curTaskID*) ($==$ *InstrumentTimeliness Med*))
2. (\Rightarrow ($==$ *TaskVar curTaskID*)
($==$ *InstrumentPowerUsage SpectrometerPowerUsage*))
3. (\Rightarrow ($==$ *TaskVar curTaskID*)
($==$ *InstrumentQuality VisibilityVariable*))

Timeliness Constraints: If the instrument's timeliness is High, the task's overall timeliness is less than or equal to the timeliness of the selected speed. If the instrument's timeliness is less than High, the task's overall timeliness is less than or equal to the timeliness of the selected speed minus one (as an instrument penalty). If the instrument's timeliness is less than Medium, the task's overall timeliness is less than or equal to the timeliness of the selected speed minus two (as a larger instrument penalty):

1. (\Rightarrow ($==$ *InstrumentTimeliness High*)
(\leq *TaskTimeliness SpeedTimeliness*))
2. (\Rightarrow ($<$ *InstrumentTimeliness High*)
(\leq *TaskTimeliness* ($-$ *SpeedTimeliness* 1))
3. (\Rightarrow ($<$ *InstrumentTimeliness Med*)
(\leq *TaskTimeliness* ($-$ *SpeedTimeliness* 2))

Resource Constraints: If the instrument's power consumption is greater than Medium and the speed's power consumption is greater than Medium, the task's resource conservation is by default Low. If the battery charge is less than Medium and the task's resource conservation is Low, the task's quality is Low. The task's

overall power usage equals the sum of the speed power and instrument power usage. The task's overall resource conservation level is less than or equal to the power generated by the robot's generators (generator power output) and the overall task power usage:

1. $(\Rightarrow (\&\& (>= \text{InstrumentPowerUsage Med_High})$
 $(>= \text{SpeedPowerUsage Med_High}))$
 $(== \text{TaskResourceConservation Low}))$
2. $(\Rightarrow (\&\& (\&\& (== \text{TaskVar curTaskID})$
 $(<= \text{BatteryVariable Med_Low}))$
 $(< \text{TaskResourceConservation Med})) (== \text{TaskQuality Low}))$
3. $(== \text{TaskPowerUsage} (+ \text{SpeedPowerUsage InstrumentPowerUsage}))$
4. $(<= \text{TaskResourceConservation}$
 $(- \text{GeneratorPowerOutput TaskPowerUsage}))$

Risk Constraints: Risk constraints are written in a way that takes advantage of the enumeration of the linguistic values. When negating a linguistic value, the value at the other end of the spectrum is obtained of equivalent distance from Medium (0 = Med). Therefore, the task's overall risk avoidance level is less than or equal to the the negation of the the maximum of instrument risk level and speed risk level. For example, if the SpeedRisk=Med_High and InstrumentRisk=Med, TaskRiskAvoidance = SpeedRisk*-1 = Med_Low:

1. $(\Rightarrow (>= \text{InstrumentRisk SpeedRisk})$
 $(<= \text{TaskRiskAvoidance} (* \text{InstrumentRisk} - 1)))$

2. ($=> (<= \textit{InstrumentRisk} \textit{SpeedRisk})$
 $(<= \textit{TaskRiskAvoidance} (* \textit{SpeedRisk} - 1)))$

Quality Constraints: The Task's quality must be greater than or equal to the expected quality defined a priori. Task quality for survey tasks is equal to the quality of the data collected using the instrument given the current configuration:

1. ($>= \textit{TaskQuality} \textit{ExpectedQuality}$)
2. ($= \textit{TaskQuality} \textit{InstrumentQuality}$)

Execution

The execution of the survey task uses the following procedure:

1. Instruments and generators are activated and deployed (if necessary).
2. Robot traverse the ice, collecting data at selected speed. Every 1800 seconds, the robot checks its state and distance travelled to determine if it should continue or stop.
3. Once the target distance is reached or 86,400 seconds (24 hours) of surveying has been conducted, the traversal loop is terminated.
4. The generator and instruments are retracted and deactivated.
5. If the survey was completed, it is removed from the task set. If not, the distance remaining is noted and the task remains available on the set of tasks.

Should the robot run out of power and be incapable of sustaining itself, the experiment is terminated.

Utility Function

The CSP can generate one or more solutions. Each solution is assigned a utility. The utility of a survey task is equal to the sum of the timeliness, resource conservation level, and risk avoidance level plus an additional bonus for the task being a survey task. The bonus is a means of ensuring that satisfying survey tasks are selected over satisfying charge tasks.

6.5 Evaluation

The metrics for evaluating the success and behavior of the decision makers is presented, along with the configuration for each experimental run. A rule-based decision maker acts as the control group for the experiment. The experimental procedure for each configuration is discussed.

6.5.1 Metrics

Several metrics were defined to evaluate the polar robot, broken into two general sets: measuring the overall success and performance of the constraint-based system versus the rule-based system, and providing insight into the actions of the system:

- *Survival Time*: This metric describes the duration in days for which the robot remains operational before it fails. The maximum duration is 366 days, corresponding to a full year. The simulation ran from November 15, 2007 to November 15, 2008; and 2008 is a leap year. A larger number implies that the robot under the given configuration is more capable of surviving.

- *Failure Rate:* A Boolean values of true is choosen if the robot fails and false) if the robot survives the full 366 days as assigned. By averaging over the total number of tasks for a run, the resulting number is an approximation of the failure rate as a percent of tasks. Lower failure rates imply that the robot is better capable of surviving, and avoiding damage.
- *Percent Mission Complete:* Each tasks requires the robot to traverse over a particular path of some predefined distance with a specified instrument active. This metric is the total completeness of the mission, by taking the total distance surveyed by the robot divided by the total requested distance from all tasks.
- *Tasks Completed:* Success for this scenario is based on completeness over all tasks and not single tasks. This metric is used to see how the decision maker chooses to select and execute tasks.
- *Tasks Started:* Similar to tasks completed, this metric is used to observe how the robot makes its decisions. It is the total number of tasks in which some surveying has been performed before the simulation terminates.

As stated previously, the system will not receive a penalty for leaving any task incomplete. Overall “completeness” is based on the total distance surveyed versus the sum of the requested survey distances.

6.5.2 Experiment Configurations

Table 6.2 present eight configurations that are used to test the decision makers for the polar robot. Each configuration is assigned an identifier which is referenced throughout the results and analysis section.

ID	Num. Tasks	Task Survey Distance (km)	Instrument Select	Quality	Performance
POLAR1	5	1000	One of Each	HIGH	All Med
POLAR2	5	1000	One of Each	HIGH	Resources = Med_High
POLAR3	5	1000	One of Each	Random	All Med
POLAR4	5	1000	Random	HIGH	All Med
POLAR5	10	500	Two of Each	HIGH	All Med
POLAR6	10	500	Random	HIGH	All Med

Table 6.2: Polar Scenario Configurations for Evaluation.

The robot receives a predefined number of tasks. Each task has an equal survey distance, with an assigned instrument to use. Some configurations require that each instrument is uniformly distributed between the set of tasks. In other configurations, each task is given a randomly assigned instrument.

The quality of each task can be either assigned as High meaning maximum data quality is desired, or it may be assigned to random, meaning that some tasks may not be as constrained as others with respect to data quality.

For each of the performance variables (task timeliness, resource conservation, and risk avoidance), a minimum level is assigned. From empirical observation, setting all performance variables to Medium generates the best overall performance. To demonstrate the impact of one of these variables, one experiment assigns the resource variable to Medium-High.

The configurations POLAR5 and POLAR6 are created to determine if using a larger number of smaller distance surveys would change the performance of the system. Therefore, they possess ten tasks with survey distances of 500 kilometers rather than five tasks with survey distances of 1,000 kilometers.

6.5.3 Rule-based System

In order to compare the constraint-based system with a reasonable alternative decision maker, a rule-based decision maker was implemented. In order to maintain as great of accuracy and uphold the maximum survey quality for each instrument, the rules are defined to be specific.

1. ***Select Task Rules:***

- If charge < 40%, task = ChargeTask;
else, select task with shortest remaining survey distance and its associated instrument may be selected.

2. ***Instrument Availability Rules*** (given availability of each instrument):

- Synthetic Aperture Radar: if terrain == Firm, available.
- Magnetometer: if wind != High and terrain != Rough, available.
- Gravimeter: if wind <= Medium, available.
- IR Spectrometer, if visibility == High, available.

3. ***Select Generator*** (given availability of each generator):

- If wind < High, generator = WindGenerator.
Else if wind < Medium-High, generator = SolarGenerator.
Else if task == ChargeTask and charge < 20%, generator = Backup.
Else generator = NONE.

4. ***Select Speed:***

- If instrument == SytheticApertureRadar, speed = Medium.
Else if terrain != Firm, speed = Medium.

Else speed = Medium-High.

6.5.4 Experimental Procedure

The following procedure was performed 30 times for each of the configurations listed above:

1. Simulation begins at November 15, 2007 with full charge and in 100% working order.
2. Decision maker guides the robot to perform surveys or charge.
3. Simulation runs until the robot runs out of charge, becomes disabled due to damage, or the date November 15, 2008 is reached.

Once completed, the data is analyzed statistically using Student's T-test to compare the average performance for each metric.

6.5.5 Computing Platform

This experiments were run on a PC running Linux with four dual core AMD Opteron processors and 20GB of RAM. The platform ran Java version 1.6.0.

6.6 Results and Analysis

This section will presents the results of the experiments and the statistical analysis. In the appendix, tables 5 and 6 present the T-test results for each configuration.

6.6.1 Failure Rate

The mean failure rate for each configuration is presented in Figure 6.1. The constraint-based system outperformed the rule-base for all configurations. The failure rate for the rule-based decision maker was near 100%, while for constraints, the maximum is near 70%. For POLAR2, the performance variable for resource conservation was constrained to Medium-High. As a result, its failure rate was nearer 5%, but this improvement resulted in lower task completion as shown later. At a 95% confidence level, the constraint-based decision maker outperformed the rule-based decision maker for all configurations.

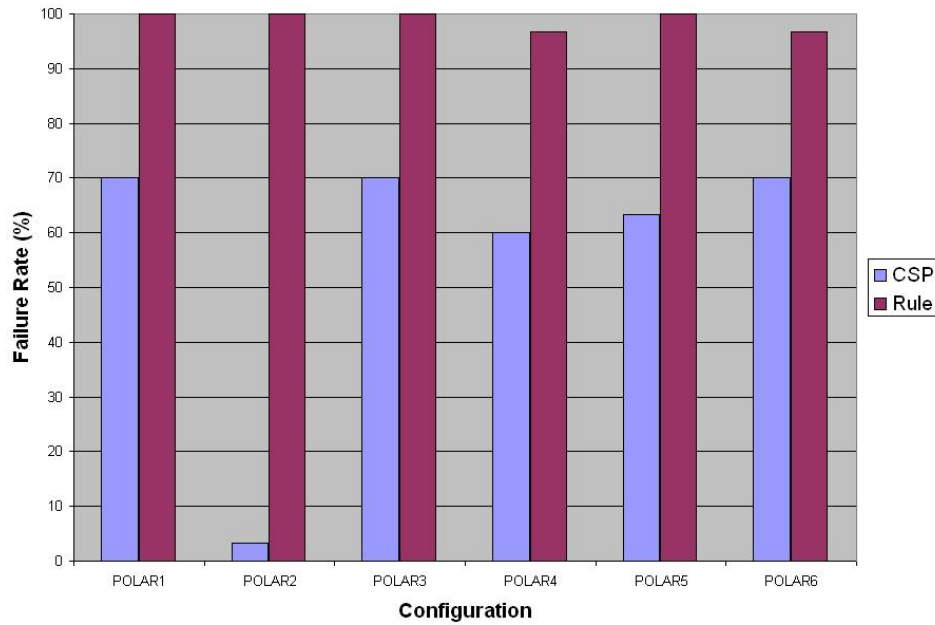


Figure 6.1: Polar scenario: failure rate versus configuration.

6.6.2 Survival Time

The mean survival time for the polarv robot is presented in Figure 6.2. This shows how long each configuration survived before failing. For all configurations, the robot’s operating time in days is fairly consistent. Under POLAR2, the performance spike with respect to energy conservation is evident with the robot’s mean survival time near a full year. The rule-based system under all scenarios failed near the half-year mark during the Antarctic winter when it must contend with colder temperatures, higher wind speeds, and no solar energy. At a 95% confidence level, the constraint-based decision maker outperformed the rule-based decision maker for all configurations.

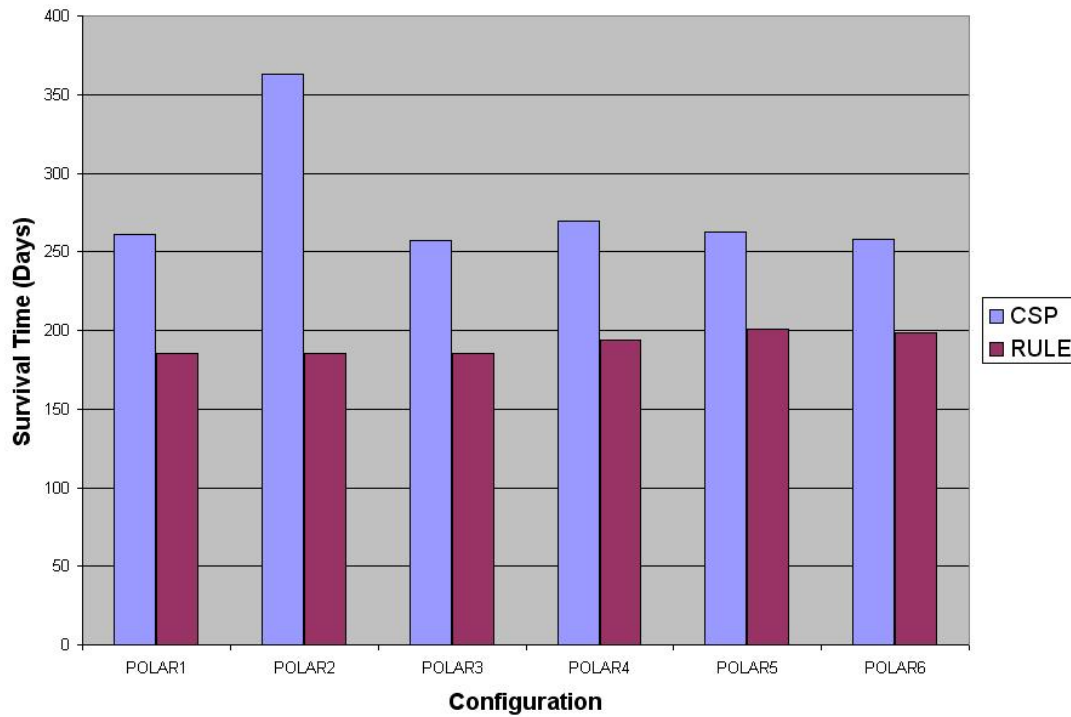


Figure 6.2: Polar scenario: survival time versus configuration

6.6.3 Mission Completeness

The mean percent completeness of the robot's mission is presented in Figure 6.3. Using constraints, the mean mission completeness for configurations POLAR1, POLAR3, POLAR5, and POLAR6 was significantly greater with means greater than 40%, compared to the mean when using rule-based which were less than 40%. The POLAR4 results were too close to conclusively determine if one system outperforms the other. POLAR2 performed much worse than the competing system as it was over-constrained toward conserving resources at a higher level. When a CSP is overconstrained, one or more variables may exist such that no solution may be resolved. Rule-based systems could be written such that no solution could be derived during a state, but often during testing such conditions are eliminated.

6.6.4 Tasks

The mean number of tasks started versus is presented in Figure 6.4. This metric provides insight into the decision maker's choice of beginning a new task versus completing a new task. Under configuration POLAR2 and POLAR4, a significantly larger number tasks were started when using rules than constraints. POLAR5 and POLAR6 yield much larger means because they have twice as many tasks. By scaling down the results, the performance would be similar.

Figure 6.5 presents the of mean number of tasks completed before the simulation ends for each configuration. From these results, one of the first conclusions is that neither of the systems completes many tasks, given the specified survey lengths and the number of tasks. When only five tasks are used with survey distances of 1,000 kilometers per task, the rule-based system completed a

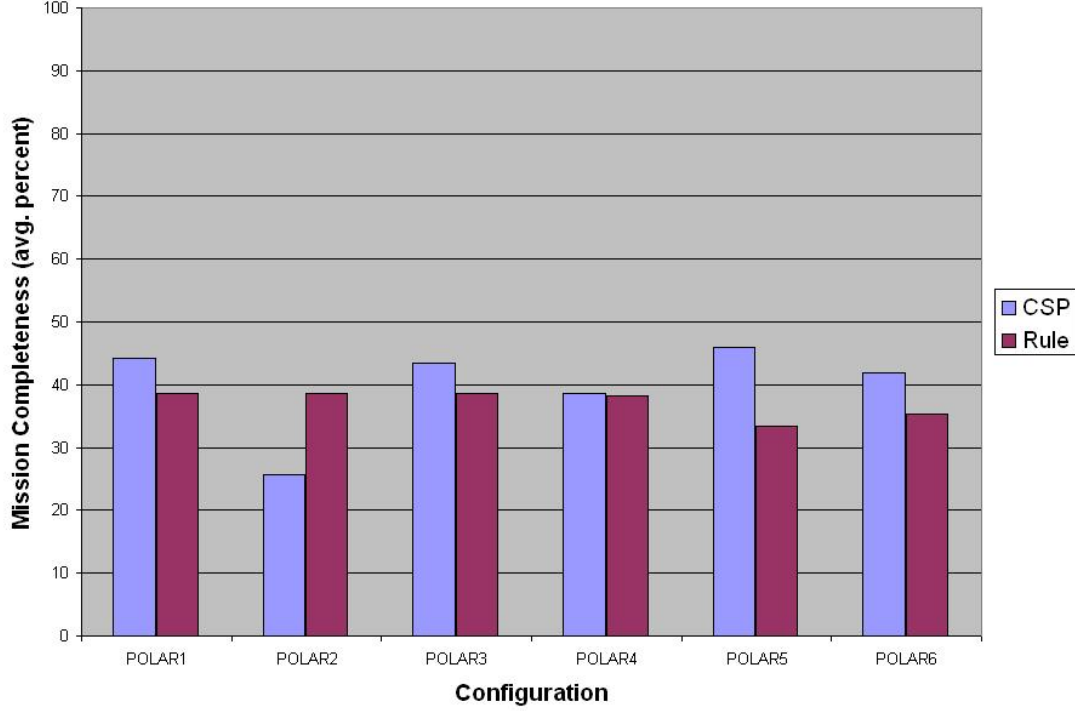


Figure 6.3: Polar scenario: completeness of total mission versus configuration.

significantly greater number of tasks versus the constraint-based system. When ten tasks with survey distances of 500 kilometers were used, the constraint-based system completed a significantly greater number of tasks.

The constraint-based system picks the task that maximizes its utility, given the current configuration. The rule-based system merely selects the task with the shortest remaining survey distance. Thus, when fewer tasks exist, the rule-based system is more tenacious toward completing one task and moving to the next task. The constraint-based system is more likely to work among a few tasks whose instruments are the most flexible with respect to risk, timeliness, and resource usage, i.e., the constraint system is more flexible and adaptive.

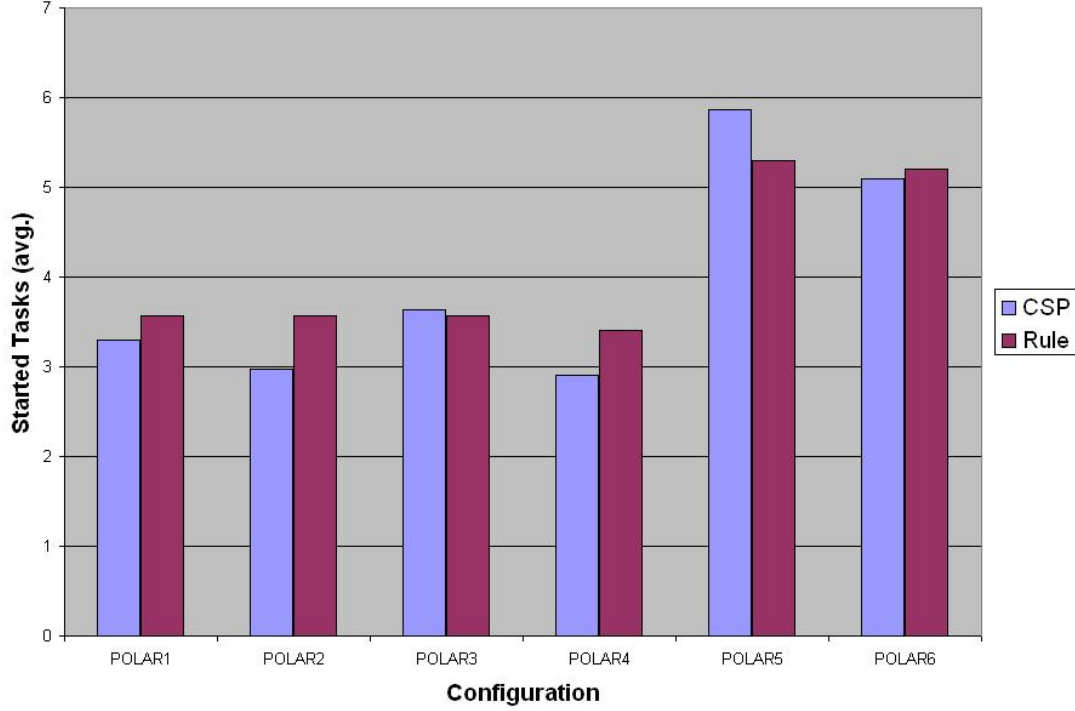


Figure 6.4: Polar scenario: tasks started versus configuration.

6.6.5 Discussion

For all metrics relating to the quality of the system’s operation, survivability and ability to complete the workload, the constraint-based decision maker outperformed the rule-based decision maker. It was successfully demonstrated that by increasing the resource conservation requirements of the system, the robot is capable of surviving longer, but at the expense of completing fewer tasks.

By using constraint-based models of the polar scenario, it has been demonstrated that this system has potential to work for real-world robotics problems in which there are a high number of conflicting goals that must be satisfied in order to make a rational decision. The greatest challenge before the system can be used by the CReSIS robotic platforms is the need for CPU speeds to increase such that

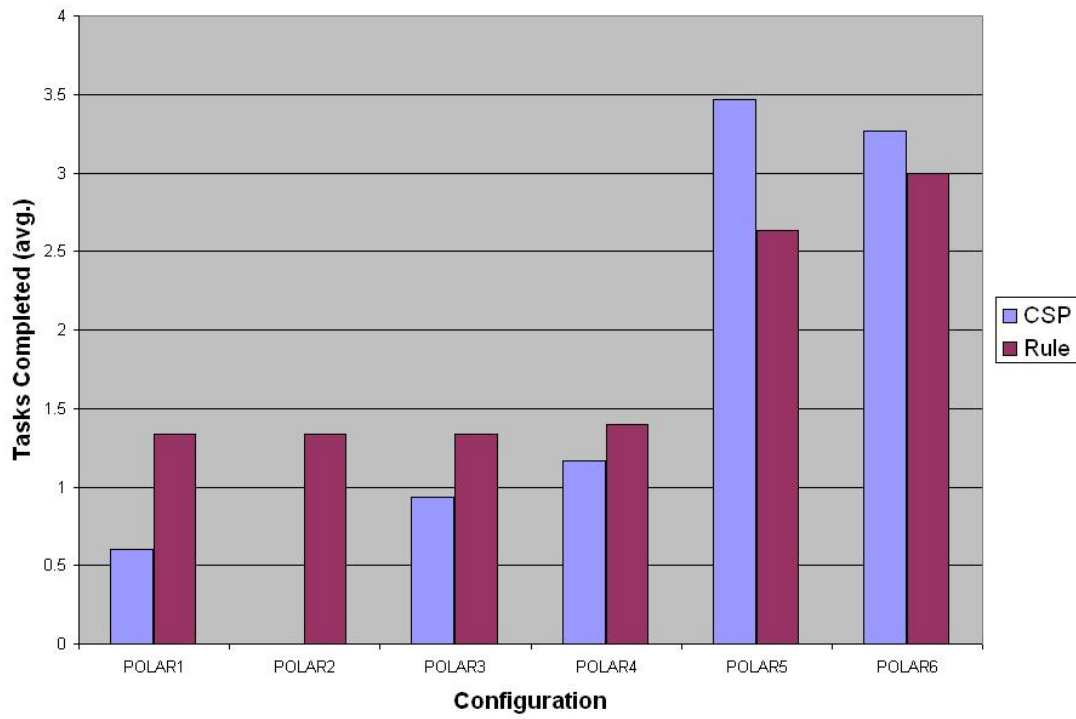


Figure 6.5: Polar scenario: tasks completed versus configuration.

real-time decision making may be obtained for the proposed approach.

Chapter 7

URBAN SEARCH AND RESCUE ROBOT

Urban search and rescue (USAR) has become a growing area of research for mobile robotics. At the University of Southern Florida, the Center for Robot Assisted Search and Rescue has been formed to conduct research in USAR and produce a response team which has aided rescues at the World Trade Center and Hurricane Charley [19]. The National Institute of Standards and Technology has helped design testbeds for USAR robots and now holds several competitions in which researchers may demonstrate and compete using their robots [38].

As robots for USAR become more autonomous, the decisions made by the robot regarding whether or not to continue searching, reporting its results to rescuers, or seeking assistance must be improved. Numerous constraints can be defined regarding how the robot must act within the search and rescue domain. This chapter demonstrates the constraint-based system's potential use for future USAR Robots.

7.1 Background

In this section, an overview of USAR is provided, followed by a discussion on varying levels of autonomy with respect to previous robot search and rescue operations and competitions. The autonomous mapping of the environment is discussed. Techniques and classifications for localization and classification of victims are described.

7.1.1 Urban Search and Rescue

This scenario focuses on urban search and rescue in response to a building collapse that may be the result of a natural disaster or caused by others. Other common USAR tasks are trench collapses (such as occurring on a construction work site) and hazardous material spills.

Building collapses take one of several common forms [15]. “Pancake collapses” occur when the building walls structurally collapse such that each floor collapses upon the lower floor forming a pancake like structure. For these collapses, victims may exist in voids between the layers and have a high-level of obstruction. “V-shape collapses” occur when the middle of a floor buckles causing a collapse in which the middle of one floor rests upon a lower floor. Two voids exist for each side of the collapse. Lastly, a “lean-to” occurs when one wall collapses resulting in a the upper floor to give way and form a triangular void.

The resulting structure poses a topology that is not known a priori. Rescuers must enter through one of the voids and explore the structure. An initial goal is to explore the site and determine potential routes to victims and locate walking wounded victims that are not trapped beneath debris. Next, victims that can easily be extracted are helped out of the rubble. Finally, extrication of the victims

and location of victims within voids are performed. Victims may be identified by being visible, producing thermal signatures, or by making noises such as cries for help.

A mobile robot used for search and rescue would be able to access areas that may be inaccessible to rescuers due to safety concerns, void sizes, potential hazards, etc. By locating victims in these locations, the rescuers can focus their search and extrication in areas where there is a higher probability of finding survivors.

7.1.2 Objectives

A mobile robot is deployed to provide assistance during USAR operations. The robot must coordinate its efforts with human rescuers on site. The robot is placed within a collapsed building whose internal topography is not known due to many structure changes. Unless tethered, since wireless communication may be frequently or continuously interrupted, the robot must operate at a sufficient level of autonomy to rationally make decisions regarding the rescue operation.

While inside the structure, the robot must locate as many victims as possible. Victims may be trapped within voids in which the robot is physically blocked from one or more areas. Coordination with rescuers can lead to later access to previously blocked areas.

The robot's second purpose is to locate hazards within the environment. Hazards may be dangerous to victims, rescuers, equipment, the robot, or all of them. If hazards are identified, the robot must notify the rescue team so that they may either cordon off the area or eliminate the hazard.

As the robot traverses the environment, it must generate a map. The map

provides rescuers with a topology of the environment and a better idea in which more or fewer victims are located. The map is also used to determine the location of the hazards and blocked passages within the environment.

The robot must operate through the environment without colliding or making contact with any of the building structure (obstacles) or the human victims.

7.1.3 Levels of Autonomy

The level of autonomous control for USAR mobile robots varies along two extremes. In tele-operated mode, the robot is controlled via remote control, without any autonomous operation. In autonomous mode, the mobile robot handles most decisions regarding its actions. Researchers at the Idaho National Engineering and Environmental Laboratory (INEEL) define several autonomy modes for USAR robots. Depending on the available level of communication, difficulty of operation, etc., a control system can be implemented that operates on a sliding scale between these levels of autonomy given, the current need [12].

Tele-operated: Robots operating in tele-operated mode are remote controlled via a wireless link or an attached tether. Tethered cases limit the mobility of the robot. Both mechanisms require an active communication link. Should the link become severed, the robots would be unable to continue operation and may be lost in the wreckage as seen with the Solem robot [54], which became disrupted after its wireless signal was lost after entering a void at the World Trade Center site.

Safe Mode: This mode is similar to tele-operated mode, except the robot can override the remote control if it is to be placed in an unsafe situation.

Shared Control: This mode allows the robot some autonomy with respect to following paths and accomplishing local tasks without user intervention. The user is still responsible for the overall guidance of the robot and may interrupt, if deemed necessary.

Fully Autonomous: In this mode, the user has very little control over the robot. The robot is capable of choosing global goals, planning routes, and performing tasks to achieve its goals. A human operator may be available to determine long-term goals and provide guidance.

7.1.4 Locating and Classifying Victims

While generating a map of the disaster site, the robot must be capable of detecting survivors within each area. In [13], research was conducted to determine whether or not a suite of sensors could be selected to locate victims. Some of the sensors explored included: a microphone, a pyro sensor, motion detectors, IR sensors, etc. The fusion of multiple sensors is necessary in order to compensate for the various limitations of each sensor, with respect to environmental conditions.

In the future, it is hoped that the robot will not only be capable of locating victims, but also accurately labeling each victim with respect to their injury level so that rescuers will be better informed regarding rescue needs. For this research scenario, it is assumed that such an ability exists for the simulated search and rescue robot.

A victim's status is based on the standard triage definition, defined in four categories [70]: immediate, delayed, minor, and dying/dead. Immediate victims require the most immediate attention as they are the most critically injured. Delayed victims' treatment may be delayed until immediate victims are treated.

Minor victims do not require treatment onsite and are in some cases classified as walking wounded.

7.2 The Robot

In this section, the simulated robot's drive system, power system, percepts, onboard memory, and internal map are presented. The details described are incorporated into the software simulation of the robot and will also be used to help construct the constraint model.

7.2.1 Base Station

It is assumed that at the rescue team's coordination location, which is referred to as the "base" location, a docking station is available for the robot to recharge its battery and download data to the rescuers.

7.2.2 Percepts

The robot is equipped with a suite of sensors that allows it to carry out its search and rescue operation. The percepts are generated from the sensor suite:

- **Hazard detection:** At each location, the robot is capable of determining the presence of a hazard. The hazard may or may not be hazardous to the robot.
- **Victim status:** At each location, the robot can locate all living victims present and determine their status as minor, delayed or critical.

- **Obstruction level:** The obstruction level of the location may be assessed and is assigned a value from Low to High. The robot is also capable of perceiving whether or not a location is blocked or inaccessible.
- **Simultaneous localization and mapping sensors (SLAM):** The robot is equipped with the sensors necessary to localize itself within the environment and construct additional map details as it traverses through the environment.
- **Battery charge level:** The percent charged level of the battery is perceived.
- **Robot damage:** Damage sustained to the robot is sensed.

7.2.3 Drive System

The robot is equipped with a drive system that is capable of moving about most debris fields. The decision maker assigns linguistic speeds from Low to High, which are mapped to actual drive speeds.

The robot may encounter locations that are blocked and it must choose an alternate route and report the blockage to rescuers.

Damage may occur to the drive system due to interaction with hazardous terrain or colliding with obstacles. Damage may be repaired at the base location, but a time penalty is incurred. If enough damage is sustained, the robot becomes inoperable and the scenario ends.

7.2.4 Power System

The robot is provided with a battery to power its onboard system and support locomotion. If the robot depletes its battery by a specified threshold, the robot must recharge. Recharging the battery causes a time penalty. If the battery becomes completely discharged, the robot becomes inoperable and the scenario ends.

7.2.5 Internal memory

The robot has a limited memory space to store data for each victim that is discovered. It is assumed that no wireless communication is available. Therefore, the robot must return to base in order to download the information to the rescuers whenever critical data is received or the memory is full. A fixed amount of time is associated with the download of each piece of data.

7.2.6 Internal Map

The robot constructs an internal map of its environment as it travels through the environment. At each location it first visits, all neighbors to the location are viewed. If a neighbor location has never been previously visited, it is added to the map. Once a location is added to the map, it may be visited by the robot so long as it is classified as accessible. The map also notes all inaccessible locations so that this information can be passed on to the rescue team.

7.3 Constraint Model

The USAR robot can choose from one of five tasks: Search, Charge, Upload, Repair, and Wait. The wait task is the default task when no solution can be derived by the CSP. All tasks, except for the wait task, are derived from an abstract task definition, movement task, which is responsible for defining all constraints related to the robot's movement within the environment. Figure 7.1 presents the class hierarchy for the USAR tasks from the Task interface to each derived task.

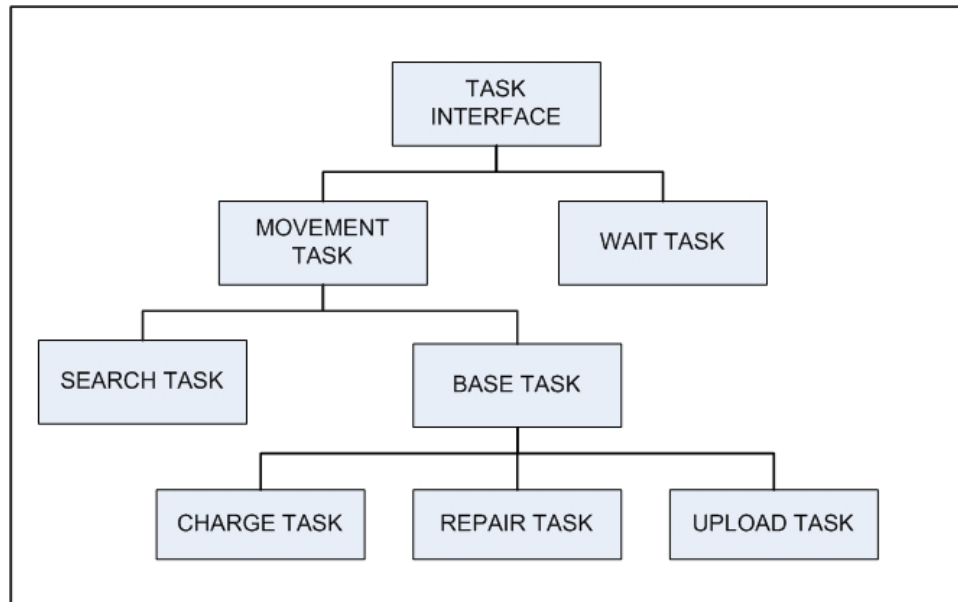


Figure 7.1: USAR Scenario: class hierarchy.

In this section, the variables used for the constraint model are defined, followed by, the constraint-model for movement, and the models for each of the implemented classes.

7.3.1 Variables

Performance Variables: For the USAR constraint models, the standard performance variables of risk avoidance, timeliness, and resource conservation are utilized. In addition, the *Quality* variable is also included. Quality is used to define the quality or desirability of performing the specified action, given the current state.

Control Variables: These variables store control values used to determine how the robot executes the selected task:

- *PathVariable* indicates the path the robot follows when executing the selected task.
- *SpeedVariable* indicates the speed at which the robot attempts to travel while following the selected path.

Percept Variables: These variables store data perceived by the robot regarding the state of the environment, the map of the environment, and the robot itself:

- *PathLengthVariable* stores the length of the selected path.
- *ObstructionVariable* stores the obstruction level of the selected path.
- *PathHazardVariable* stores a Boolean value where true indicates a hazardous location is present along the path.
- *BasePathVariable* stores a Boolean value indicating whether or not a path terminates at the rescue team's base location.
- *DamageVariable* stores the current level of damage sustained by the robot.

- *BatteryVariable* stores the current battery charge level.
- *PayloadVariable* stores the current memory usage level.
- *VictimVariable* stores the state of the most critical victim discovered.
- *HazardVariable* stores a Boolean value indicating whether or not a hazardous location's data is stored within the robot's memory.
- *BlockedVariable* stores a Boolean value indicating whether or not a blocked location's data is stored within the robot's memory.

7.3.2 Abstract Movement Task

The abstract movement model defines a set of standard constraints for movement within the USAR environment.

A movement task is responsible for the movement of the robot between locations. The speed and path are selected by the CSP. At each transition, the battery is discharged one unit, the system clock is updated, given the robot's speed and an assumed distance of three meters between locations. A collision can occur at any location given a probability that correlates to both the robot's speed and the obstruction level of the location. Hazards at each location can cause damage to the robot, but are not assessed negatively by the experimental metrics.

A movement task is preempted if a collision occurs, the robot's battery becomes fully discharged, or the robot is disabled due to damage. Unless the damage threshold is exceeded, encountering a location with a hazard does not preempt the movement task's execution.

The movement task generates a set of paths which the robot can travel. All paths to the base from the robot's current position are included. In addition, for

every location that has been sensed, but not visited, all paths to that location are included.

Path Constraints: Constraints are defined in order to integrate state data regarding the choice in paths to the movement model. The first constraint states that if a path is not traversable, the path variable cannot be assigned a value equal to its path ID. The second and third constraints set the path hazard variable to true or false depending on whether or not the path has a hazardous location. The next constraint states that if a path is selected, then the path obstruction level is equal to the path's obstruction level. Lastly, if the path variable is assigned a path's ID, then the path length variable is assigned the length of the path.

1. if path i is not traversable, ($\neq PathVariable\ i$)
2. if path i has a hazard,
 $(\Rightarrow (== PathVariable\ i) (== PathHazardVariable\ true))$
3. if path i does not have a hazard,
 $(\Rightarrow (== PathVariable\ i) (== PathHazardVariable\ false))$
4. $(\Rightarrow (== PathVariable\ i)$
 $(== ObstructionVariable\ Path(i).obstructionLevel))$
5. $(\Rightarrow (== PathVariable\ i)$
 $(== PathLengthVariable\ Path(i).pathLength))$

Timeliness Constraints: If the demanded task timeliness is Medium-Low, the speed of the robot must be greater than Low and the length of the path travelled must be less than High. If task timeliness is Medium, the robot's speed must be

greater than Medium-Low, or the path length must be less than Medium-High. If the desired timeliness is Medium-High, the robot's speed must be greater than Medium or the path length must be less than Medium. If the task's timeliness is High, the robot's speed must be High or the path length must be Low:

1. (\Rightarrow ($==$ *TaskTimeliness Medium – Low*)
 $(\parallel (> \textit{SpeedVariable Low}) (< \textit{PathLengthVariable High}))$)
2. (\Rightarrow ($==$ *TaskTimeliness Medium*)
 $(\parallel (> \textit{SpeedVariable Medium – Low})$
 $(< \textit{PathLengthVariable Medium – High}))$)
3. (\Rightarrow ($==$ *TaskTimeliness Medium – High*)
 $(\parallel (> \textit{SpeedVariable Medium}) (< \textit{PathLengthVariable Medium}))$)
4. (\Rightarrow ($==$ *TaskTimeliness High*)
 $(\parallel (== \textit{SpeedVariable High}) (== \textit{PathLengthVariable Low}))$)

Risk Constraints: The risk avoidance constraints are divided into two groups: given the performance requirements, what are the acceptable assignments and given the states how the performance is constrained.

In the first group, the first constraint states that if the expected task avoidance level is Medium-Low, then the speed of the robot must be less than High or the selected path must have an obstruction level less than Medium-High. If the risk avoidance level is Medium, either the speed of the robot or the obstruction level of the path must be less than Medium-High. If the risk avoidance level is Medium-High, either the speed of the robot or the obstruction level of the path must be less than Medium. If the risk avoidance level is High, either the speed of the robot or the obstruction level of the path must be Low:

1. (\Rightarrow ($==$ *TaskRiskAvoidance Medium – Low*)
(\parallel ($<$ *SpeedVariable High*) ($<$ *ObstructionVariable High*)))
2. (\Rightarrow ($==$ *TaskRiskAvoidance Medium*)
(\parallel ($<$ *SpeedVariable Medium – High*)
($<$ *ObstructionVariable Medium – High*)))
3. (\Rightarrow ($==$ *TaskRiskAvoidance Medium – High*)
(\parallel ($<$ *SpeedVariable Medium*) ($<$ *ObstructionVariable Medium*)))
4. (\Rightarrow ($==$ *TaskRiskAvoidance High*)
(\parallel ($==$ *SpeedVariable Low*) ($==$ *ObstructionVariable Low*)))

In the second group, if a hazardous location lies along the path and the robot's damage level is Low, the risk avoidance of the decision is less than High. If a hazardous location lies along the path and the robot's damage level is Medium-Low, the risk avoidance of the decision is less than Medium-High. If a hazardous location lies along the path and the robot's damage level is Medium, the risk avoidance of the decision is less than Medium. If a hazardous location lies along the path and the robot's damage level is greater than Medium, the risk avoidance of the decision is Low:

1. (\Rightarrow ($\&\&$ ($==$ *PathHazardVariable true*) ($==$ *DamageVariable Low*))
($<$ *TaskRiskAvoidance High*)))
2. (\Rightarrow ($\&\&$ ($==$ *PathHazardVariable true*)
($==$ *DamageVariable Medium – Low*)))
($<$ *TaskRiskAvoidance Medium – High*)))

3. $(\Rightarrow (\&\& (== \textit{PathHazardVariable true}) (== \textit{DamageVariable Medium})))$
 $(< \textit{TaskRiskAvoidance Medium}))$
4. $(\Rightarrow (\&\& (== \textit{PathHazardVariable true}) (> \textit{DamageVariable Medium})))$
 $(== \textit{TaskRiskAvoidance Low}))$

Resource Constraints: The resource conservation of the robot is defined with respect to the length of the path followed. If the conservation level is Medium-Low, then the path must be less than High. If it is Medium, the path length must be less than Medium-High. If the conservation level is Medium-High, then the path length must be less than Medium. If High conservation is required, the path length must be Low:

1. $(\Rightarrow (== \textit{TaskResourceConservations Medium - Low})$
 $(< \textit{PathLengthVariable High}))$
2. $(\Rightarrow (== \textit{TaskResourceConservations Med})$
 $(< \textit{PathLengthVariable Medium - High}))$
3. $(\Rightarrow (== \textit{TaskResourceConservations Medium - High})$
 $(< \textit{PathLengthVariable Medium}))$
4. $(\Rightarrow (\&\& (== \textit{TaskResourceConservations High})$
 $(== \textit{PathLengthVariable Low}))$

7.3.3 Search Task

The search task guides the robot to a location that has not previously been visited by the robot. At each location, the robot is capable of perceiving the presence of any hazards, victims, accessible neighboring locations, and blocked neighboring

locations. The location data becomes part of the robot's memory. The search task extends the movement task and therefore inherits all of the movement constraints. Several additional constraints are added to guide path selection and enforce the desired quality of the task.

The first constraint states that the path selected for the task cannot be a path to the base. The second constraint states that the maximum quality of the task is Medium-High. The reason for including this constraint is so that it will always have a higher priority, but may be pre-empted by system critical tasks such as allowing the charge task to execute if the battery is Low.

The remaining constraints indicate conditions in which the quality of exploring is Low. These conditions include: knowledge of an unreported victim needing immediate care, High damage, Low battery, or that the robot's memory is full:

1. $(== \text{BasePathVariable } false)$
2. $(<= \text{TaskQuality } \text{Medium} - \text{High})$
3. $(=> (== \text{VictimVariable } \text{Victim.IMMedIATE})$
 $(== \text{TaskQuality } \text{Low}))$
4. $(=> (== \text{DamageVariable } \text{High}) (== \text{TaskQuality } \text{Low}))$
5. $(=> (== \text{BatteryVariable } \text{Low}) (== \text{TaskQuality } \text{Low}))$
6. $(=> (== \text{PayloadVariable } \text{High}) (== \text{TaskQuality } \text{Low}))$

7.3.4 Charge Task

In order to operate the robot during the search and rescue scenario, the battery is continually discharged with each transition made between locations. The charge

task allows the robot to be restored to a fully charged state. The robot moves to the rescue team's base station. Once at the base, the robot charges at a rate of 36.0 seconds per charge unit (one hour to charge from a fully discharged battery). There are 100 total charge units allotted to the battery.

The charge task extends the movement task and inherits all constraints related to the robot's movement. For the charge task to be selected, the path selected must terminate at the rescue team's base location. If the robot's battery is Low, the quality of charging is High. If the battery's charge is greater than Medium, the quality of charging is Low. If the charge is Medium and the task's timeliness is greater than Medium, the quality of charging the robot is Medium. If the battery's charge is Medium-Low and the timeliness is greater than Medium-Low, the task quality is Medium. The last two constraints allow the task to be selected before it is absolutely necessary so long as it is sufficiently timely:

1. $(== \text{BasePathVariable true})$
2. $(=> (== \text{BatteryVariable Low}) (== \text{TaskQuality High}))$
3. $(=> (> \text{BatteryVariable Medium}) (== \text{TaskQuality Low}))$
4. $(=> (\&\& (== \text{BatteryVariable Medium}) (> \text{TaskTimeliness Medium})) (== \text{TaskQuality Medium}))$
5. $(=> (\&\& (== \text{BatteryVariable Medium} - \text{Low}) (> \text{TaskTimeliness Medium} - \text{Low})) (== \text{TaskQuality Medium}))$

7.3.5 Upload Task

The upload task is responsible for guiding the robot to the base station and uploading its data. The robot must upload data to the rescue team for a variety

of reasons. One basic motivation for uploading data is the robot's memory becoming full (maximum 20 locations). If the robot reports victim data, the victim is classified by the robot as rescued if they are alive at the time of reporting. Hazardous and blocked locations can be cleared after a delay, once reported. These are cleared in the order they are received with 3,600 seconds per blocked location and 7,200 seconds per hazard. The data upload takes 30 seconds per location (maximum 600 seconds).

The following hidden variables are used to indicate the quality of uploading data given the current respective state for several percepts:

- *HazardQuality* is the quality of uploading, given whether or not a hazardous location has been discovered.
- *BlockedQuality* is the quality of uploading, given whether or not a blocked location has been discovered.
- *PayloadQuality* is the quality of uploading, given the memory usage of the robot.
- *VictimQuality* is the quality of uploading, given the most critical Victim data stored in memory.

The upload task extends the movement task and inherits all of its movement constraints. It is also constrained such that it can only follow paths that terminate at the base location. The constraints for the upload task use the hidden variables to determine the quality of uploading for each respective percept. Once each is collected, the task's quality is equal to the maximum of the hidden variable's values. The robot's memory is finite. Therefore, the quality of uploading the data based on capacity increases as the memory usage increases.

With respect the the injury level of the unreported victims within the robot's memory, a number of constraints are defined. If the victim's injury level is minor, the quality of uploading victim data is Low. If the injury level is delayed, the quality of uploading the victim data is Medium-Low. If the injury level is Immediate, the quality of uploading the victim data is High as treatment must be rendered in the near future.

If the robot discovers a blocked path or a hazard has been discovered, the quality of uploading is High:

1. (\Rightarrow ($==$ *BasePathVariable true*))
2. ($==$ *TaskQuality*
Max(HazardQuality, BlockedQuality, VictimQuality, PayloadQuality))
3. (\Rightarrow ($==$ *PayloadQuality PayloadVariable*))
4. (\Rightarrow ($==$ *VictimVariable Victim.MINOR*)
($==$ *VictimQuality Low*))
5. (\Rightarrow ($==$ *VictimVariable Victim.DELAYED*)
($==$ *VictimQuality Medium – Low*))
6. (\Rightarrow ($==$ *VictimVariable Victim.IMMedIATE*)
($==$ *VictimQuality High*))
7. (\Rightarrow ($==$ *BlockedVariable true*) ($==$ *BlockedQuality High*))
8. (\Rightarrow ($==$ *BlockedVariable false*) ($==$ *BlockedQuality Low*))
9. (\Rightarrow ($==$ *HazardVariable true*) ($==$ *HazardQuality High*))
10. (\Rightarrow ($==$ *HazardVariable false*) ($==$ *HazardQuality Low*))

7.3.6 Repair Task

The repair task guides the robot to the base and allows the robot to have its damage repaired. The robot may sustain up to 100 units of damage (10 units of damage are given per collision or hazard encounter). Once at the base, the robot is repaired at an arbitrarily selected rate of one damage unit per 40 seconds.

The repair task inherits movement constraints from the movement task. The robot must select a base path in order to be repaired. If the robot's damage level is greater than Medium, the quality of seeking repair is High. Otherwise, the quality of seeking repair is equal to the level of damage sustained to the robot:

1. $(=> (== \text{BasePathVariable } true))$
2. $(=> (> \text{DamageVariable } Medium) (== \text{TaskQuality } High))$
3. $(=> (<= \text{DamageVariable } Medium) (== \text{TaskQuality } \text{DamageVariable}))$

7.3.7 Wait Task

The wait task is selected only if no other tasks can be selected. The robot waits for 60 seconds before attempting to select a new task again.

7.3.8 Utility Function

The constraint solver yields multiple solutions, and from these solutions, the one with the highest utility function is used. Ties are handled arbitrarily. The utility function for all tasks in the USAR scenario is: $Utility = TaskQuality + TaskTimeliness + TaskRiskAvoidance + TaskResourceConservation$.

7.3.9 Performance Requirements

For the previous scenarios, experimentation derived the best configuration for the CSP's performance parameters that the task selector must meet or exceed. For the search and rescue scenario, during initial experiments, it was discovered that the robot could become stuck such that a safe route could not be derived to explore or return to the base. If the performance requirements were lowered to avoid these states, the overall performance of the experiment was diminished. In order to compensate for this issue, the following procedure is carried out by the constraint solver:

1. All performance requirements are set to High performance.
2. The solver attempts to solve the problem. If solutions are derived, the best is selected and executed.
3. Otherwise, each of the performance variables minimum expectation is decremented by one.
4. The solver re-attempts to solve the problem. If solutions are derived, the best is selected and executed.
5. The process repeats until either a solution is finally derived and executed, or all performance metrics are decremented to Low.

During the evaluation of the scenario, the system's performances with and without this feature are presented.

7.4 Evaluation

This section describes the experimental setup for the simulated demonstration of the decision maker for task selection and configuration in a USAR scenario.

7.4.1 Metrics

- ***Rescues:*** the number of non-deceased victims reported to the rescue team.
- ***Hazards reported:*** the number of hazards within the environment that are reported to the rescue team.
- ***Locations reported:*** the number of locations mapped and reported to the rescue team.
- ***Collisions:*** the number of collisions between the robot and obstacles. This metric is considered negative and thus a lower number is desired.
- ***Score:*** the final metric derives its value from the other metrics. It is derived from robot search and rescue competitions. For each victim reported, ten points are awarded. For every hazard reported, ten points awarded. For every location reported, one point is awarded. Finally, there is a ten point deduction for every collision. This metric shows how well the robot meets the needs of the rescue team.

7.4.2 Experimental Parameters

Maps

Two maps are used to during the evaluation of the USAR scenario. The first (Figure 7.2) is a two loop hospital-like topology borrowed from the delivery

scenario. The second (Figure 7.3) is a three hallway hotel-like topology. The base location is connected to the central location for each map.

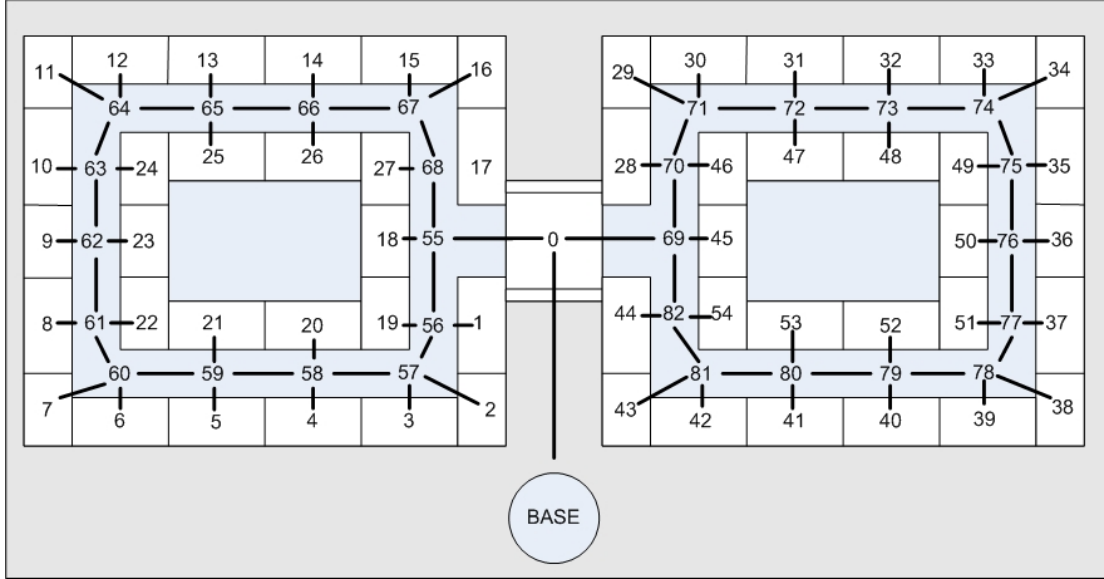


Figure 7.2: USAR scenario: the hospital topology.

The hospital topology contains one centralized location that connects two looped hallways. Locations adjacent and connected to the hallway representing patient rooms. By having multiple looped hallways in the event of a blocked path, the robot can have an alternate route available.

The hotel topology contains one centralized location that joins three radiating hallways. Each hallway is connected to a number of adjacent locations that represent hotel rooms.

Hazards, Blocked Locations, and obstruction

The selected map is populated with random blocked and hazard locations. No location is both blocked and possesses a hazard. The number of hazards and blocks is determined prior to the experiments. Every location is assigned an obstruction

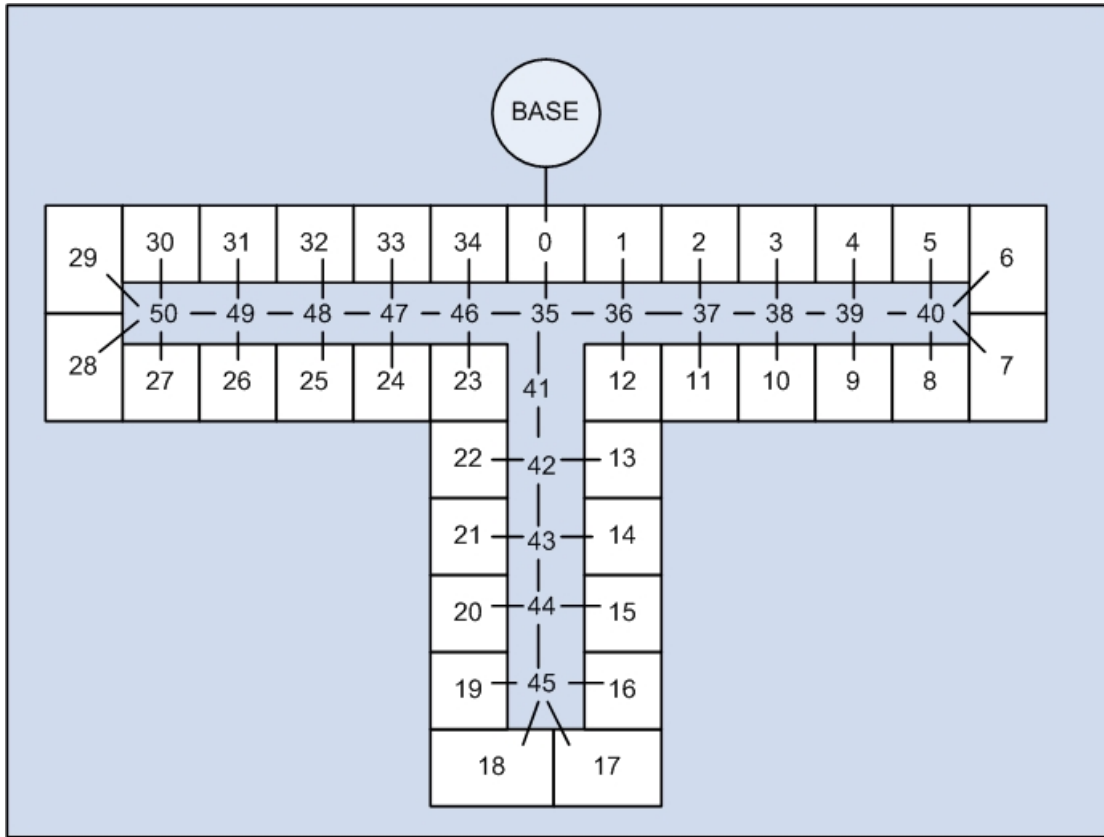


Figure 7.3: USAR scenario: the hotel topology.

level, in addition to these parameters, indicating the level of difficulty and risk associated with robot traversal.

Victim Distribution

The three levels of triage represent the initial conditions for victims distributed within the environment. Prior to run time, the levels of immediate, delayed, and minor victims are specified. These victims are randomly distributed throughout the environment. No victim is placed in locations in which hazards exist, as they would be considered inhospitable for any survivor.

Experimental Configurations

Table 7.1 presents eight configurations that are used to evaluate the USAR robot. Two maps are specified with four configurations each. The victim states are varied from a greater number of minor injuries to a greater number of delayed injuries. When there are more delayed injuries, the robot has less time to act and rescue victimsx. The number of hazards remains fixed at five. The number of blocked locations is varied between 10 and 20.

ID	Map	Victim State			Hazards	Blocked
		Minor	Delayed	Immediate		
USAR1	Hospital	125	50	25	5	10
USAR2	Hospital	50	125	25	5	10
USAR3	Hospital	125	50	25	5	20
USAR4	Hospital	50	125	25	5	20
USAR5	Hotel	125	50	25	5	10
USAR6	Hotel	50	125	25	5	10
USAR7	Hotel	125	50	25	5	20
USAR8	Hotel	50	125	25	5	20

Table 7.1: USAR Scenario: Configurations for Evaluation.

In addition to these configurations, two additional experimental runs are performed using USAR2 and USAR4. These experiments will use the constraint-based decision maker with fixed instead of dynamic performance requirements. The performance requirements for timeliness, resource conservation, risk avoidance, and quality will be fixed at Medium.

7.4.3 Rule-based System

A rule-based decision maker is generated for comparison against the constraint-based system:

1. Task Selection

If charge $< 20\%$, charge task selected.

Else if damage > 75 points, repair task selected.

Else if memory usage $> 75\%$, upload task selected.

Else if maxVictimStatus == Immediate, upload task selected.

Else if hazard found, upload task selected.

Else if blocked location found, upload task selected.

Else, search task selected.

2. Get Path

If search task, select shortest non-base path.

Else, select the shortest base path.

3. Speed

If the obstruction level is Low, the speed is Medium-High.

If the obstruction level is Medium-Low, the speed is Med.

If the obstruction level is Med, the speed is Medium-Low.

If the obstruction level is Medium-High, the speed is Low.

If the obstruction level is High, the speed is Low.

7.4.4 Computing Platform

This experiments were run on a PC running Linux with four dual core AMD Opteron processors and 20GB of RAM. The platform ran Java version 1.6.0.

7.5 Results and Analysis

This section presents the experimental results and analysis of the decision maker for the USAR scenario. Note: When the fixed performance criteria are used for USAR2 and USAR4, they are labeled as USAR2 FIXED and USAR4 FIXED.

7.5.1 Scores

Figure 7.4 presents the mean score for each configuration. While the mean presented for the constraint-based decision maker is generally higher than the rule-based decision maker, statistical analysis states that USAR2, USAR5, USAR6, and USAR8 have a higher constraint-based than rule-based mean at a 95% confidence level. The rule-based system outperformed the constraint-based system by a significant margin when fixed performance metrics were used. Comparison of the remaining data sets was inconclusive.

USAR2, USAR6, and USAR8 all had a greater number of delayed victims than minor victims. As a result, under these scenarios, the total time taken to explore the environment, locate victims and hazards, and report the data was much smaller. Therefore, it may be concluded that the constraint-based system is better when more time critical tasks are performed.

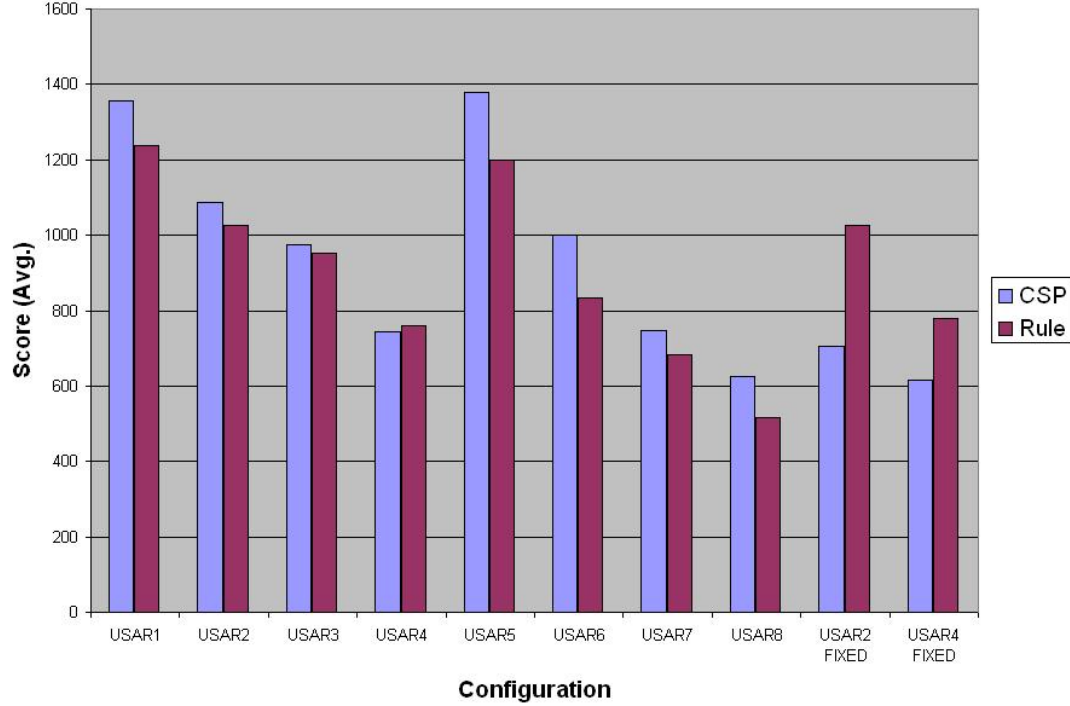


Figure 7.4: USAR scenario: performance score versus configuration.

7.5.2 Rescues

Figure 7.5 presents the mean number of victims rescued for each configuration. For this particular metric, the comparison of performance was statistically inconclusive for USAR1, USAR2, USAR3, USAR4, and USAR5 at the 95% confidence level.

The constraint-based decision maker outperformed the rule-based system under USAR6 and USAR8. Both of these experiments used the hotel map and a greater number of Delayed victims than Minor victims. It can be concluded that under these configurations the constraint-based system was able to rescue more victims in a shorter period of time.

When fixed performance requirements were used, the rule-based system out performed the constraint-based system. This illustrates the importance

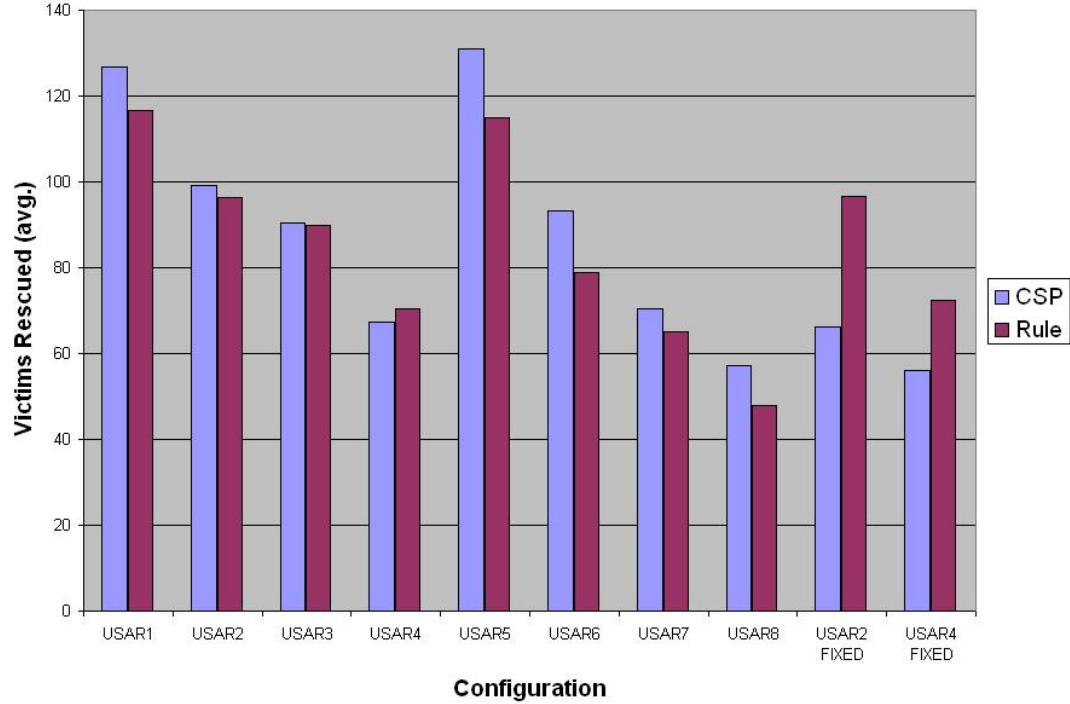


Figure 7.5: USAR scenario: rescued victims versus configuration.

of flexibility in the performance requirements when operating in uncertain environments.

7.5.3 Reported Hazards

Figure 7.6 presents the mean number of reported hazards for each configuration. Despite the means presented, the variability was so great for these results that through statistical analysis, at a 95% confidence level, the constraint-based system performed better for only USAR 6 and USAR8. For the fixed performance cases, USAR2 FIXED and USAR4 FIXED, the performance was much lower because the mobile robot frequently became stuck and could not choose a task to perform without violating the system's constraints. For all other configurations, the results

were statistically inconclusive and therefore assumed equally matched.

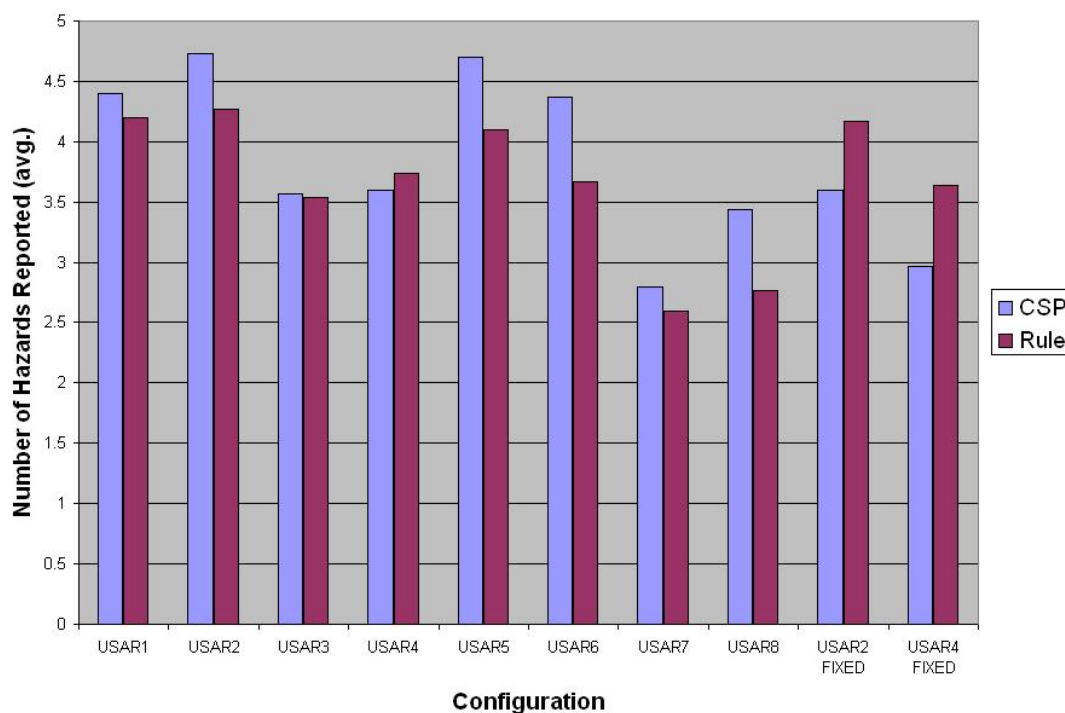


Figure 7.6: USAR scenario: hazards reported versus configuration.

7.5.4 Reported Map Locations

Figure 7.7 presents the mean number of reported locations for each configuration. Under scenarios USAR1, USAR2, USAR6, and USAR8 the mean number of locations was significantly greater for the constraint-based system than the rule-based system. As expected, the USAR2 FIXED and USAR4 FIXED, performed worse under the rule-base. For all other cases, the statistical analysis was inconclusive.

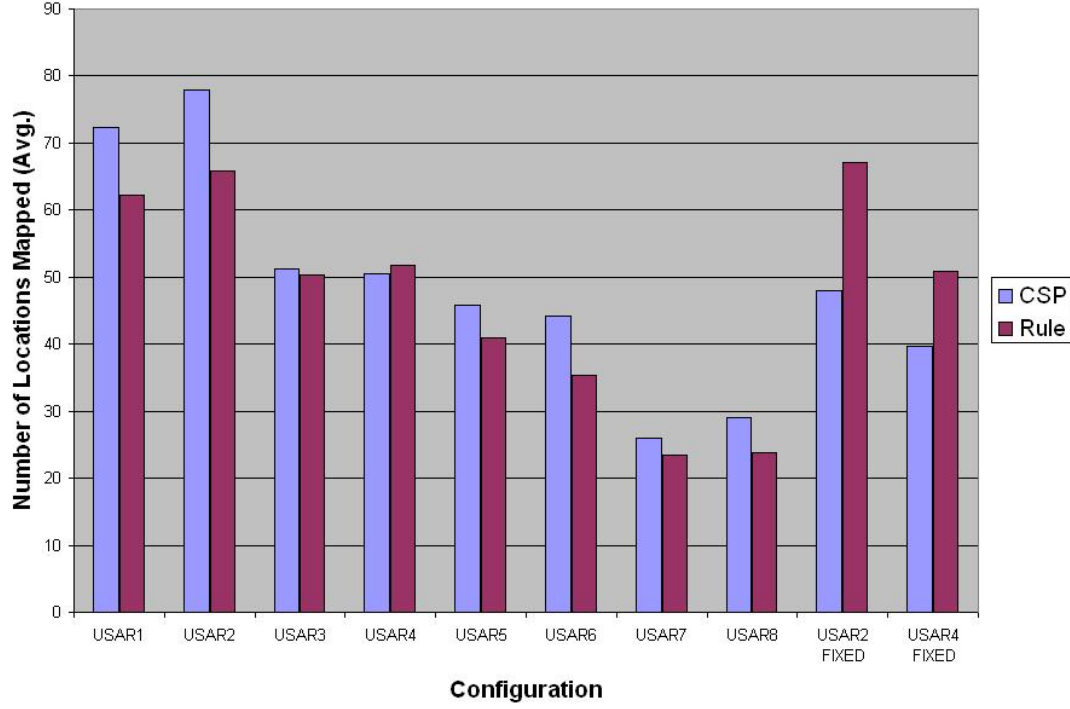


Figure 7.7: USAR scenario: locations mapped versus configuration.

7.5.5 Collisions

Figure 7.8 presents the mean number of collisions for each configuration. The constraint-based system outperformed the rule-based system with respect to avoiding collisions. For configurations USAR2, USAR3, USAR4, USAR6, and USAR4 FIXED, the difference was statistically significant. For all other cases, the results were inconclusive. None of the configurations yielded a case in which the mean number of collisions for the rule-base was less than that of the constraint-based system.

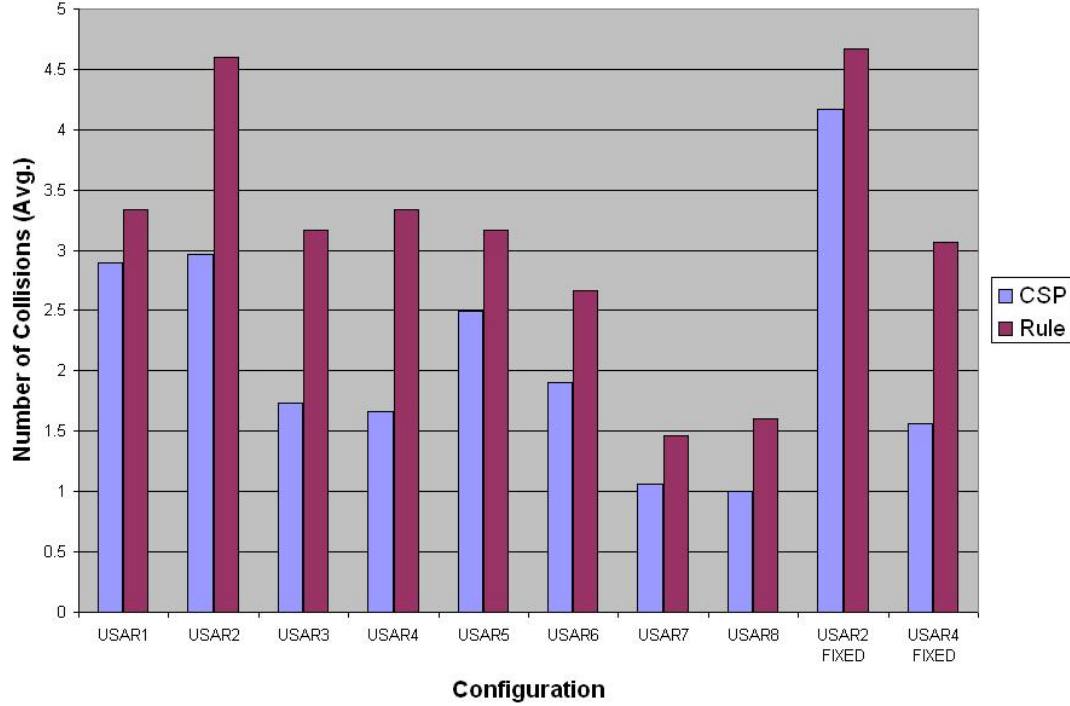


Figure 7.8: USAR scenario: obstacle collisions versus configuration.

7.5.6 Discussion

From the experimental results, a number of conclusions can be drawn. Unlike the polar scenario, for the search and rescue scenario, the performance difference between the two systems was frequently small enough so that one could not be declared significantly superior to the other. The high variability in the map between runs could be a reason as the randomization of the map may have a significant influence on score. This is likely why the figures would show the constraint-based having a better mean, but be statistically inconclusive. As the number of blocked locations increased, the system's performance often, but not always degraded. In these cases, it appears that the robot's actions are being guided more by the environment and the limits put forth by the simulator than

by the decision makers itself.

Chapter 8

CONCLUSION

A constraint-based decision maker was constructed and demonstrated for the selection and configuration of mobile robot tasks.

8.1 Contributions

A novel constraint-based decision maker was successfully demonstrated that meets the needs defined in the introduction and confirms the dissertation’s hypothesis. Constraint-based methods are capable of guiding the mobile robot toward rational decisions while meeting multiple objectives and satisfying performance goals.

For demonstration and experimentation of the proposed research, three unique robots were implemented, simulated, and evaluated that utilized the constraint-based decision maker. From these experiments several conclusions may be derived regarding the benefits and performance of using CSPs for the modeling of task selection for a decision maker.

Each scenario was implemented and simulated independently. Only the solver and decision maker framework were re-used for each scenario. The results for each

scenario demonstrated successful operation of the mobile robots in each respective scenario while using the constraint-based decision maker. By demonstrating the framework on a diverse set of robot applications, the generalizability of the system has been shown. It can be utilized with problems that have static and dynamic environments. The framework can be applied to other robot task selection problems.

The new framework is extensible. The use of object-oriented programming techniques and the ability to easily merge CSPs allowed for models to be constructed from one or more abstract models. During the construction of the constraint models, the robot tasks shared many overlaps where the same variables were required with similar constraints placed upon them. Using top-down design, the general constraints could be applied to all system variables. Then at each lower level, constraints were added and/or refined. For instance, the USAR scenario defined several base tasks: Search, Charge, Upload, and Repair. Each of these tasks were derived from the Movement task. Therefore, all of the constraints for movement were defined in one location. The task specific constraints were built upon the base constraints and merged into a single CSP.

The performance variables provided a very intuitive means for deriving constraints such that adjustments to the model could be made at run time. The constraint models implemented in this framework are flexible. Unlike rule-based models that rigidly guide, the implementation of the constraints about more than one performance expectation allowed the model to be adjusted with respect to individual expectations. The system was tuned prior to execution for the Delivery and Polar Scenarios. The results from these scenarios indicated that setting the requirements once was sufficient. For the USAR scenario, where the robot's environment was not known prior to run time, the performance requirements

were adjusted starting at the highest performance requirement and lowering until a solution was derived. In the delivery scenario, a variety of error modes were generated to demonstrate the CSP's ability to continue operation rationally regardless of unforeseen states. In the polar scenario, the decision between power generation and instrument usage was not immediately apparent. However, despite making no explicit links between these two aspects of the robot's execution, the constraint-based system consistently survived longer than the rule-based system.

While the scalability is a limitation of the system, as described below, the results have provided insight to the scalability issues of CSPs for robot applications. The delivery scenario was not capable of timely operation when it received a large number of simultaneous requests. However, it was capable of providing better decisions than the rule-based system under these loads. It may therefore be concluded that as the CPU speeds increase in the future will not be as significant of an issue the benefits of the CSP will be more apparent.

The journal articles has been written for the Journal of Intelligent and Robotic Systems [75] and the International Journal on Autonomous Robots [74] converging the polar robot and USAR robot applications of the proposed framework.

8.2 Limitations

The following limitations exist with the current implementation of the approach.

- The computation time complexity is much greater for the constraint-based system than the rule-based system. This was demonstrated in each of the scenarios and specifically presented when discussing the results of the delivery scenario. Some techniques have been used to reduce the model size such as enumerating data into linguistic values instead of continuous integer

values. These techniques reduce the complexity of the model, but may oversimplify the problem such that constraints based on subtle changes must be excluded.

- Given the time complexity issues and the complexity of the scenarios presented, only simulation experiments were used for demonstrating the proposed framework. The simulations were based on physical hardware and expert knowledge of the problem domains. However, there were some situations in which assumptions were made and arbitrary values were selected. For instance, the simulation of robot damage and collisions were based on actual events that may occur, but the amount of damage that was sustained and the probabilities of collisions occurring were estimates.
- No formal methods exist for defining or verifying the constraint models. For this research, the models were defined using expert knowledge. Their declaration was fairly ad hoc. Empirical testing is required to determine if the experiments were valid. In addition, the adjustment of the performance criteria for the polar scenario and delivery scenario were done manually. Once a configuration generated a successful performance, the criteria's value was fixed. It was possible to select configurations that prohibited the robot from performing any action.

8.3 Future Work

Given the success of the scenarios with respect to mission critical metrics, constraint satisfaction is a viable technique that can be used for this problem. As with all research, future work is needed.

In order to further demonstrate the constraint-based decision maker for task selection, a sub-system of the robot can be selected and the decision maker can be applied to guide its activities. For example, the decision maker can be incorporated with the PRISM mobile robot to select the active position and guidance sensors for the robot's control system to follow. It can also be used to select the current path following mode, given the path error rate and approximated terrain conditions.

For more extensive demonstration and application of the constraint-based system to real-time systems, artificial intelligence techniques for learning can be applied. A simulation can be used to determine an appropriate response, given the state. With the robot's success and selecting an appropriate act, it is rewarded. A wrong decision by the CSP would be penalized. Techniques such as reinforcement learning, neural networks, and genetic algorithms all have such learning capabilities.

Further exploration of CSP algorithms can be attempted to lower the computational intensity of the decision making process. Repair algorithms for CSPs are one potential technique that could be explored. For repair algorithms, each variable is instantiated with a value, possibly the previous solution. The solution is then "repaired" in order to have every constraint satisfied. For robot applications in which the environment is moderately static, the computation time between solutions could be dramatically reduced.

Formal methods and model verification should be explored before these techniques can become trusted and mainstream. Future work in these areas is necessary.

Lastly, on-chip systems have become common for other AI techniques such as neural networks, fuzzy controllers, and genetic algorithms. When placed on

hardware, the computation time may be dramatically reduced. FPGAs are a method to bridge software and hardware. After a successful demonstration of the decision maker for mobile robots, this would be a natural area of academic exploration.

Bibliography

- [1] E. L. Akers, R. S. Stansbury, and A. Agah, “Long-Term Survival of Polar Mobile Robots,” in *Proceedings of the 4th International Conference on Computing, Communications and Control Technologies (CCCT)*, vol. II, Orlando, FL, July 2006, pp. 329–333. C
- [2] C. Allen, Personal Communication, May 2007. J
- [3] American Heritage, “Task,” *American Heritage Dictionary of English Language, Fourth Edition*. Houghton Mifflin Company. www.answer.com/topic/task, 2007. B
- [4] R. Bartk, “Constraint Programming: In Pursuit of the Holy Grail,” in *Proceedings of the Week of Doctoral Students (WDS99)*, no. IV. Prague: MatFyzPress, June 1999, pp. 555–564. B
- [5] D. Bates, S. Silverman, J. Jeter, and K. Blasius, “Performance of the Miniature Thermal Emission Spectrometer (Mini-TES) for Mars 2001 Lander,” in *2000 IEEE Aerospace Conference Proceedings*, vol. 3, Big Sky, MT, USA, March 2000, pp. 135–149. C

- [6] C. Bessière, “Arc consistency in dynamic constraint satisfaction problems,” in *Proceedings of the Ninth National Conference on Artificial Intelligence*, Anaheim, California, July 14-19 1991, pp. 221–226. C
- [7] C. Bessière, “Arc consistency for Non-Binary Dynamic CSPs,” in *Proceedings of the Tenth European Conference on Artificial Intelligence*, Vienna, Austria, August 3-7 1992, pp. 23–27. C
- [8] P. Black, K. Hall, M. Jones, T. Larson, and P. Windley, “A Brief Introduction to Formal Methods,” in *Proceedings of CICC*, San Diego, CA, May 1996, pp. 377–380. C
- [9] G. Bonanno, R. Fantoni, A. Fichera, G. Fornetti, C. Moriconi, C. Poggi, M. Caponero, A. Broggi, and A. Fascioli, “The Sensing Subsystem of RAS,” in *Atti del Meeting Nazionale sulle Nuove Tecnologie*, Frascati, Italy, May 2003, pp. 1–4. C
- [10] A. Borning, “The Programming Language Aspects of ThingLab, a Constraint-Oriented Simulation Laboratory,” *ACM Transactions on Programming Languages and Systems*, vol. 3, no. 4, pp. 353–387, October 1981. C
- [11] J. P. Bowen and M. G. Hinchey, “Seven More Myths of Formal Methods,” *IEEE Software*, vol. 12, no. 3, pp. 34–41, 1995. C
- [12] D. J. Bruemmer, D. D. Dudenhofer, and J. L. Marble, “Dynamic autonomy for urban search and rescue,” in *AAAI Mobile Robot Workshop*, Edmonton, Canada, August 2002, pp. 1–5. C
- [13] S. Burion, “Human detection for robotic urban search and rescue,” Master’s thesis, Robotics Institute, Carnegie Mellon University, 2004. Thesis

- [14] F. Carsey, P. Schenker, J. Blamont, S. Gogineni, K. Jezek, C. Rapley, and W. Whittaker, "Autonomous Trans-Antarctic expeditions: an initiative for advancing planetary mobility system technology while addressing earth science objectives in Antarctica," in *5th International Symposium on Artificial Intelligence and Autonomy in Space*, no. AM017, Montreal, Canada, June 2001. C
- [15] J. Casper, M. Micire, and R. R. Murphy, "Issues in Intelligent Robots for Search and Rescue," in *Proceedings of SPIE Ground Vehicle Technology II*, 2000. C
- [16] Center for Astrophysical Research in Antarctica, "Current South Pole Weather Data," URL: <http://astro.uchicago.edu/cara/southpole.edu/weather2.html>, 2007. X
- [17] Center for Remote Sensing of Ice Sheets, "Science Requirements for Field Work in CReSIS," Center for Remote Sensing of Ice Sheets, University of Kansas, Tech. Rep. Draft v. 0.6, September 2005. Tech Report
- [18] Center for Remote Sensing of Ice Sheets, "CReSIS Technology Requirements," Center for Remote Sensing of Ice Sheets, University of Kansas, Tech. Rep. Version 2.0, February 2006. T. Report
- [19] Center for Robot-Assisted Search and Rescue, "CRASAR - University of Southern Florida," URL: <http://crasar.csee.usf.edu>, 2007. X
- [20] H. Choxi, M. Lomas, J. Franke, and K. Whitebread, "Using motivations for interactive robot behavior control," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, Orlando, FL, May 15-19 2006, pp. 1-3. C

- [21] H. W. Chun and R. Y. M. Wong, "CLSS: An Intelligent Crane Lorry Scheduling System," *Applied Intelligence*, vol. 20, no. 2, pp. 179–194, March - April 2004. J
- [22] A. Colmerauer, "Introduction to Prolog III," in *ESPRIT '87*, North Holland, 1987, pp. 611–629. J
- [23] Cosytec, "Chip V5," URL: http://www.cosytec.com/production_scheduling/chip/optimization_product_chip.htm, 2007. X
- [24] CReSIS, "Center for Remote Sensing of Ice Sheets," URL: <http://www.cresis.ku.edu>, 2007. X
- [25] A. Dechter and R. Dechter, "Belief Maintenance in dynamic constraint networks," in *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, St. Paul, MN, August 1988, pp. 37–42. C
- [26] R. Dechter and J. Pearl, "Tree-clustering schemes for constraint-processing," in *Proceedings of AAAI*, St. Paul, MN, August 21–26 1988, pp. 150–154. C
- [27] R. Dechter, "Constraint Networks," in *Encyclopedia of Artificial Intelligence*, S. C. Shapiro, Ed. New York: Wiley, December 1991, pp. 276–285. B
- [28] S. Deris, S. Omatu, H. Ohta, and P. Samat, "University Timetabling by Constraint-based Reasoning: A Case Study," *The Journal of Operational Research Society*, vol. 48, no. 12, pp. 1178–1190, 1997. J
- [29] M. S. design group, "The Alloy Analyzer - 3.0 Beta," <http://sdg.lcs.mit.edu/alloy/>, 2007. X

- [30] B. Espiau, K. Kapellos, M. Jourdan, and D. Simon, “On the Validation of Robotics Control Systems. Part I: High level Specification and Formal Verification,” Inria Research, Tech. Rep. 2719, November 1995. *T. Report*
- [31] E. C. Freuder, “A sufficient condition for backtrack-free search,” *Journal of the ACM*, vol. 29, no. 1, pp. 24–32, 1982. *J*
- [32] B. P. Gerkey and M. J. Mataric, “A Formal Analysis and Taxonomy of Task Allocation in Multi-Robot Systems,” *The International Journal of Robotics Research*, vol. 23, no. 9, pp. 939–954, 2004. *J*
- [33] C. M. Gifford and A. Agah, “Robotic Deployment and Retrieval of Seismic Sensors for Polar Environments,” in *Proceedings of the 4th International Conference on Computing, Communications and Control Technologies (CCCT)*, vol. II, Orlando, FL, July 2006, pp. 334–339. *C*
- [34] I. Hayes, “VDM and Z: A comparative case study,” *Formal Aspect of Computing*, vol. 4, pp. 76–99, 1992. *J*
- [35] F. Hayes-Roth, “Rule-based Systems,” *Communications of the ACM*, vol. 28, no. 9, pp. 921–932, September 1985. *J*
- [36] ILOG Inc., “ILOG Solver,” URL: <http://www.ilog.com/products/solver/>, 2007. *X*
- [37] D. Jackson, I. Schechter, and I. Shlyakhter, “Alcoa: the Alloy Constraint Analyzer,” in *Proceedings of the International Conference on Software Engineering*, Limerick, Ireland, June 2000, pp. 730–733. *C*
- [38] A. Jacoff, B. Weiss, and E. Messina, “Evolution of a Performance Metric for Urban search and Rescue Robots,” in *Proceedings of the 2003 Performance* *C*

Metrics for Intelligent Systems Workshop, Gaithersburg, MD, August 16-18 2003, pp. 1–10.

[39] JavaCC, “JavaCC Project Home,” URL: <http://javacc.dev.java.net>, 2007. X

[40] C. B. Jones, *Systematic Software Development using VDM*. New Jersey: Prentice-Hall, 1990. B

[41] Koalog, “Koalog Constraint Solver API documentation,” URL: <http://www.koalog.com/resources/doc/jcs-javadoc.jar>, 2007. X

[42] Koalog SARL, “Koalog,” URL: <http://www.koalog.com/php/index.php>, 2007. X

[43] J. Kosecka and R. Bajcsy, “Discrete Event Systems for Autonomous Mobile Agents,” *Journal of Robotics and Autonomous Systems*, vol. 12, pp. 187–198, 1994. J

[44] J. Kosecka, H. I. Christensen, and R. Bajcsy, “Experiments in Behavior Composition,” *Journal of Robotics and Autonomous Systems*, vol. 19, pp. 187–198, 1994. J

[45] V. Kumar, “Algorithms for Constraint Satisfaction Problems: A Survey,” *AI Magazine*, vol. 13, no. 1, pp. 32–44, 1992. J

[46] P. G. Larsen, J. Fitzgerald, and T. Brookers, “Applying Formal Specification in Industry,” *IEEE Software*, vol. 13, no. 7, pp. 48–56, 1996. J










[47] A. K. Mackworth, “Consistency in networks of relations,” *Artificial Intelligence*, vol. 8, no. 1, pp. 99–118, 1997. J

- [48] A. K. Mackworth, "Constraint-based design of embedded intelligent systems," *Constraints*, vol. 2, pp. 83–86, 1997. J
- [49] MathWorks, "The Math Works - Statistics ToolBox - Applying statistical algorithms and probability models." URL: <http://www.mathworks.com/products/statistics/>, 2007. X
- [50] M. J. Matric, "Behavior-Based Robotics," in *MIT Encyclopedia of Cognitive Science*, R. A. Wilson and F. C. Keil, Eds. Cambridge, MA: MIT Press, April 1999, pp. 74–77. B
- [51] T. McGibon, "An Analysis of Two Formal Methods: VDM and Z," DoD Data and Analysis Center for Software, Rome, NY, Tech. Rep. DACS-CRTA-97-1, August 1997, <http://www.dacs.dtic.mil/techs/2fmethods/vdm-z.pdf>. T. Report
- [52] S. Minton, M. D. Johnston, A. B. Philips, and P. Laird, "Minimizing conflicts: a heuristic repair method for constraint-satisfaction and scheduling problems," *Artificial Intelligence*, vol. 58, no. 1-3, pp. 161–205, 1992. J
- [53] S. Moorehead, R. Simmons, D. Apostolopoulos, and W. R. L. Whittaker, "Autonomous Navigation field results of a planetary analog robot in Antarctica," in *International Symposium on Artificial Intelligence, Robotics and Automation in Space*, Noordwijk, The Netherlands, June 1999, pp. 237–242. C
- [54] R. R. Murphy, "Trial by Fire: Activities of the Rescue Robots at the World Trade Center from 11-21 September 2001," *IEEE Robotics and Automation Magazine*, pp. 50–61, September 2004. J

- [55] National Institute of Standards and Technology, “Dictionary of Algorithms and Data Structures,” <http://www.nist.gov/dads/>, 2007. ✕
- [56] National Renewable Energy Laboratory, *Small Wind Electric Systems: A U.S. Consumer’s Guide*. U.S. Department of Energy, National Renewable Energy Laboratory, March 2005, no. DOE/GO-12005-2095. ✕
- [57] D. K. Pai, “Programming parallel distributed control for complex systems,” in *IEEE International Symposium on Intelligent Control*, September 25-26 1989, pp. 426–432. C
- [58] D. Pai, “Least Constraint: A framework for the control of complex mechanical systems,” in *Proceedings of the American Control Conference*, Boston, MA, June 1991, pp. 615–621. C
- [59] P. Pirjanian, “Multiple Objective action Selection & Behavior Fusion using Voting,” Ph.D. dissertation, Aalborg University Denmark, April 1998. Thesis
- [60] PRISM, “Polar Radar for Ice Sheet Measurement (PRISM) project,” URL: <http://ku-prism.org>, 2007. ✕
- [61] L. Purvis, “Dynamic Constraint Satisfaction Using Case-Based Reasoning Techniques,” in *Proceedings of the Constraint Programming ’97 Workshop on Dynamic Constraint Satisfaction*, Schloss Hagenberg, Austria, October 1997, pp. 1–14. C
- [62] L. Purvis, “A CBR Integration from Inception to Productization,” in *Proceedings of the AAAI’98 Workshop on Case-Based Reasoning Integrations*, Madison Wisconsin, July 1998, pp. 1–6. C

- [63] L. Ray, A. Price, A. Streeter, D. Denton, and J. H. Lever, "The Design of a Mobile Robot for Instrument Network Deployment in Antarctica," in *Proceedings of the 2005 International Conference on Robotics and Automation (ICRA 2005)*, Barcelona, Spain, April 2005, pp. 2111–2116. C
- [64] M. Richters and M. Gogolla, *Advances in Object Modelling with the OCL*. Berlin: Springer, 2001, ch. OCL - Syntax, Semantics and Tools. J
- [65] A. Robotics, "P3-AT: The High Performance All-Terrain Robot," URL: <http://www.activerobots.com/ROBOTS/p2at.html>, 2007. X
- [66] F. Rossi, C. Petrie, and V. Dhar, "On the Equivalence of Constraint-Satisfaction Problems," MCC Corp., Austin, Texas, Tech. Rep., 1989, technical Report ACT-AI-222-89. T. Report
- [67] S. Russel and P. Norvig, *Artificial Intelligence: A Modern Approach*, Second ed. Upper Saddle River, NJ: Prentice Hall, 2002. B
- [68] Scintrex, "CG-5 Gravity Meter," Online: <http://www.scintexltd.com>, 2007. X
- [69] R. Simmons and D. Apfelbaum, "A task description language for robot control," in *IEEE/RSJ International Conference on intelligent Robots and Systems*, vol. 3, Victoria, B.C., Canada, Oct 13-17 1998, pp. 1931–1937. C
- [70] Southeast Arizona Emergency Medical Service Council, "Disaster Triage Protocol," URL: <http://www.saems.net/protocols/>, 2007. X
- [71] Southwest Windpower, "Air-X Land / Air-X Marine: Technical Specification," URL: <http://www.windenergy.com/>, 2007. X
- [72] J. Spivey, *The Z Notation: Reference Manual*, 2nd ed., ser. Series in computer science. Prentice Hall International, Oxford 1998. B

- [73] M. Sqalli, L. Purvis, and E. Freuder, "Survey of Applications Integrating Constraint Satisfaction and Case-Based Reasoning," in *Proceedings of the First International Conference on the Practical Application of Constraint Technologies and Logic Programming*, London, UK, April 1999, pp. 1–20. C
- [74] R. S. Stansbury and A. Agah, "Constraint-based Task Selection for Autonomous Urban Search and Rescue Robots," *International Journal on Autonomous Robots*, 2007, in review. J
- [75] R. S. Stansbury and A. Agah, "Constraint-based Task Selection for Long-Term Survival of Autonomous Polar Robot," *Journal of Intelligent and Robotic Systems*, 2007, in review. J
- [76] R. S. Stansbury, E. L. Akers, and A. Agah, "Simulation and Testbeds of Autonomous Robots in Harsh Environments," in *Software Engineering for Experimental Robots*, ser. Springer Tracts on Advanced Robotics, D. Brugali, Ed. Berlin: Springer, 2007, vol. 30. J
- [77] R. S. Stansbury, E. L. Akers, H. P. Harmon, and A. Agah, "Survivability, mobility, and functionality of a rover for radars in polar regions," *International Journal of Control, Automation, and Systems*, vol. 2, no. 3, pp. 334–353, 2004. J
- [78] G. Steele, "The definition and implementation of a computer programming language based on constraints," Ph.D. dissertation, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Mass., 1980. Thesis
- [79] M. Sugeno, "An introductory survey of fuzzy control," *Information Sciences*, vol. 36, pp. 59–83, 1985. J

- [80] Sun Microsystems Inc., “Java Standard Edition,” 2007. 
- [81] M. Syrjanen, “Production management as a constraint satisfaction problem,” *Journal of Intelligent Manufacturing*, vol. 9, pp. 515–522, 1998. 
- [82] E. Tsang, *Foundations of Constraint Satisfaction*, ser. Computation in Cognitive Science. London: Academic Press, 1993. 
- [83] G. Verfaillie and T. Schiex, “Solution reuse in dynamic constraint satisfaction problems,” in *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Seattle, WA, August 1994, pp. 307–312. 
- [84] Webster-Merriam, “Webster-Merriam Online Dictionary,” URL: <http://www.w-m.com>, 2007. 
- [85] D. Wettergreen, P. Tompkins, C. Urmson, M. Wagner, and W. Whittaker, “Sun-synchronous robotic exploration: technical description and field experimentation,” *International Journal of Robotics Research*, vol. 24, no. 1, pp. 3–30, January 2005. 
- [86] Y. Zhang and A. K. Mackworth, “Will the robot do the right thing?” in *Proceedings of Artificial Intelligence*, vol. 138, 1994, pp. 255–262. 
- [87] Y. Zhang and A. K. Mackworth, “Constraint nets: a semantic model for hybrid dynamic systems,” *Theoretical Computer Science*, vol. 138, no. 1, pp. 211–239, 1995. 
- [88] Y. Zhang and A. K. Mackworth, “A Constraint-based robotic soccer team,” *Constraints*, vol. 7, pp. 7–28, 2005. 

STUDENT'S T-TEST RESULTS

The following tables present the Student's t-test confidence when comparing the mean values for each metric under each configuration. If the constraint-based system outperformed the rule-base system with a 95% confidence, the $CSP > Rule$ column is marked. Likewise, if the mean of the rule-base is greater than the constraint-base, the $CSP < Rule$ column is marked. If neither one outperformed the other at 95% confidence, the result is marked as unknown. The confidence for each known conclusion is listed. The Student's t-test results for the urban search and rescue scenario are presented in Tables 7 and 8. The Student's t-test results for the polar scenario are presented in Tables 5 and 6. The results of the Student's T-test analysis for the delivery scenario is presented. Four tables exist that present the results by metric. Table 1 presents the results for failure. Table 2 presents the results for collisions. Table 3 presents the results for execution time. Table 4 presents the results for resource usage.

	CSP < Rule	Rule < CSP	Unknown	Confidence
Static				
None	x			100.00
Avoidance	x			100.00
Drive	x			100.00
Drive + Avoidance	x			100.00
Payload	x			100.00
Payload + Avoidance	x			100.00
Payload + Drive	x			100.00
Dynamic				
None	x			99.93
Avoidance	x			100.00
Drive	x			100.00
Payload	x			100.00
All Faults	x			100.00

Table 1: Delivery scenario: failure rate Student's T-test results

	CSP < Rule	Rule < CSP	Unknown	Confidence
Static				
None	x			100.00
Avoidance	x			100.00
Drive	x			97.22
Drive + Avoidance	x			100.00
Payload	x			97.80
Payload + Avoidance	x			99.75
Payload + Drive			x	N/A
Dynamic				
None	x			100.00
Avoidance	x			100.00
Drive	x			99.19
Payload			x	N/A
All Faults	x			99.74

Table 2: Delivery scenario: collisions Student's T-test results

	CSP < Rule	Rule < CSP	Unknown	Confidence
Static				
None		x		100.00
Avoidance		x		100.00
Drive		x		100.00
Drive + Avoidance		x		100.00
Payload		x		100.00
Payload + Avoidance		x		100.00
Payload + Drive		x		100.00
Dynamic				
None		x		100.00
Avoidance		x		100.00
Drive		x		100.00
Payload		x		100.00
All Faults		x		100.00

Table 3: Delivery scenario: execution time Student's T-test results

	CSP < Rule	Rule < CSP	Unknown	Confidence
Static				
None		x		100.00
Avoidance		x		100.00
Drive		x		100.00
Drive + Avoidance		x		100.00
Payload		x		100.00
Payload + Avoidance		x		100.00
Payload + Drive		x		100.00
Dynamic				
None		x		100.00
Avoidance		x		100.00
Drive		x		100.00
Payload		x		100.00
All Faults		x		100.00

Table 4: Delivery scenario: resource consumption Student's T-test results

	CSP > Rule	Rule > CSP	Unknown	Confidence(%)
POLAR1				
Survival Time	X			100
Complete Tasks		X		100
Started Tasks			X	N/A
Failure Rate		X		99.93
Percent Complete	X			99.96
POLAR2				
Survival Time	X			100
Complete Tasks		X		100
Started Tasks		X		99.89
Failure Rate		X		100
Percent Complete		X		100
POLAR3				
Survival Time	X			99.99
Complete Tasks		X		99.73
Started Tasks			X	N/A
Failure Rate		X		99.93
Percent Complete	X			99.95
POLAR4				
Survival Time	X			99.99
Complete Tasks			X	N/A
Started Tasks		X		99.63
Failure Rate		X		99.98
Percent Complete			X	N/A
POLAR5				
Survival Time	X			99.99
Complete Tasks	X			99.85
Started Tasks	X			99.17
Failure Rate		X		99.98
Percent Complete	X			100

Table 5: Polar scenario: Student's t-test results

	CSP > Rule	Rule > CSP	Unknown	Confidence(%)
POLAR6				
Survival Time	X			99.84
Complete Tasks			X	N/A
Started Tasks			X	N/A
Failure Rate		X		99.73
Percent Complete	X			99.95

Table 6: Polar scenario: Student's t-test results (continued)

	CSP > Rule	Rule > CSP	Unknown	Confidence(%)
USAR1				
Rescues			X	N/A
Hazards			X	N/A
Locations	X			97.34
Collisions			X	N/A
Score			X	N/A
USAR2				
Rescues			X	N/A
Hazards	X			96.71
Locations	X			99.96
Collisions		X		99.71A
Score	X			95.99
USAR3				
Rescues			X	N/A
Hazards			X	N/A
Locations			X	N/A
Collisions		X		99.17
Score			X	N/A
USAR4				
Rescues			X	N/A
Hazards			X	N/A
Locations			X	N/A
Collisions		X		99.98
Score			X	N/A
USAR5				
Rescues			X	N/A
Hazards			X	N/A
Locations			X	N/A
Collisions			X	N/A
Score	X			95.75

Table 7: USAR scenario: Student's t-test results

	CSP > Rule	Rule > CSP	Unknown	Confidence(%)
USAR6				
Rescues	X			96.12
Hazards	X			95.01
Locations	X			98.55
Collisions		X		N/A
Score	X			97.53
USAR7				
Rescues			X	N/A
Hazards			X	N/A
Locations			X	N/A
Collisions			X	N/A
Score			X	N/A
USAR8				
Rescues	X			95.87
Hazards	X			96.4
Locations	X			97.44
Collisions			X	N/A
Score	X			97.44
USAR2 Fixed				
Rescues		X		100
Hazards		X		95.05
Locations		X		99.87
Collisions			X	N/A
Score		X		1
USAR4 Fixed				
Rescues		X		99.95
Hazards		X		97.52
Locations		X		99.14
Collisions		X		99.66
Score		X		99.9

Table 8: USAR scenario: Student's t-test results (continued)