Office of Graduate Studies
University of South Florida
Tampa, Florida

CERTIFICATE OF APPROVAL

Master's Thesis

This is to certify that the Master's Thesis of

AARON GAGE

with a major in Computer Science and Engineering has been approved
for the thesis requirement on February 27, 2001
for the Master of Science in Computer Science degree.

Examining Committee:

Major Professor: Robin Murphy, Ph.D.

Member: Kevin Bowyer, Ph.D.

Member: Lawrence Hall, Ph.D.

MOBILE ROBOT SENSOR ALLOCATION USING MIN-CONFLICT WITH HAPPINESS

by

AARON GAGE

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science
Department of Computer Science and Engineering
College of Engineering
University of South Florida

August 2001

Major Professor: Robin Murphy, Ph.D.

DEDICATION

To my family, for years of unwavering support and understanding.

ACKNOWLEDGMENTS

This thesis would never have been possible without the support of Robin Murphy, Ph.D., and the rest of the Computer Science and Engineering Department at the University of South Florida. I would also like to acknowledge those who, during my years of work in the USF and CSM robotics labs alike, have made the time spent more enjoyable.

This thesis was also supported in part by NSF Grant IRI-9320318, DARPA Grant AO#B460, and ONR/NRL N00173-99-P-1543.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

MOBILE ROBOT SENSOR ALLOCATION USING
MIN-CONFLICT WITH HAPPINESS

by

AARON GAGE

An Abstract

of a thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science
Department of Computer Science and Engineering
College of Engineering
University of South Florida

August 2001

Major Professor: Robin Murphy, Ph.D.

For mobile robots employing reactive behaviors, allocation of physical sensors to satisfy sensing needs should be dynamic and fast. However, the allocation mechanism may not have access to the planning or other deliberative components of a robot, and should be able to function without knowledge of future sending needs. The Min-Conflict with Happiness algorithm allows for physical sensors to be allocated to percepts quickly and with incomplete information, supports behavioral sensor fusion, and tolerates sensor failure.

This algorithm has been used successfully in mobile robot applications, and has been demonstrated twice at AAAI robot competitions. Experiments show that the algorithm is fast and finds solutions providing high utility ("happiness"), while taking measures to minimize the overhead of making assignments due to changes in the sensing state.

Abstract Approved: _____

Major Professor: Robin Murphy, Ph.D.
Associate Professor

Date Approved: _____

CHAPTER 1

INTRODUCTION

The application of autonomous mobile robots to real-world problems presents a number of challenges. The formulation of goals, planning of solutions, and execution of those plans are confounded by physical limitations and incomplete knowledge. Even in human terms, a plan that seems reasonable may be impossible due to unforeseen details: paths may be impassable, equipment may fail, or the basic premises upon which the plan was formed may be false. These are difficult problems for humans to face, and yet, an autonomous robot must be equipped to deal with these problems to survive in the real world.

An important quality that allows humans to compensate for these confounding effects is the ability to make dynamic changes to plans and to improvise when equipment fails. Providing a mobile robot with the same characteristics improves its tolerance for unpredictable events; dynamic changes may allow a robot to achieve a goal even if false assumptions and incomplete knowledge cause its initial plans to be incomplete or incorrect. In this way, the *robustness* of a robotic system benefits from the robot's ability to recover when plans or equipment fail.

This robustness can be achieved by scheduling the use of the robot's hardware to accommodate its task and working conditions. By monitoring the robot's state and changing its schedule in response to changes in the environment or hardware, a new degree of adaptivity is possible.

The problem described here is oriented toward scheduling, as opposed to planning. The distinction is that the plan that is being executed relies on the scheduling of

resources. If a robot recognizes that the resource schedule is broken, then it may be able to recover without interrupting, or altering, execution of the plan.

1.1 Mobile Robots and Perception

Modern mobile robots are equipped with numerous resources, typically including a sensor suite. Intelligent management of these resources is necessary for the completion of robotic tasks, such that the robot can adequately sense both its environment and its progress. However, the scheduling of sensors is not a trivial task. The perceptual needs of a robot will depend on its particular course of action, and changes to a plan may change these needs. The problem of allocating sensors in an appropriate and intelligent manner resembles some scheduling problems in Artificial Intelligence (AI), but considerations must be made regarding the physical world with which a mobile robot interacts.

The allocation problem for sensors requires a form of scheduling. According to Fox, "scheduling selects among alternative plans and assigns resources and times for each activity so that the assignments obey the temporal restrictions of activities and the capacity limitations of a set of shared resources."[1] In this case, the "alternative plans" are the different possible assignments, and the resources are sensors. In other words, the difficulty lies in choosing the best assignment of sensors to percepts (the sensing strategy) so that the resulting assignments are possible and allowable given the state of the robot.

This is in contrast to planning. According to Fox, "planning selects and sequences activities such that they achieve one or more goals and satisfy a set of domain constraints." [2] In this case, the "activities" refer to sensing. The problem of planning is more deliberative, dealing with the task to be accomplished, and anticipating what

[1] Taken from [7], page 3.
[2] Taken from [7], page 3.

sensing will be needed at each step. For every task in a plan, a separate schedule may be needed.

Of the hardware available on a mobile robot, sensors are particularly interesting for allocation. A robot may have sensors that are somewhat redundant; one may provide the same kind of percept as another, though the sensors may work in entirely different ways. This allows some flexibility; if one sensor fails, another may provide the same kind of perception, though perhaps with less accuracy or resolution. This substitution may be direct, meaning that the reading from one sensor can replace a reading from another with no change. The substitution may also be algorithmic, such that the output of one physical sensor can be manipulated to emulate the output of another. It may also be possible to combine multiple sensor readings into one percept through *sensor fusion*.

In the ideal case, the sensing needs of a mobile robot may be predictable, in accordance with a plan for action. In the real world, however, unforeseen events may cause a plan to break and require repair. For instance, a path may be blocked, hardware may fail, or the environment may change in a way that makes following the plan impossible. Therefore, the allocation of resources should be dynamic, changing to suit the robot's ideal plan in terms of the reality of the robot's hardware and environment.

A particular heuristic which allows repair-based allocation in a reactive manner is the *Min-Conflict* heuristic [21], which is effective in the domain of Constraint Satisfaction Problems (CSP)[12]. This method for allocation has many features which make it favorable to the domain of robotic sensor allocation, though it does not directly address some of the issues related to actual sensor performance. Min-Conflict provides an appropriate starting point for sensor allocation, but it must be modified to work in the domain of robotics.

1.2 Research Question

The research question that this thesis will address is thus:

> How can perceptual resources be dynamically allocated on an autonomous mobile robot, using incomplete information, such that it is tolerant of sensor failure and unpredictable requests?

This question involves a number of related issues pertaining to mobile robots and scheduling.

- Representation: In order to allocate resources on a robot, there must be some representation of those resources as well as the constraints that the allocations must obey. Identifying a method of representation of the sensors and assignments is necessary for a solution. This method should also be extensible to allow for future enhancements.

- Search: Locating a solution to a given allocation scenario requires that different possibilities are considered, a process that takes the form of a search. While there are many different accepted methods for searching through a problem space, determining which of these is best suited for the mobile robot domain is necessary for the design.

- Utility: As with other scheduling problems, there may be multiple solution states that could end a search. However, the quality of the solutions may vary. For this domain, it is necessary to both characterize and utilize the utility of sensors for a particular perceptual task, and to allocate accordingly.

1.3 Other Requirements

There are also a number of requirements that a satisfactory solution must meet that deal with performance on a mobile robot.

First, there should be support for *behavioral sensor fusion* [1], which is the combination of multiple sensor readings to filter out noise and to provide percepts that cannot be measured by a single sensor. Behavioral Sensor Fusion allows for a greater utilization of sensors that might be unused otherwise, and provides an improved state of sensing.

In order to be compatible with reactive and hybrid deliberative/reactive robot architectures, the allocation of resources should be reactive. In other words, in order to deal with the unpredictable needs of reactive behaviors (whose activity is driven by changes in the perceived environment), the allocation must work without access to deliberative mechanisms (such as task planning or goal formulation). The allocation mechanism should satisfy demands imposed at a higher level, but will not be involved in the process of formulating those demands or be capable of contesting them. An implication of this constraint is that scheduling is done without complete information. That is, it must react to requests without reasoning about why they are made or predicting the sequence or duration of those requests. Other methods for allocating sensors on a mobile robot exist, but many rely on the prediction of perceptual needs over time, or upon the involvement of the allocation mechanism in the planning stage. Min-Conflict can be used without any form of prediction, and functions entirely in a repair capacity, which makes it inherently appropriate for this domain.

Another issue deals with the unfortunate, but unavoidable, problem of sensor failure. Either as a result of damage to a robot or drastic changes in the environment, sensing hardware may no longer perform as required. These events, which cannot be easily planned for, can cause the particular assignment of sensors to tasks to become unsatisfactory, which may in turn lead to the failure of a plan. In the event of hardware failure, the allocation of sensors should be quickly repaired, if possible, to allow a plan to continue.

Next, sensors that are logically equivalent [5] should be interchangeable. This provides the flexibility needed to overcome sensor failures, or the contention for sensors which are simultaneously needed for divergent and incompatible tasks. This logical equivalence may require a software component that allows one sensor's percepts to replace another. Further, if there is any advantage to assigning one sensor over another for the same perceptual task, then the process of allocation should consider the relative utility of each possible assignment and select the highest utility. This also requires some measure of sensor utility.

Finally, once a particular allocation has been made, it should not be broken to satisfy another allocation unless there is no other option. This is the principle of *least disturbance*, which is intended to minimize the overhead of the allocation process. Without this consideration, the system may be susceptible to *thrashing*, meaning that the cost of overhead overshadows the time spent doing meaningful sensing. Thrashing refers to a state in which allocations are made faster than a sensor can be effectively used.

1.4 Approach Taken

The approach in this thesis is to modify the Min-Conflict heuristic in order to handle the allocation of sensors. Min-Conflict was chosen because of its successful use in solving constraint-satisfaction problems and because of its performance compared to generative backtracking techniques, including "most-constrained first" search [20]. In order to answer this research question, Min-Conflict was modified in three particular ways. First, the differences between logically equivalent sensors were accounted for by generating a utility measure for each possible allocation, based on an *a priori* preference ordering. Next, the repair of schedules was sequenced in a manner that provided for *least disturbance* of existing assignments. Sensor fusion was then addressed, and a more objective measure of sensor utility was found, both of which provide greater

functionality on top of the Min-Conflict heuristic. The resulting system was tested in simulation, then implemented on two heterogeneous mobile robots that entered in the 1999 and 2000 AAAI "Hors d'oeuvres, Anyone?" competitions, winning two technical achievement awards and one third place award.

One intent with the design of this algorithm has been its integration into the *Sensor Fusion Effects* (SFX) robotic architecture [22]. Many of the features of the algorithm are in fact necessary for this integration, including its non-predictive scheduling and its capacity for dynamic changes to the schedule. Within the SFX architecture, the Min-Conflict with Happiness algorithm can be used to build the Sensor Manager that chooses the sensors that each reactive behavior will be allowed to use.

1.5 Contribution

The contribution of this work is a dynamic algorithm capable of allocating physical sensors to logical sensors [10] in a manner that supports behavioral sensor fusion, tolerates sensor failures, and that prevents thrashing, while requiring no advance information about future sensing needs. This thesis also contributes a quantitative representation of sensing quality using t-norms [19], allowing fused sensors to be compared with single sensors for the same percept.

CHAPTER 2

APPROACH

Min-Conflict with repair was chosen as a foundation for this work because of its effectiveness in solving problems that are highly constrained, and because of its intuitive nature. Min-Conflict was detailed in [12], and will be described below. It was necessary to make changes to the algorithm in order to represent sensors more appropriately, as Min-Conflict typically deals with homogeneous assignments while sensors are decidedly heterogeneous.

2.1 Constraint Satisfaction Problems

The class of Constraint Satisfaction Problems (CSP) contains problems where assignments are made (such as queens to a chess-board in the n-queens problem), but certain relationships or states are illegal (such has having two queens in the same row, column, or diagonal). These illegal states are constraints, and to solve the problem implies satisfying all assignments (placing all n queens) without violating any constraints [26]. Other problems in this domain include scheduling and graph 3-colorability.[12]

Scheduling is of particular interest here, because the allocation problem for sensors can be expressed in terms of scheduling. Sensors are assigned to particular percepts, but some assignments are illegal, or constrained, due to unsharable resources or physically incompatible demands. Solving the scheduling problem for sensors requires that all requests be satisfied (meaning that all assignments are made) but without leaving behind illegal allocations.

There are different approaches that can solve problems of this nature, some of which are based on a depth-first search [26]. The first is a simple, generative backtracking method. Using n-queens as an example, this method would place each of the n queens in its own column on a chess board in any position that did not violate a constraint. In this case, violating a constraint means putting a queen in the same row or column as another previously placed queen. If there are multiple possible positions in which to place a queen, one is chosen randomly. This would continue until all queens had been placed, or until there were no legal positions for the next queen (that is, the locations of the first k queens leave no legal positions for the $k+1$ queen). In this case, the algorithm would backtrack, moving the most recently placed queen and trying to place it differently, as doing so may leave a spot for the next queen. If all possibilities were exhausted in this fashion (that is, the last legally placed queen had been tried in all of its own legal locations), the algorithm would backtrack further [26], effectively unraveling its own solution when it reaches a dead-end. A variant on this heuristic is the "most-constrained first" approach [26, 27], in which the branching factor of the search is reduced by choosing the most constrained row as the next one on which to place a queen.

The Min-Conflict heuristic would begin solving this problem in the same manner, placing each successive queen in a legal position in its own column until it reached a point where there were no legal moves for the next queen. Instead of backtracking, this heuristic would place the next queen in the position that produced the fewest *conflicts* with other assignments; that is, the position that shared rows or diagonals with the fewest other queens, breaking ties randomly. Once all queens had been placed in this manner, each queen whose assignment violated a constraint (that is, each queen that was in conflict with another) would be moved to a position in the same column that did not increase the number of violations. By applying this method iteratively, the total number of conflicts will reduce until all constraints are satisfied.

There is at least one qualitative advantage of Min-Conflict over the generative backtracking method. That is, Min-Conflict is repair-based and allows all assignments to be made before any are repaired, meaning that any changes are more informed than in the generative case (where only a subset of the assignments are made before repairs). As a result, Min-Conflict with repair tends to reach solutions faster than generative backtracking methods. A comparison of Min-Conflict to other algorithms for solving constraint satisfaction problems, including generative backtracking and "most-constrained first", can be found in [20]. Minton, et al also describe using Min-Conflict with informed backtracking, which resembles the approach used in this thesis, that they claim is complete (that is, it will find a solution if one exists) [20].

2.2 Sensor Allocation

The approach taken in this thesis uses a modified Min-Conflict with repair heuristic to allocate physical resources to logical sensors [10]. The possible assignments for each logical sensor are each of the physical sensors with which the logical sensor can produce a desired percept. Conflict refers to the assignment of an unsharable physical resource to multiple logical sensors, and repair consists of reducing conflicts through the substitution of other physical sensors for a particular percept. The changes that were made to Min-Conflict were needed to capture the inherent heterogeneity of robotic sensing hardware and algorithms. These issues are described below.

- Utility: A logical sensor may be more accurate or more descriptive given a particular physical sensor. This relative utility should be maximized across assignments, providing each logical sensor with the best alternative available, and ties should be broken based on utility. This utility is the "Happiness" of the assignments, and is the namesake of the algorithm.

- Flexibility: The flexibility of a logical sensor is based on the different ways that it can generate a percept. In other words, a logical sensor is considered more *flexible* when it has more options for sensing. For instance, a logical sensor that measures distances from the robot to objects in the environment may have many ways of doing so: sonar transducers, infrared sensors, laser rangers, and even tactile sensors can measure distances directly, and vision systems can estimate distance from depth of focus, optic flow, stereo, convergence, and structured light techniques. In this case, such a logical sensor may be considered very *flexible*. A logical sensor that must detect orange safety vests (or any other color fiducial or landmark) is limited to using a camera in combination with some vision algorithm. If there are no available cameras, this logical sensor will be unable to function. By virtue of depending on a single physical sensor, this logical sensor is not *flexible*.

- Least Disturbance: Each time a physical sensor is allotted, its parameters may need to be changed to suit its new task. In addition, the algorithms that produce a percept from that sensor may incur some overhead (reading parameters from a file, for instance). To minimize this overhead cost, assignments should be disturbed as little as possible once they are made. Otherwise, the time spent setting up a particular sensor for use may become significant compared to the time spent using the sensor, which will be referred to as *thrashing*. For example, suppose that a robot's cameras may be used for hallway navigation as well as for object recognition, but not both at once. If the cameras are allocated to navigating the hallway, then a newly active object recognition task should not interrupt the navigation task if there is some other alternative. Of course, if there is no other option, then the navigation task may be interrupted.

- Behavioral Sensor Fusion: The combination of readings from different sensors is useful for improving the quality of perception. The allocation scheme should allow multiple physical sensors to be allocated together for a single percept (logical sensor) if they are available, and to fall back to a state that does not use fusion if sensors are not available. For example, suppose that human faces can be detected through color detection, but that the results can be improved with a thermal sensor. When both the thermal sensor and camera are available, they should both be used, because the fused sensor readings are more reliable. However, if the thermal sensor is needed for some other task, the face finding percept should be able to fall back to using only the camera (with a corresponding reduction in sensing quality).

2.3 Utility

The Min-Conflict algorithm treats all assignments equally, and breaks up conflicting assignments randomly. Changes were required to respect the utility ("happiness") of different assignments. However, this leads to a far broader topic, that of how sensor quality should be measured. Two methods were used to determine the "happiness" of an assignment.

The first of these methods is to put the programmer in charge and allow them to specify the order in which physical sensors should be considered. The utility is then a function of the index n of the sensor that was finally selected (where n starts at zero for the first sensor), relative to the number of options (referred to as *alternates*). The chosen function was:

$$happiness = \frac{(L-n)}{L} \quad where \quad n \geq 0$$
$$L = |alternates|$$

Table 1. Role of measurements in true positive and true negative rates.

	Reports stimulus	Reports ¬stimulus
Stimulus	True positive	False negative
¬Stimulus	False positive	True negative

True positive + False negative = Number of positive samples
True negative + False positive = Number of negative samples

That is, if a logical sensor can use four physical sensors, then $L = 4$, and if the second is chosen, then $n = 1$. Thus, the utility would be $\frac{4-1}{4} = 0.75$. While this function has its merits, it does not reflect the real performance of the robot's sensors for the tasks required. Since it relies on an *a priori* list of sensors being built (by what may be arbitrary criteria), the resulting utility may not be appropriate. A method developed for this thesis is to base the happiness of a logical sensor on its measured rate of *correct* readings, where *correct* is a function of the true positive and true negative rates. This information was gathered by activating a logical sensor and comparing its readings against the known state of the world. For example, the *face-find* logical sensor was assigned each of its alternate physical sensors and allowed to run while a person stood before the robot's sensors. The results were recorded, and the test was repeated with nobody in front of the robot. This provided four values: *the rates at which the logical sensor correctly identified a person or the lack of a person, the number of false positives, and the number of false negatives.* The relationships between these values are shown in Table 1.

This presents a new problem: finding an appropriate method for combining these values into a single utility value. The frequency of true positive and true negative readings should both be considered, as the following example shows. Suppose that a particular sensor generates a high true positive rate and a low true negative rate for a particular percept. This sensor should be given a lower utility value than one which has high true positive and true negative rates (meaning that it is correct more

often). A simple method of combination is to use a function which would capture the quality of the positive and negative readings while reducing them to a single value.

2.3.1 *t*-norms

There is a class of functions, called t-norms, which map a pair of values from $[0, 1]$ into a single value in $[0, 1]$. Menger [19] first introduced these as *triangular norms*, which later became known as t-norms in the literature of fuzzy set theory [13]. Menger specified five properties for these functions. First, the range of the functions is on the interval $[0, 1]$. Second, the functions are non-decreasing in either of their two parameters (in other words, the functions are monotonic). Third, *triangular-norm* functions are commutative. The remaining two properties are boundary conditions: if both parameters have a value of 1, then the result is 1, and further, if one parameter is 1 and the other is greater than zero, then the result is greater than zero.

Certain aspects of these functions make them appropriate to this problem of combination. First, they are commutative, so the order of the operands is unimportant. Next, they are bounded (due to monotonicity) to be no larger than the smallest operand. This is important for the pathological case, in which a logical sensor's alternate always returns a certain value regardless of the actual state of the world, meaning that either its true positive or true negative rate is zero (such as if *face-find* always reported a face, even when none was present, so its true negative rate would be zero, as it would report no negatives). By bounding the combined value of these operands to be no greater than the smaller operand, the alternate in this case would be given a utility of zero, which is appropriate, since it is returning no useful information.

There are numerous t-norm functions, two of the most common being the algebraic product of the operands, and the minimum of the operands [13]. While the minimum function captures the two important aspects of this class of functions, its result only reflects one of the operands (that which was smaller). Consider the following example.

Suppose that an algorithm was being tested for *face-find* whose true positive rate was 80% and that its true negative rate was 90%. Next, suppose that by using fusion with a thermal sensor, the true negative rate can be improved to 100%, meaning that it is more discriminating. This scenario is shown in the following table:

	True Positive	True Negative	Minimum
Without Fusion	0.8	0.9	0.8
With Fusion	0.8	1.0	0.8

If the utility of this new algorithm was calculated as the minimum of the true positive and true negative values, then this new algorithm would be given the same utility (0.8) with or without fusion, which does not reflect the improvement that adding another sensor provided.

In order to capture information about both the true and false positive rates, the utility for a particular logical sensor given one of its physical sensors for this thesis was defined as follows. The true positive and true negative rates are normalized into the range $[0, 1]$, and the results are combined using the algebraic product. That is, the normalized true positive and true negative rates are multiplied together. In the example given above, the utility of *face-find* without fusion would have been $0.8 \times 0.9 = 0.72$, and in the case with fusion, $0.8 \times 1.0 = 0.8$. Thus, the improvement in adding another sensor is reflected by a higher utility. The algebraic product goes to zero as either of its operands approaches zero, and reaches its maximum value of one only when both operands are one. Further, when only one of the operands is varied, the change in the result is linear. Since this operator is simple to use, easy to verify, and somewhat intuitive, as well as the fact that it uses both operands equally to produce a result, it is an adequate combination function for this problem.

2.4 Flexibility

The flexibility of a logical sensor is determined by the relative number of distinct ways that it can generate a particular percept. When a logical sensor makes a request for sensing, there are different ways in which its request can be satisfied. The different possibilities, or *alternates*, are known at the time of the request, and their number can be counted. The relative flexibility of each logical sensor is used when resolving conflicts; it determines the order in which conflicting logical sensors are searched for new assignments. This directs the search toward more likely solutions: it will be easier to swap the physical sensor attached to a logical sensor that can sense in many different ways than one that depends on a particular physical sensor.

2.5 Least Disturbance

The Min-Conflict heuristic does not make any consideration for existing assignments when it makes repairs. That is, any assignment may be changed in order to test a new state. On a robot, however, there may be costs associated with each assignment, and those costs should be kept to a minimum. The modifications made to Min-Conflict divide its allocations into three stages: assignment, local repair, and global repair. Changes to any existing assignments when a new request is made are biased toward the global repair stage, so that solutions which do not incur any cost are considered first, and those which will only disturb a single assignment are considered next. The final stage may cause significant disturbance (and cost), but it is seldom needed, as is discussed in Chapter 5.

2.6 Behavioral Sensor Fusion

Handling behavioral sensor fusion with the Min-Conflict algorithm was done through the representation of sensors. As Min-Conflict is largely symbolic in nature (it can

Figure 1. Selection process performed to determine the best physical sensor to allocate.

make assignments, but in very general terms), behavioral sensor fusion becomes a simple extension of the algorithm. That is, a set of fused sensors is considered together as though they were a single sensor. The other issues described above still have to be addressed in the code, as well as testing whether all of the sensors in a fused set are available, but the abstraction to handle sensor fusion is straightforward.

2.7 Robotic Constraints

The Min-Conflict with Happiness algorithm (MCH) is part of a larger system for physical sensor selection. There are additional hard constraints that may make otherwise functional sensors unsuitable for assignment, including update rate, power consumption, and viewing angle. These constraints may eliminate physical sensors from consideration before MCH attempts to perform an assignment. When MCH is finally invoked, it is assumed that all remaining physical sensors can potentially be assigned. The role of MCH in this system is illustrated in Figure 1.

The update rate is important when operating in the real world with real constraints. Having a highly accurate vision routine that can identify people by name, for instance, would be a useful logical sensor, but it is useless if it takes so long to complete one iteration of the algorithm that the person being recognized has already left when it finishes. Similarly, if a pair of cameras are being used to generate depth from stereo for obstacle avoidance, this process might not be fast enough to actually prevent the robot from hitting anything, unless the robot were to stop until the algorithm finished. Ideally, a minimum speed should be established for the robot to

move, and if a particular physical sensor could not be used to produce updates fast enough to safely maintain that minimum speed, then the sensor should be removed from consideration.

Field of view refers to the angles over which a sensor can measure. On a Nomad200 robot, for instance, a SICK laser rangefinder could only measure 180 degrees, at high resolution, while the sonar rings could measure all 360 degrees, but with a lower resolution. For measuring distance, the laser produces more accurate readings, but is unusable for obstacle avoidance, because it cannot "see" everything around the robot at once. However, the sonar could be used in place of the laser for detecting things directly in front, just not as well.

Finally, there is the issue of power consumption. While this is not a problem that has been encountered in the robots described here, it might be an issue in other domains. Different sensors may require more power than others (for instance, a structured light system will consume more power than a pair of simple cameras for distance readings). However, a robot may be so limited on power (either stored power or maximum sustained power drain) that certain choices for sensors are inappropriate (given the power remaining to the robot) or impossible (given the maximum power that is available at any moment).

For the cases presented above, physical sensors that violate hard constraints should be removed from consideration before MCH begins its assignments. These constraints are different from the problem of sensor failure (which is discussed in Section 5.3.5). The power available to the robot may change as its batteries discharge, but there may be many combinations of sensors that will consume an acceptable amount of energy, whereas failed sensors cannot be used at any time. The minimum update rate for the robot might be subject to change according to a higher-level deliberative decision, which might eliminate that constraint. Finally, field of view may make a sensor unsuitable, even though the sensor will work for other tasks.

2.8 Robotic Application: AAAI Competitions

Finally, the algorithm must be demonstrable in a real robotic domain. The Min-Conflict with Happiness algorithm was implemented in LISP and incorporated into two different robotic architectures and run on robots with heterogeneous hardware and software. These robots were entered in the 1999 and 2000 AAAI "Hors d'oeuvres, Anyone?" competitions, where they were awarded Third Place and two Technical Achievement awards. As they competed, they used the MCH algorithm to handle their sensing requirements, even when faced with unpredictable hardware failures. Resulting from these and other tests, there exist data that show the performance of the MCH algorithm as suitable for sensor allocation. It is also shown to outperform other techniques, including a *greedy* algorithm that is also the first phase of the MCH algorithm, and a *random* algorithm. The details of these techniques are provided in Chapter 5.2.2.

CHAPTER 3
LITERATURE SEARCH

3.1 Previous Work in Allocation

The domains of scheduling algorithms and resource allocation have been widely studied [7], but primarily in terms of AI applications. Scheduling and allocation for planning have been explored in solving the n-queens and machine shop scheduling problems, among others, using the Min-Conflict with repair heuristic [12]. Planning in real time has been explored through such methods as Real-Time-A* search [15] and its variations [18, 24].

Work more directly related to robotics has been done in resource allocation, most noteworthy are [6, 29]. The method in [6] relied on the economics associated with each plan, such as the estimated reward and completion time. As discussed in Chapter 1, these facts may not be available to the scheduler, and this thesis assumes that they are not. Other work has been done in planning for resource contention [29], though that approach assumed that sensors could be shared between behaviors. The implementation described in this thesis did not make this assumption, but this issue remains open for future work.

Hovland and McCarragher [11] developed a system for dynamically selecting sensors for robots, though their approach was concerned with process monitoring for manipulators. Their approach was to balance the confidence in a particular process monitor (that is, its accuracy) against its execution time (as their robot was running with real-time constraints), using these two attributes to compute a *reward*. Their approach does resemble that used in this thesis: their execution time measure is di-

rectly related to the update rate, which is used in this thesis to eliminate possible assignments. Further, the reward for using each process monitor resembles the utility ("happiness") used in this thesis, as both are used to sort the possible choices. However, their approach is shown only in terms of a single percept that is required and the different ways (combinations of sensors and algorithms) in which it can be generated in real-time. Their approach does not address the issues of contention (whether a particular sensor is available), since only one percept is required at any time. Further, complications such as sensor failure, or features such as sensor fusion, are not addressed in their work.

Sensing and planning have been considered for robotics architectures, but those efforts have concentrated on either generating plans from scratch or recovering from problems. Approaches to plan about sensing through a deliberative process have been explored [3], including the advantageous use of new information. Architectures that detect or compensate for problems such as sensing failures have also been proposed [23, 22]. The architecture in [23] detects failures by monitoring the motor behavior from the deliberative layer. The SFX architecture [22] keeps the tasks of monitoring and exception handling at the behavioral level. Lopez-Orozco, de la Cruz, Domínguez, Besada, and Polo [17] describe a sensing architecture that allows for sensor fusion and tolerates sensor failure. However, their work does not discuss the possibility of conflict between simultaneous sensing needs.

Dekhil, Sobh, and Efros [4] introduced *commanding sensors*, which are a combination of raw sensor inputs and algorithms for extracting percepts, such that the commanding sensor can directly issue a motor command. In their approach, sensors are hard-coded to particular percepts, thus circumventing the allocation problem. Action selection (choosing between the different commands being issued by the sensors) is done based on a simple priority measure, where the priorities are assigned by a programmer in advance.

Celinski and McCarragher [2] describe a *Sensory Perception Controller* that is similar to the Sensing Manager in the SFX Architecture, which is a target for this thesis. Their approach is concerned with determining what physical sensors can most cheaply generate a particular percept, but does not address the issue of contention. If all sensors that can generate a percept are in use, then the system fails to assign a sensor. In the approach presented in this thesis, repairs are made that may provide solutions that their approach would miss. However, their work could be useful in generating the sets of physical sensors that each logical sensor could use.

3.2 Previous Work in Sensor Utility

Research has been done that deals with the utility of a sensor for a task. Zheng [30] used logical sensors in the area of industrial manipulation, and explored the impact of sensing on various tasks. This work recognized the need for maintaining a minimum update rate, and assigned a cost function to the response time of the logical sensor and to the uncertainty associated with that sensor (which reflects the desire to keep the accuracy of sensing high). In this work, the impact of sensing was applied to an *intelligence* measure, which reflected how much "smarter" the robot became through use of a given sensor. These intelligence scores were enumerated over seven levels.

Lindner, Murphy, and Nitz [16] developed a system for exploring the utility of a robot's sensors in an unknown environment, which used the expected utility and cost of a particular sensor when selecting the minimal set of sensors needed to adequately perceive the world, favoring less costly sensors. These methods differ from the approach used here, as the cost of a particular assignment is not considered, since this measure is difficult to acquire for the general case. Instead, as will be described, any sensor which is not eliminated due to hard constraints may be used, with preference going to the sensor or algorithm that produces the most accurate results.

Xu and Vandorpe [28] also addressed the problem of sensor fusion and sensing quality, including the detection of an errant sensor through inconsistent readings, and modifying the belief in a sensor reading by propagating the recent performance of the sensor forward. This work is also concerned with using the least sensing data that are needed to perform a task. However, the task at hand seemed to be limited to navigation and localization, and uses measurable error in order to determine the utility of its sensors. Further, while different utility functions were referred to, no optimal function was provided for the general case.

Rosenblatt [25] discussed utility for mobile robots in terms of *utility fusion*, using an arbiter in the DAMN architecture to perform action selection to maximize utility. In that application, behaviors provide the arbiter with possible trajectories (that is, steering commands) with expected utilities. The arbiter then chooses the trajectory that will produce the highest expected utility. However, this approach uses subjective utility values for each outcome, and models any uncertainty using a Gaussian distribution, which may not describe real sensor performance.

Kobayashi, Arai, and Fukuda [14] use a possibility measure to quantify sensor reliability, though their method seems biased toward verifiable range sensor data. They also contribute a system for selecting sensors based on a set of rules that are determined in advance by a human operator. The possibility measures are used as preconditions for rules that choose what sensors to use. They then used a recurrent neural network to fuse sensor data. The approach presented in this thesis determines for itself what sensors can be used, rather than deferring this responsibility to a human operator.

CHAPTER 4

ALGORITHM

This chapter describes the components of the Min-Conflict with Happiness algorithm (MCH). The assumptions that allow the algorithm to function and the context in which MCH would be used are revealed and explained. Pseudocode for the algorithm is provided, as well as a verbal description of what the code does and why. The algorithm is split into its primary components, and each of these is explained in turn, with examples. Extensions to MCH are then presented, including support for opportunistic repair, behavioral sensor fusion, and sensor failure.

4.1 Notation and Assumptions

The Min-Conflict with Happiness algorithm has three primary components, one which makes assignments, and two which repair conflicts that result from those assignments. The first stage is *Greedy Allocation*, which is followed, if necessary, by *Local Repair*, which in turn is followed by *Global Repair* if necessary. A variant on the algorithm, MCH+, adds an opportunistic repair stage at the end (after *Global Repair*). These stages are illustrated in Figure 2, along with the relative frequency that each stage performs a successful assignment based on simulation data.

This chapter will describe the algorithm in terms of physical sensors $(p_1, \ldots, p_n)$, logical sensors $(l_1, \ldots, l_n)$, and the various physical sensors that each logical sensor can use, which will be referred to as alternates (each l_i has a list a_1 through a_n). Examples in terms of real hardware will also be used as necessary. Pseudocode for the algorithm can be found in Figures 3–6.

Figure 2. Stages of MCH Algorithm. Values shown are from simulation. To the right is the fraction of cases that each stage successfully solves; assignments that *Greedy Assignment* cannot perform are attempted by *Local Repair*, and so on. To the left is the gain provided by MCH+, discussed below.

This algorithm assumes that the state of the robot's sensors is known. This implies that there is a list of physical sensors available, which may include whether a particular sensor is functioning. It also assumes that each logical sensor that may be activated is associated with a list of physical sensors that it can use, and that these "alternates" are either sorted according to their utility, or that their utility is specified (a discussion of utility will follow). Finally, while the description of the algorithm suggests that many changes will be made to the sensing state while searching for a solution, the algorithm is intended to start with the current sensing state and assert a new sensing state when it is done. The intermediate changes do not need to be enacted; only the differences between the starting state and the ending state should be enforced.

These assumptions require that some representation of the physical sensors and logical sensors be established. Physical sensors are relatively easy to represent for this problem; they have a name and a set (or list) of logical sensors to which they have been assigned. Logical sensors are more sophisticated. These too have names and

sets of physical sensors that have been assigned to them, but each also has a variable number of alternate sensors that it could use. Associated with these is a utility which directs assignments, as well as any number of other criteria that can remove possible alternates from consideration (such as the update rate of the logical sensor with a particular alternate). A suggested implementation of this algorithm is in Common LISP, such that logical sensors are structures that contain a list for alternates and a list for the physical sensor or sensors currently being used. Each alternate can be represented with a structure as well, naming the physical sensor or set of fused sensors required, a utility value for the alternate, and other attributes, such as update rate. A hash table can be built using the names of all physical sensors on the robot, and each physical sensor can be associated with a logical sensor structure when assignments are made.

The process described by the stages below (and outlined in Figure 7) constitutes a search through possible assignments that would provide a logical sensor with a physical sensor upon request. There are some characteristics to this search that make this approach favorable. First, modern mobile robots tend to have a small number of sensing modalities. That is, the number of physical sensors to choose from is expected to be limited. Next, the solutions that cause the *least disturbance* should be considered first, as discussed in Chapter 2.2. Third, as requests are to be satisfied immediately, the search should produce an answer quickly, favoring simpler techniques to solve a majority of requests, and resorting to more exhaustive means only when no other option exists. Finally, this search is done without any information about future requests, or knowledge regarding how long a particular assignment will be needed. This final point disallows techniques that rely on prediction.

There is also the issue of utility, which for the case of the algorithm description, simply dictates the order in which choices are considered. A more complete discussion of this topic appears in Chapters 2 and 5.

Given a list of physical sensors, $P = \{p_1, p_2, \ldots, p_n\}$ and a list of logical sensors, $L = \{l_1, l_2, \ldots, l_m\}$ where each logical sensor maintains a list $A = \{a_1, \ldots, a_k\}$ of alternate sensors in the order it prefers to use them, the algorithm is:
When logical sensor l_i becomes active,

```
    Procedure activateLogical(l_i)
        # First assign with Min-Conflict
        minAssign ← ∞
        bestSensor ← NIL
        for j := 1 to k do
            if numAssigned(a_j) < minAssign then
                minAssign ← numAssigned(a_j)
                bestSensor ← j
        addLogicalToSensor(l_i, bestSensor)
        # Now repair the assignment
        for c := 1 to n do
            if conflictExists(p_c) then
                localRepair(p_c)
        for c := 1 to n do
            if conflictExists(p_c) then
                globalRepair(p_c, NIL)
```

Figure 3. Min-Conflict with Happiness Algorithm

```
    Procedure localRepair(p_c)
    # Displace logical sensors to other physical sensors to reduce local conflict
        flexList ← list of logical sensors assigned to p_c
            in the order of how many alternative sensors each
            logical sensor can use, from most to least
        for each l_i in flexList while conflictExists(p_c)
            removeLogicalFromSensor(l_i, p_c)
            success ← assignToBestFreeSensor(l_i)
            if success = FALSE then
                restoreLogicalToSensor(l_i, p_c)

    Procedure assignToBestFreeSensor(l_i)
    # Assign logical sensor to best unused physical sensor in list A
        for j := 1 to k do
            if numAssigned(a_j) = 0 then
                addLogicalToSensor(l_i, a_j)
                return TRUE
        return FALSE
```

Figure 4. MCH Algorithm, continued

```
Procedure globalRepair(p_c)
# Assumes that all possible local repairs have been made
    flexList ← list as in localRepair
    Start list takenSensors with p_c
    solutionPath ← findPath(flexList, p_c, takenSensors)
    if solutionPath != NIL then
        applySolutionPath(solutionPath)
```

Figure 5. MCH Algorithm, continued

```
Procedure findPath(flexList, p_c, takenSensors)
# Recursive procedure to find what changes must take place
# on other sensors to make room for a logical sensor from the current
# sensor.
   if flexList = NIL then
      # This path is a dead-end
      return NIL
   repairList ← NIL
   candidate ← first of flexList
   sensorList ← list A from candidate
   possibleSensors ← setDifference(sensorList, takenSensors)
   currentSensor ← first(possibleSensors)
   if possibleSensors = NIL OR conflictExists(first(possibleSensors)) then
      # Go on to the next logical sensor on this physical sensor
      return findPath(rest(flexList), first(possibleSensors), takenSensors)
   if numAssign(currentSensor) = 0 then
      # Logical sensor can be moved to this sensor
      Append candidate and currentSensor
      to repairList and return repairList
   else
      # Test if logical sensor on currentSensor may be
      # moved to make space (recursively)
      Add currentSensor to takenSensors
      repairList = findPath(logicalUsingSensor(currentSensor),
         currentSensor, takenSensors)
      # If that found a solution, use it
      if repairList != NIL
         Add logicalUsingSensor(currentSensor) and currentSensor
         to repairList and return repairList
      # Otherwise, continue (note that currentSensor has been
      # added to takenSensors, so the recursive step is a smaller case)
      else
         return findPath(flexList, first(possibleSensors), takenSensors)
```

Figure 6. Algorithm, continued

Figure 7. Overview of the MCH assignment process, in terms of the individual stages and representation used.

4.2 Greedy Allocation

This stage is the first step in the allocation of a physical sensor to a logical sensor, and begins when a request is made on behalf of a logical sensor. The list of alternates for the logical sensor (that is, all of the physical sensors that it could use) are sorted by their utility, and hard constraints are applied (meaning that if a sensor is broken or otherwise unsuitable, it is removed from consideration). The remaining alternates are then checked in order for availability. The logical sensor is provided with the first physical sensor (or set of fused sensors) available. If none of the alternates were available, then the logical sensor is assigned the alternate that has the fewest prior assignments to other logical sensors (the minimum number of conflicts). If there is a tie, the alternate that provides higher utility is assigned.

Greedy Allocation is an effective method for making assignments quickly and without disturbing previous assignments. Simulation has shown that Greedy Allocation handles a vast majority of allocation requests (experiments described later suggest

greater than 85%) without generating conflicts. However, this simple approach will fail if none of the alternates for a logical sensor are available. The next two stages repair any conflicts left after Greedy Allocation is finished.

4.3 Local Repair

This stage takes its name from the fact that any changes (repairs) are made only to the way a single physical sensor has been assigned. That is, the repairs are local to a single physical sensor. It is assumed that this stage is always performed after Greedy Allocation has finished, and that conflicting assignments for some physical sensor exist.

For each physical sensor p whose assignment needs repair, a list of the logical sensors it is assigned to $(l_1, \ldots, l_m)$ is generated. This list is then sorted according to the *flexibility* of each logical sensor in the list. *flexibility* is simply the number of alternates that each logical sensor can use. The purpose of sorting the list in this manner is that it biases the next part of the repair to check possible reassignments for logical sensors that have many alternates over those that have few. Given a uniform distribution of previous assignments, there is a higher probability that a logical sensor with many alternates will have (at least) one that is free than a logical sensor with few or no alternates. This ordering also prevents checks for alternates for logical sensors that can only use one physical sensor (which is, by definition, a dead-end).

Next, for each logical sensor l_i in the sorted list, as long as a conflict still exists on the physical sensor p for which the list was created, the logical sensor l_i is tested. This test consists of unassigning l_i from p, and using Greedy Allocation to find a free alternate. If Greedy Allocation is successful, then one conflict on p has been repaired; the process will continue on the next l_i while conflicts still exist. If the Greedy Allocation fails to find any free alternates for l_i, then l_i is assigned back to p (rather than generating new conflicts elsewhere).

Figure 8. Case where Global Repair is needed.

Local Repair can fix many of the cases where Greedy Allocation fails, but it does so by disturbing existing assignments. However, this disturbance is limited to the logical sensors assigned in conflict to a single physical sensor. In simulation, this stage was the final step needed in a successful allocation less than 10% of the time (as Greedy Allocation handled the first 85%). This stage may fail, however, if none of the logical sensors l_i assigned to p have free alternates, though even in this case, a solution may exist that would satisfy all active logical sensors. For this reason, the final stage is needed to do a more far-reaching search.

4.4 Global Repair

The final stage of repair is by far the most complex, time-consuming, and powerful of any stage. It is also required the least. In simulation, this stage was called upon to satisfy 1.9% of the allocation requests, and took 18 milliseconds of processor time on average (on the robot's Pentium 233). It builds on the procedure from Local Repair, but does so in a recursive manner. As this is a recursive procedure, it will be described in terms of its base cases and inductive steps. It is assumed that this procedure is called only after Greedy Allocation and Local Repair have failed to resolve conflicts.

Suppose that logical sensors l_a and l_b have been assigned in conflict to p_x. l_a is only capable of using p_x, but l_b can use p_x or p_y. However, Greedy Allocation and Local Repair will both fail to make an assignment without conflict if p_y is already in use. Finally, suppose that l_c is using p_y, and is capable of using p_z as well, which

Figure 9. Solution after Global Repair.

happens to be free. This is illustrated in Figure 8. Given this scenario, although the first two stages have failed to make a legal assignment, a solution still exists: l_c should be moved to p_z, which frees up p_y for l_b, leaving l_a on p_x (shown in Figure 9). This scenario is chosen carefully for example purposes, but the actual solution to the problem could be much more complex. It is the goal of the Global Repair stage to find that solution, if one exists. This solution will be a sequence of changes that must be made in order to eliminate conflicts.

This stage begins by choosing a physical sensor p_a with conflicting assignments after the Greedy and Local Repair stages have finished. It also creates a list of physical sensors that have been tested already (initially empty) and adds p_a to it. This will prevent the search from running in cycles. Given p_a, the logical sensors using p_a are gathered into a list, and the set of sensors that each can use is built into a list of alternates. Upon success, this stage will produce a solution in the form of an ordered set of pairs, containing logical sensors and the physical sensors to which they should be assigned to reach a solution. At each level of the recursion, reaching a base case implies creating the first step in this solution path, and as the recursion returns, each calling instance adds its own current logical and physical sensors as pairs. This solution path can then be enacted. If there is no solution, the solution path will be null.

There are two base cases to this stage: either some physical sensor p_b is found that is not in use (which solves one conflict), or no untested alternates remain. With

this in mind, there are three recursive calls, one for each of the following scenarios. If there are no untested alternates available, or if the physical sensor is in conflict already, then the search is called recursively on the remaining logical sensors on p_a (starting with the first alternate of the next local sensor) and without modifying the list of tested sensors. Otherwise, there were no conflicts on the physical sensor under consideration, meaning that p_a has no conflicts because it is unused, or p_a has no conflicts because the assignment(s) on it do not constitute a conflict.

If there was no assignment to p_a, then a base case has been reached. Otherwise, if there was an assignment, the search recurses, checking the next alternate p_b. The result of this recursion is stored; if it leads to a solution, then the solution is returned. If it did not work, then the search recurses on the next alternate physical sensor for the current logical sensor being tested.

4.5 Examples

This section will provide examples of the mechanics of MCH so that the algorithm is made more clear. These examples are for illustration only and may be skipped if desired.

Butler is a Nomad200 robot, and will be the basis for the following examples as it has many different sensors. Butler is equipped with two color cameras on a pan-tilt unit, which allows the cameras to be aimed over more than 180 degrees by panning. Butler has two rings of sonars that fire in unison, each with 16 sonars. Each sonar covers an angle of 22.5 degrees. Butler also has a SICK planar laser rangefinder, which provides 180 degree coverage in front of the robot, with 0.5 degree resolution. Next are two bumper rings on the base of the robot, each of which has 16 contact switches. Optionally, a thermal probe can also be added through a serial connection.

4.5.1 Greedy Allocation and Local Repair

Suppose that Butler's first task is to navigate down a hallway while looking for a colored sign next to a door. In this case, the logical sensor being used to follow the hallway (*hall-follow*) has been written to use vision, following the baseboards of the hallway, or to use sonars and to keep the robot centered between the walls. However, vision is preferred because the walls are specular and can fool the sonars. Vision happens to be the only sensor that the sign finding logical sensor (*sign-find*) can use, so there will be contention.

Given this task, and assuming that nothing else is already using the cameras or sonar, a behavior requests a sensor suitable for hall-following. Since vision is preferred for *hall-follow* and nothing else is in use, *Greedy Allocation* assigns the cameras to the *hall-follow* logical sensor.

Shortly thereafter, *sign-find* is requested. Since this logical sensor can only use cameras, and because the cameras have already been assigned to *hall-follow*, *Greedy Allocation* fails, and leaves both the *sign-find* and the *follow-hall* logical sensors assigned to vision. At this point, *Local Repair* begins, and finds that two logical sensors have been assigned in conflict. *Local Repair* generates a list of the conflicting logical sensors and sorts it by their flexibility. Because *hall-follow* can use two sensors and *sign-find* can only use one, *hall-follow* is picked first. The physical sensor that *hall-follow* is using is noted, and *hall-follow* is removed from vision and restarted using *Greedy Allocation*. *Greedy Allocation* checks the list of physical sensors that *hall-follow* can use, finds that its first choice (vision) is already taken, but that its second choice (sonars) is not. *hall-follow* is assigned to sonars, and no conflicts remain, so the repair ends successfully.

If the sonars had not been available, *hall-follow* would have been restored to vision, and (given no other logical sensors using vision to test), a *Global Repair* call would be attempted.

4.5.2 Global Repair

Another example task for Butler is to move through a doorway. Suppose that there are three logical sensors that may be needed: *find-door*, which looks for a range profile consistent with a doorway using either laser or sonar; *avoid-obstacle*, which keeps the robot from running itself into the wall and door frame using sonar or if necessary its bumpers, and a high-resolution logical sensor for mapping (*map-room*), which can only use the laser.

Suppose that Butler begins by running *avoid-obstacle*, which through *Greedy Allocation* is assigned to sonar, and *find-door*, which is similarly provided with the laser. At this point, the robot is able to begin its task of moving through a doorway. However, as the robot is completing this task, a cartographer behavior becomes active to map what is an otherwise unexplored room, and requests its only viable sensor, the laser.

At this point, *Greedy Allocation* fails, because the only sensor that *map-room* can use is not available. At the end of the first stage, the laser is assigned in conflict to both *map-room* and *find-door*. *Local Repair* also fails, as follows. *Local Repair* attempts to find a free alternate for one of the logical sensors using the laser. *find-door* is checked first as it is most flexible, but its only alternate sensor (sonar) is already taken by *avoid-obstacle*. *map-room* has no alternates, so there is no solution there, either.

In order to reach the solution, *Global Repair* is needed. When this routine begins, it finds that the laser is being used by both *map-room* and *find-door* in conflict. Starting with the most flexible logical sensor, *find-door*, *Global Repair* notes that the laser has been searched (so it will not search it again), and looks to *find-door*'s alternates, which leads to the sonar. It then finds the sonar being used by *avoid-obstacle*, and checks *avoid-obstacle*'s alternates, also noting that the sonar have been searched. This leads it to the bumper, which it finds is unused.

From here, the recursion begins to return with the solution. *Global Repair* first pairs the bumper with the logical sensor that had led it there (*avoid-obstacle*), meaning that those two should be together. It then pairs the sonar with *find-door* and prepends these before the other pair. *map-room* is left alone, because it does not need to move. The list of pairs is then used to guide the reassignment of logical sensors to their new physical sensors, and the system is left in an acceptable, conflict-free state.

4.6 Opportunistic Repair: MCH+

As it is written, the MCH algorithm focuses on making assignments only when physical sensors are requested. However, physical sensors that are freed are not automatically reassigned. On one hand, this approach follows the principle of *least disturbance*, as existing assignments are not broken to make use of newly available sensors. On the other hand, this means that a higher overall utility may be possible if unassigned sensors are used.

Consider this example. Suppose that a *follow-hall* behavior is active, and its rangefinding logical sensor can use either sonar or laser, and that the laser provides better performance. Initially, the laser is assigned to this behavior's logical sensor, but after a long interval, a *find-door* behavior requests the laser and gets it, forcing *follow-hall* onto the sonar. If *find-door* releases the laser, it is not automatically given back to *follow-hall*; in fact, it will remain unused until explicitly requested. If *follow-hall* never requests any change to its sensing needs, it will always be stuck with the sonar, even if a better sensor (the laser) is available.

A variant on MCH, called MCH+, allows for optimization between requests. MCH+ adds an *Opportunistic Repair* stage, which allows every logical sensor to test whether a better physical sensor than the one it is using is available. This is done by recording the sensing state of the logical sensor, then attempting to free and request that logical sensor's resources through the MCH allocation system. If doing so results

in a higher utility, then the change is enacted; if not, the old state is restored. This process can be done to all of the logical sensors in order of increasing "happiness", so that those with the least utility have the first chance at improving their sensing state.

One problem with opportunistic repair is that it may disturb existing assignments, and if done too frequently, may lead to *thrashing* (that is, more time may be spent reassigning than actually sensing). Unfortunately, it is difficult to quantify what "too frequently" means. A suggested method is to perform opportunistic repair after a certain period of inactivity. The length of this period can either be assigned arbitrarily, or based on measured activity.

4.7 Sensor Failure

A useful addition to the MCH algorithm is the capability to respond to sensor failure. If a particular physical sensor becomes inoperative, MCH should provide a functioning alternative if one exists.

The representation of a failed sensor can be handled by adding a special logical sensor (in this case, called *BROKEN*) to act as a placeholder for a physical sensor. When a sensor fails, a function is called which assigns *BROKEN* to the sensor, and all of the previous assignments to that sensor are reassigned (using the same mechanism to start and stop logical sensors that would normally be used). Once *BROKEN* is assigned, any other assignments to that sensor are seen as being in conflict, and since *BROKEN* can never be relocated, it keeps that sensor from being reused. This is simple to reverse as well; if a failed sensor begins working again, the *BROKEN* assignment is removed, leaving that sensor open for immediate reassignment.

4.8 Behavioral Sensor Fusion

This section describes the changes needed in order to support Behavioral Sensor Fusion in the MCH algorithm. This feature allows for more reliable sensor readings to be gathered, and can provide a higher degree of sensor utilization.

Behavioral Sensor Fusion is the integration of sensor data from different sensor modalities at a symbolic level, well above the signal level. Three types of Behavioral Sensor Fusion have been identified by [1]. In *sensor fission*, concurrent behaviors with distinct sensors generate divergent results, which are then combined by some mechanism. *Action-oriented* sensor fusion combines the inputs from different sensors into a single percept for a behavior. This integration can be very task-specific, which suits it to specific actions. The final type of behavioral sensor fusion is *sensor fashion*, which deals with the sequence of sensing as it relates to solving a problem. Of these, *sensor fission* and *sensor fashion* are the most common. By combining multiple sensor readings together, the accuracy of a particular percept can be improved. Further, the capability of using many sensors together for a single percept suggests that sensors may be utilized when they might otherwise be idle.

As the MCH algorithm is described above, it only allows one physical sensor to be assigned to any given logical sensor at a time. However, sensor fusion implies that a logical sensor may employ multiple physical sensors simultaneously. Certain changes must thus be made to MCH for it to support Behavioral Sensor Fusion. The significant changes that must be made deal with representation; the underlying assignment techniques are unchanged, but additional steps must be taken for *fused* sensors, meaning those sets of physical sensors that may be assigned together to a single logical sensor. What follows is a description of the changes to the MCH algorithm that are needed for behavioral sensor fusion to work.

The function *Assign* in the pseudocode, which is not explicitly defined, takes two parameters, l (a logical sensor), and *BestSensor(l)*. For the simple case where there

is no fusion (such as all of the examples given above), BestSensor(l) refers to a single physical sensor p, and the *Assign* function would update whatever data structures were needed to represent the assignment of p to l. These details are implementation specific, and have therefore been left vague.

To handle fusion, a more abstract interpretation of p is needed. In addition to p representing a single physical sensor, it can also represent a set of physical sensors that work together. This new interpretation of p immediately allows groups of sensors to be considered as a single fused sensor, and allows fused sensors and unfused sensors to be treated equally when selecting among them. Minor modifications must then be made to behave properly with fused sensors.

To this end, *BestSensor* is extended to return either single physical sensors or sets of sensors, and *Assign* is modified to repeat the allocation task on all elements of p if p is a list. In other words, if p represents a set of fused sensors, the *Assign* function must make assignments for every physical sensor in p, where before, *Assign* would make only one assignment.

The *LocalRepair* function, when treated with the same abstraction, does not need to be otherwise changed. The process of removing logical sensors from physical sensors and moving them to free alternates is still correct even if the physical sensors are actually groups of sensors, and the alternates may be groups of sensors as well. *LocalRepair* effectively stops and restarts each logical sensor on a contested physical sensor by the same mechanism that *GreedyAssign* did in the first place, so the handling of sensor fusion is performed in much the same way.

The final stage of repair, *GlobalRepair* (and its helper function, *findPath*), also require modifications to allow for cases of sensor fusion, but these changes are small, as *GlobalRepair* and *findPath* build on *LocalRepair* and inherit its changes. The only difficulty with this stage is that the *currentSensor* variable may refer to a set of fused sensors, and each of them must be checked for availability.

Unfortunately, there is a problem when extending this function to support Behavioral Sensor Fusion that may cause it to skip solutions. As it is written, the *findPath* function avoids cycles by only considering each physical sensor once. In the second recursive call in *findPath*, the current physical sensor being considered is added to the list of taken sensors, such that the sensor will not be considered again. If interpreted literally, this means that when a fused set of sensors reaches that call because of a single physical sensor, then the entire set of fused sensors is removed from consideration. The correct interpretation is that only the physical sensor that is causing the recursion should be added to this list of taken sensors, not the entire set it is fused with.

When these changes are made, it becomes possible for logical sensors to request groups of physical sensors together as though they were one sensor, and the allocation and repair steps remain largely unaffected. Through these modifications, support for Behavioral Sensor Fusion is gained, with no loss in functionality. Further, sets of fused sensors can coexist with ordinary sensors, and both are handled equally.

4.9 Summary

This chapter provided a description of the Min-Conflict with Happiness algorithm. The assumptions upon which the algorithm was designed were presented. Pseudocode was provided, accompanied by a verbal description of the steps taken throughout each of the stages of execution, including examples. Extensions to the basic MCH algorithm were then discussed, including opportunistic repair (MCH+), behavioral sensor fusion, and sensor failure.

CHAPTER 5

IMPLEMENTATION AND EXPERIMENTS

This chapter describes how the MCH algorithm was implemented and validated, including the 63% and 142% improvements over other methods tested (detailed in Section 5.2.2) in terms of successful assignments. The algorithm was implemented both in simulation and on a pair of mobile robots, and data from experiments in both cases are provided. A discussion of the advantages of the MCH algorithm concludes this chapter.

5.1 Introduction

Some testing of the algorithm was done at AAAI robot competitions, which presented a challenging perceptual environment, leading to frequent changes in the sensing needs of the robots. These contests are an important step in the validation of the algorithm, as they require that MCH work in a system together with other software, as well as working on-line and in a dynamic situation. Further discussion of the AAAI competitions will follow.

The algorithm performed favorably in simulation and on mobile robots. Simulation showed that MCH handled 63% more requests than a *greedy* algorithm, and 142% more requests than a *random* approach (both algorithms are described in Section 5.2.2).[1] The utility ("happiness") of assignments throughout those tests was also marginally higher with MCH+ (MCH using opportunistic repair), averaging 24.63% better utility than the random approach. On robots, the algorithm allowed for dy-

[1]These values differ from those cited in [8] and [9] due to a tallying error.

namic changes in the sensing state throughout two competitions, performing each change in under 20 milliseconds on a Pentium 233, as the robot's activity generated requests at an average rate of 106.5 changes per hour. Tests have also been done with behavioral sensor fusion that show an improvement of 27.5% and 75% in the rate of true negative readings from fused sensors versus the case where there is no fusion, indicating that support for sensor fusion in MCH was necessary.

MCH has been used in three separate software systems. Its first incarnation was in simulation for the purpose of testing. Next, MCH was joined with a prototype of an object-oriented implementation of the SFX architecture, where it was coupled with the existing *Sensing Manager* program. The resulting system was tested on a pair of heterogeneous Nomad200 robots, which indicates that it is not reliant on one particular hardware configuration. The same robots were used for the final integration of MCH into a simplified framework designed to make up for the weaknesses of the former (most notably speed). Given its current interface, the MCH code can be adapted to work with most software environments.

The testing of the algorithm in simulation was necessary to ensure that it worked properly and to compare it to other allocation methods. The simulation environment allowed for cases to be explored that might not be possible on a real robot, such as dealing with sensors that are not available or simulating interactions between behaviors that do not yet exist. Testing on real robots was also necessary in order to show that the algorithm worked in the real world, and that it could be used with different hardware and software configurations. These tests are sufficient because they demonstrate the effectiveness of the algorithm in complex situations, while also demonstrating its practical use.

5.2 Simulation Results

The first tests of MCH were performed in simulation (and the details of the simulation are provided in Section 5.2.2). The purpose of these tests was to determine how MCH performed compared to other techniques. In order to make this comparison, two metrics were chosen: *how many assignments the algorithm could make before failing*, and *the utility of the assignments that it made*. Measuring these characteristics in simulation was preferable to measuring them on a real robot, because the overhead in implementing all of the algorithms to be compared on a robot and the time needed to run a robot through enough tasks to adequately test the algorithms was not practical. Further, by doing the tests in simulation, it was possible to test all of the assignment techniques using the same input data, whereas the activity generated by a real robot might not be consistent due to real-world interactions. Simulation also allowed the time required for MCH to perform an assignment to be measured, though this was not compared against other methods. In these simulations, MCH performed 63% more assignments than the *greedy* approach and 142% more than *random*, and the utility of MCH+ averaged 24.63% better than *random*. The *greedy* and *random* approaches are described in Section 5.2.2.

5.2.1 Simulator Implementation

The implementation of the MCH algorithm was guided by two principles. First, the implementation itself should be portable, not only across operating systems and computer architectures, but across robot architectures as well. This implies that the implementation must be separate from the specific details of any particular robot, including its sensor suite and the mechanism for communication between parts of the architecture. Next, the implementation should be as simple as possible.

The Min-Conflict with Happiness algorithm was implemented in Common LISP in order to do simulations, and was extended to handle real assignments and later,

Table 2. Parameters for first simulation.

Logical Sensor	Physical sensors
avoid-noise	microphone
avoid-obstacle	sonar, bumper
follow-hall	vision, sonar, laser
follow-worker	vision, microphone, heat-sensor
frontier-search	GPS, vision, sonar, bumper
localize-map	GPS, compass, sonar
locate-green	vision
locate-survivor	vision, heat-sensor, microphone
move-thru-door	vision, sonar
wander	sonar

sensor fusion and sensor failures. LISP was chosen as the programming language since the allocation problem is largely symbolic and contains recursive steps (especially the *Global Repair* stage). To function with a robotic architecture, a number of functions were written to handle socket communication (through the acl-socket package of Allegro Common Lisp) with UNIX processes. Running under LISP, this code communicates with a process written in C which acted as a server, handling multiple simultaneous connections to MCH. Beyond this state, the MCH code becomes independent of any particular platform, and can function anywhere that UNIX sockets can be used.

5.2.2 Experimental Setup

The purpose of the algorithm is to satisfy requests for physical sensors by logical sensors, so in order to test the algorithm, a set of (fictional) heterogeneous logical sensors was fabricated (shown in Table 2). As LISP is symbolic, the actual names of the logical sensors were unimportant; the list of physical sensors that each could use was paramount. The set of physical sensors for each logical sensor was unique, such that the allocation problem was non-trivial, and the total number of physical sensors was less than the number of logical sensors, which ensured contention as well as impossible requests (if no physical sensors remained when a request was made).

Eleven logical sensors were chosen, their names and acceptable physical sensors based on what a robot might be expected to do. In this case, the names were based on possible behaviors, though the effect is the same as if they were named after percepts. A total of eight physical sensors were available for use. The logical sensors and their preference orderings of physical sensors can be seen in Table 2. Under these conditions, each logical sensor could be given one physical sensor at a time, and physical sensors could not be shared.

The simulation experiment was intended to test the effectiveness of MCH in finding satisfactory assignments given an unpredictable sequence of requests. For comparison, two other techniques (*random* and *greedy*) were tested as well. Given a request, the *random* algorithm would choose one of the physical sensors that could satisfy that request at random, but would fail if the sensor had already been assigned. The *greedy* method was effectively equivalent to the first stage of MCH; this technique was chosen because it could illustrate the need for the added complexity of MCH, and because it is generally an effective strategy. *greedy* would assign the best available physical sensor to satisfy a logical sensor's request, but would fail if no alternates for that logical sensor remained.

The experiment consisted of ten random sequences of 20 events. Each event could be either a new request for sensing or the release of a previously assigned physical sensor. The sequence of events was chosen at random by a C program, whose *rand()* function is approximately uniformly distributed. Each allocation method was provided the same sequences, and for each, the point at which the algorithm failed was noted. The total utility across all logical sensors (termed the "global happiness") was also noted for each method until a request failed.

Table 3. Number of assignments completed in simulation by each algorithm. Maximum possible is 200, 20 for each of 10 runs.

Method	1	2	3	4	5	6	7	8	9	10	Total
MCH	16	20	20	7	9	20	17	12	20	19	160
greedy	3	18	4	7	8	3	13	4	19	19	98
random	4	9	10	5	8	3	7	9	4	7	66

5.2.3 Length of Successful Sequences

The result of this experiment shows that MCH was able to satisfy a significantly greater number of requests than the other methods before failing. Compared to the *greedy* allocation, MCH handled 63% more requests on average (with a statistical significance of $p = 0.0317$), and compared to the *random* allocation, MCH handled 142% more requests (with a statistical significance of $p = 4.08 \times 10^{-5}$). The actual number of requests satisfied in each case are shown in Table 3.

The data are also shown graphically in Figure 10, which can be read as follows. The horizontal line in each box represents the mean number of requests satisfied. The box itself represents the inner-quartile range, meaning that half of the data points fall within the range covered by the box. The maximum and minimum values are represented by the thin lines that protrude from the ends of the box. A long box means that the data were distributed across a large range, while a short box means that the data were closely grouped. The plot in Figure 10 thus shows that the average for MCH/MCH+ is much higher than both *random* and *greedy*. Further, it shows that the number of requests satisfied by *greedy* varies widely across the sequences.

5.2.4 Utility performance

In all cases except for one, MCH matched or outperformed *greedy* and *random* assignment at every step. In the exceptional case, the random approach managed a slightly higher utility for a single assignment by making a suboptimal assignment previously. The logical sensor for *Follow-Hall* could either use vision (utility 1), sonar (utility

Figure 10. Box-whisker plot of number of requests handled before failure by each algorithm.

Table 4. Happiness of all algorithms for one typical trial. A − symbol indicates when the heuristic failed. Higher values represent better performance. *greedy* algorithm drops out first, *random* lasts longer. MCH is only outperformed when checks are made for improvements between requests with MCH+.

Request	1	2	3	4	5	6	7	8	9	10	11	12	13
MCH+	1	2	1	2	2	2.66	2.66	3.66	2.66	3.16	2.66	3.66	-
MCH	1	2	1	2	2	2.66	2.16	3.16	2.66	3.16	2.66	3.66	-
random	0.5	1.5	0.5	1	2	2.33	1.83	2.33	1.83	-	-	-	-
greedy	1	2	1	2	-	-	-	-	-	-	-	-	-

0.67), or laser (utility 0.33), followed by a request for *Avoid-Obstacle* that could use either sonar (utility 1) or the bumper (utility 0.5). In this case, vision was unavailable, and the random choices were for *Follow-Hall* to use the laser (a suboptimal choice, since sonar was available), leaving sonar for *Avoid-Obstacle*, for a total of 1.33. Meanwhile, the *greedy* approach (which MCH uses as its first stage of assignment) gave *Follow-Hall* sonar, which left *Avoid-Obstacle* with only the bumper, for a total of 1.17. Thus, the random approach made a better choice. However, this happened only once out of the ten runs, and the difference in utility was minor.

MCH+ provided further improvement above MCH, reaching as high as 140% improvement in global utility as resources became available (that is, in one case, MCH+

Figure 11. Box-whisker plot of happiness before failure. Higher happiness is preferable.

achieved a total utility of 2.0 while MCH only reached 0.83). This improvement in utility was possible because MCH does not exploit newly freed sensors; it only performs assignments when new requests are made. However, though MCH+ did achieve higher utility, it did so at the cost of disturbing any logical sensor which could use a newly freed physical sensor with a higher utility than its current assignment. A typical run is shown in Table 4, where the sequence of activations causes the global utility to vary according to the different algorithms. On average, MCH+ provided 8.88% greater global happiness than MCH, and 24.63% improvement over the *random* approach. These values are reflected in Figure 11.

However, while MCH+ provides a marginal improvement over MCH in terms of utility, it requires that opportunistic repair be performed, which disturbs existing assignments. Ideally, these disturbances should occur very seldom relative to the normal rate of requests. An analysis of the best time to perform an opportunistic repair is provided in Section 5.4, using real robot performance as its basis.

5.2.5 Time Performance

The LISP implementation was tested in simulation on different platforms, and the performance of the code was measured. When run on a Sun UltraSPARC 168Mhz computer, each event (allocation or deallocation) took approximately 4 milliseconds when there was little contention and approximately 5 milliseconds when there was high contention. On one of the robot's processors, an Intel Pentium 233, the same cases took 11 and 17 milliseconds.

5.3 Robot Results

Once the MCH algorithm had been tested successfully in simulation, it was demonstrated on robotic hardware, which competed in robot competitions held by AAAI in 1999 and 2000, and was demonstrated at the Tampa Museum of Science and Industry. The following section describes the robots used and the tasks they performed, as well as performance data for the algorithm. Through the competitions, MCH was ported to different software systems and to heterogeneous robot hardware, which validated the claim of portability. The competitions also provided an environment for unpredictable sensing needs, which tested whether MCH could operate under those conditions. These aspects of the algorithm must be tested on real hardware, as portability is difficult to test in simulation, and the effectiveness of the algorithm for handling real-world events must be tested in the real world. The algorithm performed favorably in these tests; at MOSI, MCH handled 71 changes over 40 minutes (an average rate of 106.5 changes per hour), requiring approximately 14 milliseconds each, or a total of approximately 0.99 seconds of processing over 40 minutes. A discussion of the MOSI tests appears in Section 5.3.6.

Figure 12. Nomad200 (Butler) dressed as a waiter shark (left), and as an armadillo (right).

Figure 13. Nomad200 (Leguin) dressed as a puffer fish (left) and as an armadillo (right).

5.3.1 Equipment

Two Nomad200 robots were used to compete in these competitions, Butler (shown in garb in Figure 12) and Leguin (shown in costume in Figure 13). Both robots are holonomic, capable of moving in any direction from a stop. Both robots are also equipped with dual color cameras on a pan-tilt unit, as well as bumper rings and radio ethernet for communication. However, their hardware is heterogeneous, because Butler has two rings of sonar, where Leguin has only one, and Butler uses a SICK planar laser ranger. The robots can also be equipped with a thermal sensor, which is typically given to Butler.

5.3.2 AAAI '99

The first tests of the MCH algorithm in a robotic task came in the form of the annual conferences and competitions of the American Association for Artificial Intelligence (AAAI). The robot competitions held by AAAI pose complex challenges for robots to complete, and are also a source for gathering experimental data.

Of the various competitions held each year by AAAI, the "Hors d'oeuvres, anyone?" competition (also dubbed the "waiter" competition) requires that the robot entrants serve appetizers to guests of the exhibition at the end of the conference. The robots are judged according to certain criteria, including the technical merit of their approach, how well they attract attention and interact with guests, and whether they are aware of the status of their proffered food. Further points are awarded for being able to recognize judges and other VIP's by their badges. The goal of the robot entrants is to circulate through a crowded room offering food, obtaining refills as needed, and interacting as much as possible with the people around them. The event takes place in the exhibition hall of the conference, which is usually a large open space with vendor booths and artificial lighting.

Table 5. Behaviors and sensors used by Borg Shark (Butler).

Behavior	Logical Sensor	Physical Sensors
face-find	faceFind	Camera, Thermal Sensor
AvoidObstacles	Depth360	Sonar ring
food-count	foodCount	Laser

Both Nomad robots were entered in the 1999 and 2000 competitions, using a similar strategy each year. As Butler had a larger variety of sensors available, she was designated as the primary waiter for the USF entry, and Leguin was tasked to bring a refill tray of appetizers from the refill station to Butler when needed.

5.3.3 Behaviors and States

In the 1999 competition, Butler was programmed to serve and request refills through a set of states, where each state had its own set of behaviors running to generate the activity appropriate for that state. This included driving between waypoints and serving at each stop. Butler would use the laser (placed behind the food tray) to detect when food was taken, and when she ran out of food, she would call to Leguin for a refill. The sequence of states is shown in Figure 14, and the active behaviors and their sensors are shown in Table 5. For this competition, the theme for the USF robot team was of ocean life, so Butler was a shark, while Leguin was a puffer fish.

Butler would begin by moving to a waypoint, using an obstacle-avoiding tactical behavior. This navigation required the use of sonars, which had to be allocated using MCH. When at a waypoint and offering food, Butler would use a *face-find* behavior that would use computer vision and a thermal probe to detect human faces. Meanwhile, the laser would be scanning for motion above the tray of food in the shark's mouth, which would indicate that food had been taken. If enough food had been taken, Butler would call Leguin.

Leguin would spend most of her time waiting by the refill station, "sleeping". This behavior consisted of using sonars to deflect the cameras away from any nearby

Figure 14. Sequence of states used by the shark robot in order to serve appetizers to people. Mapping of logical to physical sensors is shown in Table 5.

Figure 15. Sequence of states used by the puffer robot in order to bring food refills to the shark. Mapping of logical to physical sensors is shown in Table 6.

objects, with the intent of keeping people from touching the cameras (a common occurrence) as well as giving the robot some personality. Upon being called, Leguin would stop using the sonars for this behavior and instead use them for navigation, which required an explicit request. When she was near Butler (as close as dead-reckoning would allow), she was coded to use vision to face Butler, and upon being signaled, return home. This sequence is shown in Figure 15, and the behaviors and sensors used are shown in Table 6.

MCH was incorporated into a prototype for an Object-Oriented design of the SFX architecture, which is diagrammed in Figure 16. This was accomplished by adding

Table 6. Behaviors and sensors used by Puffer Fish.

Behavior	Logical Sensor	Physical Sensors
Sleepy	lookAway	Sonar ring
AvoidObstacles	Depth360	Sonar ring
TrackShark	sharkFind	Camera
Wait	Tactile	Bumper

a socket interface between the LISP code and the *Sensing Manager* process, such that the *Sensing Manager* would receive requests for sensing, which would be passed on to the MCH code. The MCH algorithm would specify some allocation, which the *Sensing Manager* would interpret and execute (by starting a new perceptual process). This process is shown in Figure 17.

The interaction with the MCH algorithm can be thought of in terms of selection; the *Sensing Manager* would start with a set of possible physical sensors, each of which would be tested to see if they satisfied the basic needs for the logical sensor making the request, then MCH would determine the best sensor to use from those remaining. Similarly, any freed sensing resources would be reported to the MCH code, which would signal the *Sensing Manager* to deallocate the resource. Since the SFX Architecture was implemented in C++ and run under UNIX, this was handled by finding and killing the appropriate perceptual process.

This particular domain did not create much contention for resources, because the sensors were really only needed by one task at a time as the robots stepped through their scripts. However, it did generate a large number of state changes, and subsequently, many changes to the set of active sensors. Unfortunately, due to hardware issues unrelated to MCH, the robots were unable to place in the 1999 AAAI waiter competition, but were awarded a Technical Achievement Award. As a result of the competition, however, various observations were made to improve the performance of the robot using behavioral sensor fusion, which is discussed below.

Figure 16. SFX Architecture diagram. The *Sensing Manager* (to the right) used MCH to decide how sensors should be allocated.

Figure 17. Role of MCH code as part of SFX Architecture. Due to the C++ implementation of SFX and the LISP implementation of MCH, these were forced to communicate using a socket interface.

The robotic team used a slightly different approach the following year, with better results.

5.3.4 AAAI 2000

For the AAAI 2000 "Hors d'oeuvres, Anyone?" entry, the Nomads were dressed as armadillos, a popular theme in the host city of Austin, Texas. Their basic approach to solving the task was the same, with Butler serving while Leguin would bring refills. This time around, Butler used an emotional model to determine the need for refills based on the rate of food being taken as well as the distance from Leguin, and would actively seek out and intercept Leguin if necessary to perform a refill if she was making little progress. Butler's sensing needs were similar to the previous entry, though a visual signal was used to indicate to Butler that she had been refilled. This required another explicit sensor request, which added some contention when used alongside the *face-find* behavior. In this competition, the robot team took third place overall, and the Nils Nilsson award for technical merit.

5.3.5 Sensor Failure

A strength of the MCH algorithm is its tolerance of sensor failure. This feature has been tested on Butler, making use of a panel of switches that have been wired to the power to each sensor on the robot. By flipping a switch, it is possible to cause a sensor to go offline, producing the same reaction that would be caused by a blown fuse (meaning that it is a hardware failure, rather than an algorithmic or software failure). Code was written to detect failure of different devices: for cameras, a signal sync can be detected, which vanishes if the camera is in any way disconnected; for sonars, the values tend to drift high (as the controller board is on a different circuit, and the sonars lack the amplification to send out the chirp they expect to receive back) when power is lost. Other devices (such as serial devices) may drop a carrier

signal, or will at least stop responding, which allows them to be detected after an interval.

During the 2000 AAAI "Hors d'oeuvres, Anyone?" competition, Butler was demonstrated doing a serving task while periodically having its sensors turned off. As expected, the robot would detect these failures and attempt to reallocate. In the case of the cameras, where there was some redundancy, the robot would recover by swapping over to the other camera if it was working. In the case of the sonars, Butler had no other navigational sensors, and would stop moving when the sonars failed (to prevent a possible collision). This took place concurrently with other requests for sensors.

5.3.6 MOSI

While performing at the AAAI competitions, the robots were making use of the MCH algorithm to handle changes in the allocation of sensors. This data was bolstered by a demonstration of the robots at the Tampa Museum of Science and Industry (MOSI) held to explicitly collect data. From these tests, various observations can be made. These results correspond to Butler, for two reasons: Butler required more changes of sensing state than Leguin, and Butler was equipped to simulate sensor failure (discussed in the previous section).

In the MOSI tests, the robots were run for 40 minutes. In this time, MCH received 71 total messages, which could be requests for sensors, the release of a sensor, or a change in the sensor's status (operational or broken). Of these, a total of 54 requests were denied, due to sensors being unavailable (in this case, due to sensor failure). Nine messages indicated a sensor failure, while five were corresponding sensor repair messages. There were two cases where MCH reassigned an active logical sensor to a different physical sensor to reduce conflict.

Figure 18. Plot of the cumulative CPU time used throughout a typical run at MOSI. The vertical axis is time in milliseconds, while the number of requests is shown on the horizontal axis.

In this test, the *Global Repair* stage was never needed, as the combination of behaviors and sensors used in this test did not produce much contention. Over the 40 minute run, the MCH implementation in LISP required a total of 990 milliseconds (meaning that on average, handling each message required approximately 14 milliseconds, and the entire test required less than a second of CPU time for MCH). Initializing the LISP interpreter to begin the tests required approximately 300 milliseconds, which is a one-time cost. The CPU time for a representative run is plotted in Figure 18.

The total happiness possible during the test for Butler was 2.0, which could be reduced by causing sensors to fail. Across the four runs that were measured, the average happiness after each request varied between 0.57 and 1.44. The happiness for a representative run is plotted in Figure 19.

Figure 19. Plot of the level of happiness at each request during a typical run at MOSI. The maximum possible in that configuration was 2.0; the vertical axis shows 2.5 to set an appropriate scale.

5.3.7 Summary of Results

From the data discussed above, it is evident that the algorithm was viable for a robotic task, and that it can handle numerous requests for sensing while also dealing with unpredictable events and sensor failure. The time taken for each assignment was less than 20 milliseconds, which suggests that this approach is valid for an on-line system. Although the real-world tests did not involve as many behaviors and sensors as the simulation, they still verify that the MCH algorithm can be incorporated into a robotic system and successfully used to handle sensor allocation.

5.4 Opportunistic Repair

An issue discussed above is that of opportunistic repair; that is, making changes to the sensing state in order to optimize overall utility between assignments. A problem that this creates is that if opportunistic repair is done too soon after an assignment is made (or, alternatively, too soon before a request), thrashing may occur. Therefore, in order to entertain the idea of opportunistic repair, such changes should only be made when the system is otherwise inactive. This section describes one possible method

Table 7. Delays for opportunistic repair by standard deviation. The rightmost column indicates the fraction of samples that fall within the given interval.

σ	Interval (seconds)	%
1.0	96.1	0.8413
1.645	140.1	0.95
2.0	164.4	0.9772
3.0	232.7	0.9987

for choosing an appropriate delay that should occur after a sensing request is made before opportunistic repair is allowed. This solution is based on the measured pattern of requests made to MCH within a particular task, and may not be appropriate for all tasks. This method could also be made adaptive, in order to adjust itself to the robot's activity, but such changes are beyond the scope of this thesis.

MCH does not use predictive methods to make its assignments, so it cannot directly compute the time until the next request. However, statistics can be used to ensure that an opportunistic repair is made sufficiently after any other request that thrashing is avoided. The measured activity of the robots while running at MOSI will be used as an example.

While letting the robot team run through their task for over 40 minutes, the pattern of sensor requests was captured, and afterward analyzed to determine what would be considered an adequate period of inactivity required before making an opportunistic repair.

Assuming that the rate of requests is normally distributed, the mean and standard deviation of the time in seconds between assignments (as captured above) are particularly useful. By choosing how many standard deviations from the mean to use, it is possible to estimate the fraction of requests that will take place after a particular delay. In particular, throughout eighty such intervals during the waiter task, the mean time in seconds between requests was 27.8 seconds, with a standard deviation of 68.3. Given these values, Table 7 shows what periods of inactivity would be required to assure that a given percentage of requests will have already happened.

Table 8. Experimental results for face-find logical sensor, without fusion.

	Reports Face	Reports ¬Face
Face	118/120	2/120
¬Face	33/120	87/120

face-find	Vision
True positives	98.3 %
True negatives	72.5 %

If a request came while waiting for this period to elapse, the delay would be restarted; these values represent a period after which it is deemed "safe" to reallocate. These values are only meaningful for the waiter task, but this method could be used to determine appropriate opportunistic repair delays for any application. For the case of the waiter task, these numbers indicate that almost 98% of all requests will occur within two minutes and forty-four seconds of the previous, and that after this delay, opportunistic repair should not cause thrashing. The actual number of standard deviations used should depend on the overhead of restarting behaviors (as more overhead should be risked less often), and tuned accordingly.

5.5 Behavioral Sensor Fusion Data and Analysis

The 1999 AAAI competition motivated the issue of sensor fusion, as two of the tasks facing the shark, to identify people and greet them, and to detect when food had been taken from a tray attached to the robot, were susceptible to false positive readings. The logical sensor which detected human faces was fooled by the color of the walls in the room, but could be improved by using a thermal probe to detect the presence of heat, and dismissing any face-like objects which were not at least above the ambient temperature. Similarly, the planar laser rangefinder that was used to detect when a person's hand had reached for food on the robot's tray was susceptible to seeing the food itself, or parts of the robot's shark costume. These false positives could be reduced significantly by dismissing any perceived motion before the laser that was not observed along with corresponding sonar readings (such as the person whose hand was near the laser).

Table 9. Experimental results for face-find logical sensor, when vision is fused with thermal probe.

	Reports Face	Reports ¬Face
Face	117/120	3/120
¬Face	0/120	120/120

face-find	Vision+thermal
True positives	97.5 %
True negatives	100 %

Table 10. Experimental results for food-count without fusion.

	Reports Hand	Reports ¬Hand
Hand	112/120	8/120
¬Hand	92/120	28/120

food-count	Laser + sonar
True positives	93.3 %
True negatives	23.3 %

Sensor fusion is certainly a desirable feature to have in a robotic architecture, as shown by the improvements in accuracy in a logical sensor when using fused sensors as opposed to using sensors individually. For instance, the human face detector, *face-find*, provided a high rate of successful classifications but it suffered from a high rate of false positives (as shown in Table 8). When fused with a thermal probe, which tested for heat above the ambient temperature, the *face-find* logical sensor was made more discriminating, which prevented false positives (Table 9). Similarly, the logical sensor designed to count the number of times food had been taken, *food-count*, showed improvement with sensor fusion (Tables 10 and 11).

As described in Chapter 2.3.1, these true positive and true negative rates can be combined using the algebraic product t-norm, producing utility values shown in Table 12. These values are reflected in Figure 20, which indicates that the fusion results (A and C) provide a higher utility than the results without fusion (B and D).

After MCH had been extended to support sensor fusion, as described in Chapter 4.8, it was tested to verify that it still performed as expected. In these tests, logical sensors which resembled those used at AAAI were given different groups of physical sensors (based on what the robot actually has), some of which were fused, such that a solution would exist when all logical sensors were active. This is shown in Table 13. A total of four logical sensors and six physical sensors were used for this test, meaning that it was feasible to test every permutation of the order in which the

Table 11. Experimental results for food-count logical sensor, when laser is fused with sonar.

	Reports Hand	Reports ¬Hand
Hand	112/120	8/120
¬Hand	2/120	118/120

food-count	Laser + sonar
True positives	93.3 %
True negatives	98.3 %

Table 12. Measured values with algebraic product combination.

Logical Sensor	True Positive Rate	True Negative Rate	Utility
face-find with thermal	0.975	1.0	0.975
face-find	0.9833	0.725	0.713
food-count with sonar	0.9333	0.9833	0.9177
food-count	0.9333	0.2333	0.2177

Figure 20. Plot of measured true positive and true negative values using algebraic product. A and B refer to *face-find* with thermal and without, respectively, while C and D refer to *food-count* with sonar and without, respectively. A higher utility is reflected by being higher on the plot.

Table 13. Test case setup. Logical sensors were given various physical sensors to use, with associated utility values. The values for face-find are measured, while the others are determined heuristically.

Logical Sensor	Physical Sensor(s)	Utility
Depth360	SonarRing	1.0
Depth90	SonarRing	1.0
	Laser	0.66
	Vision	0.33
food-count	Laser	1.0
	Bumper	0.5
face-find	(Vision, Heat-Sensor)	0.9756
	Vision	0.7555

Table 14. Test results for MCH and others for sensor fusion, using 24 permutations of possible requests. The second column indicates how many times the algorithm was able to handle all four sensing requests without failing, each of the 24 times. The second column shows how many requests, on average, each method managed to handle before failing. The utility column shows the average global utility that each method produced, based on the utility values shown in Table 13.

Method	All requests handled	Average requests handled	Average Utility
MCH	24/24	4.0	3.1327
Greedy	4/24	2.833	2.9045
Random	3/24	2.708	1.8481

logical sensors might make requests, for a total of 24 permutations, instead of testing over some randomly generated sets of requests, as was done in earlier testing.

In order to verify that the algorithm was performing at least as well as it had been before the modifications for sensor fusion had been made, two other allocation schemes were tried (*greedy* and *random*). This comparison was done in the same way as was described earlier in this thesis, though using a different set of data. Two metrics from that the previous tests (the average number of requests handled before failing to allocate, and the average utility before failing) are shown in Table 14, along with the number of times that each method satisfied all requests. As expected, the MCH code did significantly better than these other methods (as it had before the new changes).

5.6 Discussion

This thesis describes an algorithm for sensor allocation on mobile robots, with the claim that it will work using incomplete information, in a fashion that is tolerant of sensor failure, and without using any form of prediction about the sequence or duration of requests. In order to be worthwhile, the algorithm should also provide advantages over other methods. The data described in this chapter provides evidence of these claims.

First, regarding the issue of sensor allocation, the MCH algorithm was shown to work in simulation, performing assignments that were not only correct, but in all but one case, superior to other techniques tested. As for mobile robots, an implementation of the algorithm was successfully ported to a pair of heterogeneous mobile robots and demonstrated at successive AAAI robot competitions and MOSI. The algorithm has also been shown to be largely independent of the software system being used, as it performed equally well in simulation and two different robot software configurations.

In each case, Min-Conflict with Happiness dealt with requests and performed assignments in order, without reasoning about what future requests might be or what previous requests had been made. This enabled it to deal with the issue of sensor failure, where the unforeseen loss of a sensor was handled quickly and effectively, even while the robot was performing another task. The manner by which MCH performed its assignments was also guided by the principle of *least disturbance*, such that existing assignments would not be broken unless no other solutions were available.

Finally, in addition to satisfying requests for sensing, the algorithm supports behavioral sensor fusion, allowing higher utilization of sensors and improved sensor readings. Further, the algorithm performs well, requiring a nearly insignificant amount of processing time (less than 20 milliseconds, or 14 milliseconds on average) in order to serve requests.

It is by these demonstrations and tests that the Min-Conflict with Happiness algorithm has been verified and tested, using both simulation and mobile robot platforms. The claims that the algorithm is functional and provides advantages to other methods have also been supported through experimental data.

These data were somewhat limited by certain experimental issues, which merit brief discussion. First, gathering information about the patterns of requests from behaviors required that MCH be part of a larger software system, and as such, these tests reflect the functioning of the whole system. This means that the difficulty of the allocation problem on real mobile robots was based on the availability of behaviors that might request resources, and in some cases, there were not enough behaviors active at once to push the algorithm to its limits. However, simulation did provide a test of these more complex scenarios. Next, further simulation that demonstrates the usefulness of behavioral sensor fusion in a highly constrained situation with unfused sensors might be illustrative, but the tests for sensor fusion with the principles of the unfused allocation mechanism should serve to verify the algorithm's correctness.

The AAAI competitions may not have been the best venue for testing the algorithm, because they did not provide situations with much contention for sensors. However, given the performance at AAAI 2000 and MOSI, it appears that the underlying basis is sound.

5.7 Summary

This chapter has discussed the stages of testing intended to show the suitability of the Min-Conflict with Happiness algorithm to the task of sensor allocation on mobile robots. The algorithm was tested in simulation for situations involving large numbers of physical and logical sensors where high contention was possible. These tests show that the algorithm outperforms *greedy* and *random* allocation, based on the number of successful allocations before failing to satisfy a request. In particular, MCH handled 63% more requests than the *greedy* algorithm and 142% more requests than the *random* approach. Further, the addition of opportunistic repair (MCH+) provided an improvement in utility, offering 24.63% higher utility than *random* and 8.88% higher utility than MCH. The behavioral sensor fusion support of the algorithm was also tested with favorable results (in which MCH handled 4.0 requests on average, out of 4.0 possible, while *greedy* handled only 2.833 and *random* handled 2.708). Further, it was shown that using behavioral sensor fusion could improve the accuracy of sensor readings, providing an increase in the true negative readings of two sensors by 27.5% and 75%.

Once verified in simulation, the algorithm was integrated into two robotic software systems on heterogeneous mobile robots that competed in the 1999 and 2000 AAAI "Hors d'oeuvres, Anyone?" competitions. The algorithm performed as expected, even handling sensor failures, while the robots performed their tasks, earning third place and a Nils Nilsson award for technical merit. It is through these demonstrations and

tests that the algorithm's effectiveness has been measured, indicating that it works as well in a practical robot task as in simulation.

CHAPTER 6
CONCLUSIONS

This thesis was concerned with the problem of scheduling perceptual resources on a mobile robot such that logical sensors were mapped to physical sensors. Constraints on a solution included the lack of any prediction of sensing needs over time, the need to make dynamic changes (due to requests as well as sensor failure), and the need to run on-line. The proposed scheduling algorithm, Min-Conflict with Happiness (MCH), provides a method of allocation that satisfies these criteria. MCH represents the logical sensors available on a robot symbolically, associating each with a set of physical sensors that can be used for a perceptual task, and a utility measure that represents how accurate each physical sensor or set of fused physical sensors is for that task. When a new request for a logical sensor is made, MCH searches through the logical sensors in a manner that causes the *least disturbance* to existing assignments; that is, it first checks for solutions that can be made without breaking prior assignments, followed by checks that may disturb one assignment, followed by an exhaustive search. This thesis also presents a discussion of sensor utility, and offers a method of quantifying that utility in terms of real sensor performance.

This algorithm has been run through various tests, including simulation against other allocation schemes, and as part of a software system that was twice entered in the AAAI robotics competitions. These tests have included the detection of and compensation for failed sensors. The software has been demonstrated on robots with heterogenous perceptual hardware, and due to its implementation in Common LISP, can be run on most computational platforms.

6.1 Future Work

Work remaining beyond the scope of this thesis includes further investigation of the notion of *opportunistic repair*, in which changes to the robot's sensor allocation are periodically made to optimize the sensing performance between requests. The suggested method uses the mean and standard deviation of the time between requests, such that it is "safe" to perform opportunistic repair when new requests are unlikely (that is, when the robot's allocations have become stable for some period of time). This may require that the robot adapt to its own pattern of sensing requests in order to make opportunistic repairs effective.

There are a number of other possible extensions to this work, including the allocation of sensors between robots; in other words, given a group of robots, each of which may have unique sensors, to assign a particular robot's sensor to another robot's sensing task, or to choose a robot for a task based on its sensing capabilities. This ties into the problems of surrogate sensing and multiagent cooperation. The representation of sensors can also be extended to characterize what sensors can *achieve* a particular percept, so that instead of mapping logical sensors to a predefined set of physical sensors, the set of physical sensors could be determined at run-time based on their attributes.

REFERENCES

[1] R.C. Arkin. *Behavior-Based Robotics*. MIT Press, Cambridge, MA, 1997.

[2] T. Celinski and B. McCarragher. Experiments in the control of perception in multi-sensor system. In *Proceedings. 1999 IEEE/SICE/RSJ. International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI '99)*, pages 19–24, 1999.

[3] C.X. Chen and M.M. Trivedi. Task planning and action coordination in integrated sensor-based robots. *IIIE Transactions on Systems, Man, and Cybernetics*, 25(4):569–591, 1995.

[4] M. Dekhil, T.M.Sobh, and A.A. Efros. Commanding sensors and controlling indoor autonomous mobile robots. In *Proceedings of the 1996 IEEE International Conference on Control Applications*, pages 199–204, 1996.

[5] B. Donald and J. Jennings. Perceptual limits, perceptual equivalence classes, and a robot's sensori-computational capabilities. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1397–1405, 1991.

[6] E. Ephrati, M. Pollack, and J. Rosenschein. A tractable heuristic that maximizes global utility through local plan combination. In V. Lesser, editor, *ICMAS–95 Proceedings*, pages 94–101. MIT Press, 1995.

[7] M. Fox and M. Zweben, editors. *Intelligent Scheduling*. Morgan Kaufmann Publishers, San Francisco, CA, 1994.

[8] A. Gage and R. Murphy. Allocating sensor resources to multiple behaviors. In *IROS 99*, 1999.

[9] A. Gage and R. Murphy. Sensor allocation for behavioral sensor fusion using min-conflict with happiness. In *IROS 2000*, 2000.

[10] T. Henderson and E. Shilcrat. Logical sensor systems. *Journal of Robotic Systems*, 1(2):169–193, 1984.

[11] G.E. Hovland and B.J. McCarragher. Dynamic sensor selection for robotic systems. In *Proceedings of the 1997 IEEE International Conference on Robotics and Automation*, pages 272–277, 1997. vol. 1.

[12] M. Johnston and S. Minton. *Intelligent Scheduling*, chapter Analyzing a Heuristic Strategy for Constraint-Satisfaction and Scheduling. Morgan Kaufmann Publishers, San Francisco, 1994.

[13] G.J. Klir and B. Yuan. *Fuzzy Sets and Fuzzy Logic: Theory and Applications*. Prentice-Hall, Inc., Up Saddle River, NJ, 1997.

[14] F. Kobayashi, F. Arai, and T. Fukuda. Sensor selection by reliability based on possibility measure. In *Proceedings 1999 IEEE International Conference on Robotics and Automation*, pages 2614–19, 1999. vol. 4.

[15] R.E. Korf. Real-time heuristic search. *Artificial Intelligence*, 42(2–3):189–211, 1990.

[16] J. Lindner and R.R. Murphy. Learning the expected utility of sensors and algorithms. In *Proceedings of 1994 IEEE International Conference on MFI '94. Multisensor Fusion and Integration for Intelligent Systems*, pages 583–590, 1994.

[17] J.A. Lopez-Orozco, J.M. de la Cruz, E. Domínguez, E. Besada, and O.R. Polo. An open sensing architecture to autonomous mobile robots. In *Proceedings of the 1998 IEEE International Symposium on Intelligent Control (ISIC) held jointly with the IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA) Intelligent Systems and Semiotics (ISAS)*, pages 610–615, 1998.

[18] S. Matsubara and T. Ishida. Real-time planning by interleaving real-time search with subgoaling. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*, pages 122–127, Menlo Park, CA, 1994. AAAI Press.

[19] K. Menger. Statistical metrics. In *Proceedings of the National Academy of Sciences of the United States of America*, volume 28, pages 535–537, December 15 1942.

[20] S. Minton, M.D. Johnston, and P. Laird. Solving large scale constraint satisfaction and scheduling problems using a heuristic repair method. In *AAAI Proceedings*, pages 17–24, Menlo Park, California, 1990. AAAI Press.

[21] 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*, 58(1–3):161–205, 1992.

[22] R.R. Murphy and A. Mali. Lessons learned in integrating sensing into autonomous mobile robot architectures. *Journal of Experimental and Theoretical Artificial Intelligence, special issue on Software Architectures for Hardware Agents*, 9(2):191–209, 1997.

[23] F.R. Noreils and R.G. Chatila. Plan execution monitoring and control architecture for mobile robots. *IEEE Transactions on Robotics and Automation*, 11(2):255–266, 1995.

[24] J.C. Pemberton and R.E. Korf. Incremental search algorithms for real-time decision making. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*, pages 140–145, Menlo Park, CA, 1994. AAAI Press.

[25] J.K. Rosenblatt. Optimal selection of uncertain actions by maximizing expected utility. In *Proceedings 1999 IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA '99)*, pages 95–100, 1999.

[26] S. Russell and P. Norvig. *Artificial Intelligence: a modern approach.* Prentice Hall, 1995.

[27] H.S Stone and J.M. Stone. Efficient search techniques - an empirical study of the n-queens problem. *IBM Journal of Research and Development*, 31:464–474, 1987.

[28] H. Xu and J. Vandorpe. Perception planning in mobile robot navigation. In *Proceedings on 1994 IEEE International Conference on MFI '94. Multisensor Fusion and Integration for Intelligent Systems*, pages 723–730, 1994.

[29] M. Youssefmir and B. Huberman. Resource contention in multiagent systems. In V. Lesser, editor, *ICMAS–95 Proceedings*, pages 398–403. MIT Press, 1995.

[30] Y.F. Zheng. Integration of multiple sensors into a robotics system and its performance evaluation. *IEEE Transactions on on Robotics and Automation*, 5(5):658–669, 1989.