Sensor Scheduling in Mobile Robots Using Incomplete Information via Min-Conflict With Happiness

Aaron Gage and Robin Roberson Murphy

***Abstract*—This paper develops and applies a variant of the Min-Conflict algorithm to the problem of sensor allocation with incomplete information for mobile robots. A categorization of the types of contention over sensing resources is provided, as well as a taxonomy of available information for the sensor scheduling task. The Min-Conflict with Happiness (MCH) heuristic algorithm, which performs sensor scheduling for situations in which no information is known about future assignments, is then described. The primary contribution of this modification to Min-Conflict is that it permits the optimization of sensor certainty over the set of all active behaviors, thereby producing the best sensing state for the robot at any given time. Data are taken from simulation experiments and runs from a pair of Nomad200 robots using the SFX hybrid deliberative/reactive architecture. Results from these experiments demonstrate that MCH is able to satisfy more sensor assignments (up to 142%) and maintain a higher overall utility of sensing than greedy or random assignments (a 7–24% increase), even in the presence of sensor failures. In addition, MCH supports behavioral sensor fusion allocations. The practical advantages of MCH include fast, dynamic repair of broken schedules allowing it to be used on computationally constrained systems, compatibility with the dominant hybrid robot architectural style, and *least-disturbance* of prior assignments minimizing interruptions to reactive behaviors.**

***Index Terms*—Incomplete information, mobile robots, robot architectures, sensor allocation, sensor fusion, sensor scheduling, sensor utility, uncertainty.**

I. Introduction

OUR WORK in autonomous mobile robots is motivated by the need to have the best possible sensing at any given time for any condition of the sensor suite. A fundamental assumption of this work is that the mobile robot will use a hybrid deliberative/reactive architecture, the dominant architectural style at this time [1]. Under a hybrid architecture, behaviors couple sensing (perceptual schemas) with action (motor schemas), following [2]. The motor and perceptual schemas act as autonomous, independent agents; essentially logical behaviors and sensors [3]. In general, the set of active behaviors is determined by a deliberative *mission planner* which generates the plan, or a partial plan, for achieving the mission, and a *sequencer* which instantiates the behaviors (as schemas) needed at any given time to accomplish the current task. Note that the sequencer is a specialist agent: it cannot relax the plan, only satisfy it or return an error to the mission planner. The sequencer allocates the resources needed for a behavior. The sensing resources may be either a single sensor or the fusion of multiple sensors (*behavioral sensor fusion*) [1].

Allocating sensing resources to satisfy a set of behaviors is difficult due to contention and having to resolve that contention with incomplete information. A *sensing resource* consists of control over the sensor position (e.g. pointing), control over hardware parameters (e.g. autofocus, white balance, firing frequency, etc.), and the format of the output. The association of a physical sensor with a perceptual processing algorithm forms a *logical sensor*. A percept may be generated from one of a set of equivalent logical sensors, so a behavior often has choices in which sensor it uses. In a behavioral system, more than one behavior can share the output from the same sensor and process the output using perceptual schemas, if the algorithms expect the sensor to be at the same position with the same parameters.

Contention occurs when such sharing is not possible. Mobile robots typically have relatively few sensors (sonar and vision) while executing a larger number of concurrent behaviors (avoid, navigate, search), creating a one-to-many mapping. The physical mounting of sensors often prevents sharing of sensors for behaviors (for example, one behavior may need the camera to face forward for navigation while another simultaneously requires the camera to face to the side for searching [4], [5]). The possibility of contention is increased in practice due to inevitable sensor failures. As a result, a mobile robot must be able to identify and substitute alternative sensing strategies as needed.

Contention for sensing resources falls into three categories.

1) *No contention*. This arises when all perceptual schemas have access to their preferred sensing resources during their execution. This could be the result of each behavior having a perceptual schema with a dedicated sensor, or multiple perceptual schemas that are able to share a sensor. This is the simplest case and is included both for completeness and to emphasize that the allocation mechanism should be effective for both complex and simple situations.
2) *Competition by two or more perceptual schemas for the same physical sensor*. In this case, multiple perceptual schemas prefer the same resource, which cannot be shared. For example, two schemas may want control of the same camera, but require different hardware parameter settings.
3) *Competition by two or more perceptual schemas for the same effector*. This case represents the example of

Manuscript received June 17, 2002; revised May 2, 2003. This work was supported in part by the NSF under Grant IRI-9320318, DARPA under Grant AO#B460, and ONR/NRL under Grant N00173-99-P-1543. This paper was recommended by Associate Editor C.-T. Lin.

The authors are with the University of South Florida, Tampa, FL 33620 USA (e-mail: agage@csee.usf.edu; murphy@csee.usf.edu).

Digital Object Identifier 10.1109/TSMCB.2003.817048

a camera attempting to look down a hallway and at the wall simultaneously, with either the camera pan-tilt head or the robot body being the effector. Eventually a scenario equivalent to a person being able to turn their head only so far before being forced to turn their body will surface. This case is distinct from the previous case of competition between multiple perceptual schemas in that it adds the challenge of how perceptual and motor schemas communicate, negotiate, and/or are supervised.

The goal of sensing resource contention is *to find a set of adequate substitutions which permit all behaviors to continue to execute concurrently and to optimize the allocation based on the appropriate utility metric (ex. certainty)*. Otherwise, the plan, which assumes concurrent execution, must be declared as failed. The solution implemented in [4] of changing the order of execution of behaviors (i.e., each behavior takes a turn at controlling the sensing resource) could not arise from the sequencer or scheduler, but must reflect a change by the task planner. This ensures that changing the order of the tasks to satisfy sensing does not violate some other domain constraint. Consideration of domain constraints is, by definition, a function of the planner, not the scheduler.

Contention resolution is challenging because of incomplete information. On one hand, the sequencer should minimize reallocations of sensing resources, otherwise a condition of no effective progress (akin to *thrashing* in operating systems) occurs. However, if the sequencer makes the best possible allocations over the set of active behaviors at any given time without any projection of what the upcoming behaviors are, it may also lead to thrashing.

The levels of knowledge associated with various behaviors will impact a scheduler. At this point in time, three levels have been identified. In order to make this discussion clearer, the following terminology is introduced. The set of behaviors which are active at the same time is denoted by B (note that B is distinct from the set of all possible behaviors). Robot execution can then be viewed as a sequence of i sets of active behaviors, $B_{1,I_1}, B_{2,I_2}, \ldots, B_{i,I_i}$, each of which has a particular start time and duration denoted with the subscript I_j, for the jth duration interval. Note also that resources must be allocated to satisfy the current B_{i,I_i} for duration I_i in order for the plan to continue to be executed.

The levels from most to least informed are as follows.

Level I: *Can project the sequence of active behaviors, $B_{1,I_1}, B_{2,I_2}, \ldots, B_{n,I_n}$, and the duration of time the robot will spend executing each set*. In this case, all n sets are known as well as how long the system will reside in execution of each set. This situation would arise when the robot was traversing a known area using a predefined collection of behaviors; for example, using an abstract navigation behavior to go down a hallway of known length or average traversal time. This is the ideal case, where the scheduler has a time horizon that it can optimize over. However, this projection may be disrupted by reality; in particular, by a highly cluttered hallway or a sensing failure. Therefore, the scheduler must be able to handle dynamic changes even for this case.

Level II: *Can project the sequence of active behaviors, $B_{1,I_1}, B_{2,I_2}, \ldots, B_{n,I_n}$, but* not *the duration of time the robot will spend executing each set*. All n sets are known but not the associated duration, I_j, of each. This would be the case where the robot was using a predefined collection of behaviors to navigate an area not stored in memory (i.e. going down a new hallway).

Level III: *Cannot predict the sequence of active behaviors or the duration*. Neither n nor I_j is known. In this event, the robot would be operating at a purely reactive level. It also includes knowing only a subset of each B_i, where the robot might know that *avoid-obstacle* will always be active, but nothing about other behaviors.

This paper addresses scheduling only for the Level III case. This level is representative of the current state of the practice in hybrid architectures, where most mechanisms are purely reactive or memoryless and model-less. That is, an overall plan or sequence of behaviors is not necessarily known or available (especially in the case of purely reactive or hybrid architectures), so this approach does without. This does not, however, preclude scheduling for other cases; while the approach here does not make use of the sequence or duration of sensing requirements, it will work in systems where that information is available (by simply disregarding it, or by using the information to perform opportunistic repair appropriately, as will be discussed in Section VI). The quality of solutions from this approach suggests that more sophisticated techniques for scheduling for Levels I and II may not be necessary.

While the types of contention and levels of available information are helpful in framing the research issues from a theoretical perspective, mobile robots introduce practical considerations as well. We define the practical constraints on the scheduling of sensing resources for an autonomous mobile robot as follows.

- *It is not a dispatch problem*. The scheduler does not change or control the order of execution of the behaviors; instead, it schedules resources to permit the execution of the behaviors in the order prescribed by the task planner.
- *It is a multiagent problem*. The perceptual schema for each behavior represents a separate agent. Furthermore, we assume that the agent has limited intelligence; it can construct a partial preference ordering on its different methods for sensing the percept.
- *The agents are uncooperative*. Each behavior wants to maximize its own goals, and thus, wants the best sensing for its purpose.
- *Search is combinatorially small*. Unlike job shop scheduling and other traditional scheduling applications which may attempt to schedule thousands of activities, a robot is expected to have ten or fewer behaviors active at a time. However, this does not mean that the search space is trivial enough to merit expanding every possibility. In particular, sensor sharing and sensor fusion make for a large search space (see Section III-F), and it is anticipated that sensors may be allocated across a distributed robotic system, so that even though the computational complexity of the problem is small due to a small number of sensors per robot, there may be multiple robots being managed at

once. Further, a design goal is to minimize the overhead of changing existing assignments (the idea of *least disturbance*) when determining where to assign a physical sensor given an existing sensing state. This requires more than a simple expansion of all possible search states.

- *The potential for resource contention is high*. Although a robot may have few concurrent behaviors, it will probably have even fewer physical sensors.
- *The time devoted to scheduling is limited*. The robot must operate on-line, meaning that it cannot deliberate indefinitely. The allocation mechanism must be an anytime algorithm.
- *The agents may or may not be able to project sensing needs into the future*. As noted earlier, future sensing demands may or may not be known, and the robot may encounter unpredictable sensing problems. Therefore, the system needs to be able to repair a set of sensing resource allocations.
- *A task or behavior cannot be removed from the plan in order to make scheduling resources possible (no relaxation)*. If a task or behavior was removed, the plan itself has failed and must be repaired.

This paper views allocation of sensing resources to behaviors as a scheduling problem, rather than a planning problem per se, for reasons described in detail in Section II. It does not consider allocating sensing for strictly deliberative processes (e.g., map-making), though that change is expected to be a trivial extension. The article presents a novel modification of the Min-Conflict [6] algorithm, called *Min-Conflict with Happiness (MCH)*, which is an efficient heuristic algorithm able to dynamically optimize the sensing "happiness" of each behavior over the entire set given incomplete information. The modification is shown through simulation to be superior to other methods in terms of the number of assignments it can make (63% over greedy and 142% over random) and in overall utility (up to 140% higher), and data collected from implementation on mobile robots confirms that it is able to handle unpredictable requests, sensor failures, and behaviors which require sensor fusion. (See Section IV, V.) In addition, the algorithm is portable between robots and robot platforms, as shown by its deployment on two robots with different sensor suites in Section V. As a result, the article contributes a general solution to scheduling resources which can be applied to any hybrid architecture.

II. Related Work

Our approach is to treat sensor resource allocation as a scheduling problem, and to use a variant of the Min-Conflict algorithm to perform the allocations. The resulting Min-Conflict with Happiness heuristic algorithm finds a contention-free mapping of *physical sensors* (hardware devices that sense the environment, such as sonar rangefinders, cameras, and thermal sensors) and *logical sensors* (algorithms that can derive percepts, such as the presence of a stimulus, given the raw data from one or more physical sensors). This mapping is guided by the relative utility ("happiness") of each combination of physical and logical sensors, where utility can be defined per implementation, and is only needed to sort possible mappings into a preference ordering. Conflicts are defined as assigning one physical sensor to two or more logical sensors *that require mutually exclusive configurations of that sensor*. For the presentation and testing of this algorithm, it is assumed that no two logical sensors can share a physical sensor, so assigning a physical sensor to two or more logical sensors will result in a conflict. However, given a mechanism to determine whether multiple logical sensors can simultaneously use the same physical sensor without conflict, this algorithm easily supports sharing of sensors. Finding such a conflict-management mechanism is beyond the scope of this work. In particular, such a mechanism requires that reactive behaviors (which, by definition, are unaware of each other) be able to represent their sensing requirements in such terms that the scheduler can determine whether any given set of assignments would generate conflict. Given the multitude of parameters across numerous sensing modalities, finding a general solution to this problem is nontrivial, and is a direction for future work.

Given this description of the problem, this section reviews related work, starting with an overview of resource allocation in robotics, and ending with an analysis of the Min-Conflict algorithm, which we use as the basis for our approach. Preliminary work has been done in resource scheduling for robotics, but unlike our approach, either assume complete information or violate the criteria established for hybrid architectures in Section I. Min-Conflict [6]–[8] is the closest existing algorithm for this problem, but is not sufficient without modification.

A. Resource Scheduling in Robotics

The domains of scheduling algorithms and resource allocation have been widely studied (see [9] for a survey), but primarily in terms of AI applications. Scheduling and allocation for planning have been explored in solving such problems as the n-queens and machine shop scheduling, using the Min-Conflict with repair heuristic [8]. Planning in real time has been explored through such methods as Real-Time-A* search [10] and its variations [11], [12].

Work more directly related to robotics has been done in resource allocation, most noteworthy are [13], [14]. The method in [13] relied on the economics associated with each plan, such as the estimated reward and completion time, which are not available to a scheduler with Level III knowledge. Other work has been done in planning for resource contention [14], though that approach assumed that sensors could be shared between behaviors. The implementation described in this paper does not make this assumption, but this issue remains open for future work.

Hovland and McCarragher [15] 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 paper: their execution time measure is directly related to the update rate, which is used in this paper to eliminate possible assignments. Further, the reward for using each process monitor resembles the utility ("happiness") used in this paper, 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.

Other approaches to sensor allocation include the use of Hidden Markov Models and POMDP [16]–[18], and others have modeled the problem using stochastic differential equations [19], [20]. These models are largely theoretical, and their application to the domain of mobile robots is unproven. Specifically, these approaches tend to consider choosing one of a set of potential sensors or measurements, but do not address the issue of contention between simultaneous assignments.

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 [21], including the advantageous use of new information. Our efforts concentrate on the inverse problem: how to plan for sensing. Further, we wish to avoid the computational overhead and time delays of interleaving planning and execution at the reactive layer incumbent in the approach in [21] unless either a failure occurs (which mandates a new plan), or an opportunity arises for significant gains in sensing. Architectures that detect or compensate for problems such as sensing failures have also been proposed [22], [23]. The architecture in [22] detects failures by monitoring the motor behavior from the deliberative layer and repairing plans.

The SFX architecture [23] is an agent-oriented hybrid architecture that keeps the tasks of monitoring and exception handling at the behavioral level. SFX has an explicit agent called the *Sensing Manager* that manages the robot's sensing resources. This architecture is the target for the scheduler described in Section III.

Aspects of our approach bear a superficial resemblance to the CIRCA architecture [24]. That system is primarily concerned with guaranteeing safe real-time execution of behaviors. Sensing influences the frequency of execution of Test-Action Pairs (TAP's). If a sensing resource is not available or adequate, CIRCA directs the robot to substitute another sensing resource or slow the robot in order to reduce the sensing demands. No discussion is given of how an appropriate substitute sensing source is identified, nor of the impact that the selection process would have on the TAP's schedule (for instance, would all activity cease while the sensing allocation was determined?). The architecture is also heavily dependent on the assumption that the robot's interactions with the environment can be modified, allowing all states to be known. As noted in Section I, the sensing resource allocation is complicated by unpredictable sensing failures. Instead, CIRCA and our work can be taken as complementary; one concentrates on scheduling sensing and acting, while the other schedules sensing in such a way as to make sensing and acting possible.

B. Min-Conflict

The Min-Conflict heuristic was chosen as a basis for scheduling sensor resources based on its success in other scheduling and constraint satisfaction problems such as machine shop scheduling and n-queens [8]. Min-Conflict is repair-based, so it tends to be more informed than traditional depth-first generative searches with backtracking (such as in the Sicstus system [25]), and as a result, tends to reach solutions faster; it is shown in pseudocode in Fig. 1. A comparison of Min-Conflict to other algorithms for solving constraint satisfaction problems, including generative backtracking and "most-constrained first" can be found in [6].

```
MIN-CONFLICT heuristic
1   Assign values to all variables in V,
2           even if this violates constraints
3   while CONFLICT-EXISTS(V)
4     do v ← FIND-VARIABLE-IN-CONFLICT(V)
5           Assign v a value that minimizes the number
6           of conflicts in V, breaking ties randomly
```

Fig. 1. Min-Conflict heuristic. Given a set of variables V and a set of binary constraints (represented by CONFLICT-EXISTS), iteratively reduce the number of variables in conflict by choosing a variable in conflict and assigning a new value that results in the minimum number of conflicts.

However, Min-Conflict as described in [6] must be modified due to several issues in mobile robots. First, Min-Conflict breaks ties randomly when making repairs, which is not appropriate for choosing among heterogeneous sensing strategies. Consideration for the utility of an assignment, dubbed "happiness", is required to select the best sensor for a task from the sensors available. Next, also due to the heterogeneous nature of the sensing strategies, repairs should first be attempted on the logical sensors that have the most *flexibility*; that is, a logical sensor that can generate a percept using one of many physical sensors (a *flexible* logical sensor) should be chosen for reassignment before a logical sensor that can only use one physical sensor. This bias toward more *flexible* sensors directs the search away from probable dead-ends (logical sensors that cannot be given a different physical sensor). Third, Min-Conflict does not consider the overhead (startup time) incurred by repairs. In order to avoid thrashing (a loss of productivity due to frequent changes in sensing strategies), assignments should not be broken if some other alternate solution exists. This is referred to as the principle of *least-disturbance*. Finally, the issue of behavioral sensor fusion presents further modifications: that certain sets of physical sensors may be assigned together to satisfy a single request.

In previous work, we have proposed the needed modifications to Min-Conflict for single sensor systems [26] and for sensor fusion systems [27]. This article unites and expands upon these previous efforts, contributing a single, coherent algorithm and detailed description of results, along with a discussion of the role of contention and incomplete information.

III. Min-Conflict With Happiness Algorithm

As outlined in Section II-B, the extension of Min-Conflict to create Min-Conflict with Happiness (MCH) deliberately reflects certain design criteria, including the utility of assignments, the minimization of disturbance to existing assignments (to minimize overhead costs), heterogeneous assignments (as not all logical sensors can use the same physical sensors), and allowing multiple assignments to be made atomically (for sensor fusion). The heuristic algorithm described in this section

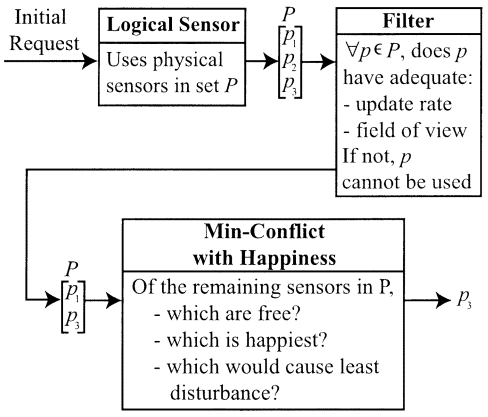

Fig. 2. Selection process performed to determine the best physical sensor to allocate.

resembles Min-Conflict, but contains numerous modifications and abstractions to be applicable in the domain of sensing for mobile robots.

The Min-Conflict with Happiness algorithm is event-driven, taking requests from logical sensors for physical sensors, and determining which physical sensor (or sensors, in the case of sensor fusion) should be assigned to the requesting logical sensor. To satisfy the request, MCH takes as input a list of physical sensors (some of which may be disqualified by being broken), and a set of logical sensors, each of which has a list of the physical sensors it can use and at what utility. MCH produces a new mapping of physical sensors to logical sensors (that is, a schedule), or may signal an error if no solution to the scheduling problem exists.

The utility for each physical sensor that a logical sensor can use may be defined as needed; it is simply a number, and can be based on measured performance or assigned arbitrarily. In general, a decrease in utility corresponds to an increase in sensor uncertainty. A discussion of two utility measures explored for this work can be found in Section VI. Some of these physical sensors may be removed from consideration if they are malfunctioning or if the logical sensor's update rate with a physical sensor is inadequate for the robot's current velocity. Of the remaining physical sensors, the Min-Conflict with Happiness algorithm selects which should be used. This process is shown in Fig. 2. This section will concentrate on the MCH part of this process only.

A. Notation and Assumptions

The Min-Conflict with Happiness algorithm has three primary steps: *Greedy Allocation* which makes assignments, followed by *Local Repair* that is called if conflicts are created by *Greedy Allocation*, and *Global Repair* that is called if *Local Repair* cannot repair a conflict. A variant on the algorithm, MCH+, adds an opportunistic repair step after *Global Repair* to improve assignments between request events.

This section 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). Pseudocode for the algorithm can be found in Fig. 3–7.

```
    ACTIVATE-LOGICAL(l)
1     ▷ First assign using Min-Conflict
2     A ← ordered alternates list for l
3     bestSensor ← MINIMUM-ASSIGNED(A)
4     ▷ minimumAssigned finds the first alternate
5     ▷ with least conflict
6     ADD-LOGICAL-TO-SENSOR(l, bestSensor)
7     for c ← 1 to n
8         do if CONFLICT-EXISTS(P[c])
9             then LOCAL-REPAIR(P, c)
10    for c ← 1 to n
11        do if CONFLICT-EXISTS(P[c])
12            then GLOBAL-REPAIR(P, c)
```

Fig. 3. Min-Conflict with happiness algorithm. Assumes 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.

```
LOCAL-REPAIR(P, c)
1     ▷ Displace logical sensors to other physical sensors
2     ▷ to reduce local conflict
3     flexList ← list of logical sensors assigned to P[c]
4         in the order of how many alternative sensors each
5         logical sensor can use, from most to least
6     for each l_i in flexList while CONFLICT-EXISTS(P[c])
7         REMOVE-LOGICAL-FROM-SENSOR(l_i, P[c])
8         success ← ASSIGN-TO-BEST-FREE-SENSOR(l_i)
9         if success = FALSE
10            then
11                RESTORE-LOGICAL-TO-SENSOR(l_i, P[c])

ASSIGN-TO-BEST-FREE-SENSOR(l)
1     ▷ Assign logical sensor to best unused
2     ▷ physical sensor in list A
3     for j ← 1 to k
4         do if NUM-ASSIGNED(A[j]) = 0
5             then ADD-LOGICAL-TO-SENSOR(l, A[j])
6                 return TRUE
7     return FALSE
```

Fig. 4. MCH algorithm, continued. LOCAL-REPAIR is applied to any logical sensor with a resource conflict immediately after the initial assignment.

```
GLOBAL-REPAIR(P, c)
1     ▷ Assumes that all possible local repairs have been made
2     flexList ← list as in LOCAL-REPAIR
3     Start list takenSensors with P[c]
4     solutionPath ← FIND-PATH(flexList, takenSensors)
5     if solutionPath ≠ NIL
6         then APPLY-SOLUTION-PATH(solutionPath)
```

Fig. 5. MCH algorithm, continued. GLOBAL-REPAIR attempts to resolve conflicts left behind by LOCAL-REPAIR by starting a recursive call to FIND-PATH.

It is assumed that physical sensors cannot be shared, and if a physical sensor is simultaneously assigned to two logical sensors, then this is a conflict that must be repaired. If a mechanism were added to the *CONFLICT-EXISTS* function that determined whether certain logical sensors could both use a physical sensor, then the algorithm would support sensor sharing. However, such a mechanism has not been explored.

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.

These assumptions require that some representation of the physical sensors and logical sensors be established. Physical

```
FIND-PATH(flexList, takenSensors)
1     ▷ Recursive procedure to find what changes must
2     ▷ take place on other sensors to make room for
3     ▷ a logical sensor from the current sensor.
4     if flexList = NIL
5         ▷ This path is a dead-end
6             then return NIL
7     repairList ← NIL
8     candidate ← FIRST(flexList)
9     sensorList ← list A from candidate
10    possibleSensors
11            ← SET-DIFFERENCE(sensorList, takenSensors)
12    currentSensor ← FIRST(possibleSensors)
13    if possibleSensors = NIL
14          OR CONFLICT-EXISTS(currentSensor)
15        ▷ Go on to the next logical sensor
16        ▷ on this physical sensor
17              then return FIND-PATH(REST(flexList),
18                    takenSensors)
      Continued in Fig. 7
```

Fig. 6. Algorithm, continued. The FIND-PATH procedure is the heart of the GLOBAL-REPAIR procedure, recursively searching for combinations of reassignments that will reduce conflict. The remainder of the procedure is listed in Fig. 7. Note that the SET-DIFFERENCE function treats its arguments as sets and finds the difference between them, so the result of line 11 will be all of the physical sensors in *sensorList* that are not in *takenSensors*. Also note that the list A in line 9 is the list of alternates for the candidate logical sensor.

```
      Continued from Fig. 6
19    if numAssign(currentSensor) = 0
20        ▷ Logical sensor can be moved to this sensor
21            then append candidate and currentSensor
22                  to repairList
23                 return repairList
24    else
25        ▷ Test if logical sensor on currentSensor may be
26        ▷ moved to make space (recursively)
27        Add currentSensor to takenSensors
28        repairList ← FIND-PATH(
29            LOGICAL-USING-SENSOR(currentSensor),
30              takenSensors)
31        ▷ If that found a solution, use it
32        if repairList ≠ NIL
33            then
34            add LOGICAL-USING-SENSOR(currentSensor)
35                  and currentSensor to repairList
36                 return repairList
37        ▷ Otherwise, continue (note that currentSensor has
38        ▷ been added to takenSensors, so the recursive
39        ▷ step is a smaller case)
40        else
41            return FIND-PATH(flexList,
42              takenSensors)
```

Fig. 7. FIND-PATH procedure, continued from Fig. 6. Note that the LOGICAL-USING-SENSOR function locates and returns a reference to the logical sensor that is using the identified physical sensor (which is *currentSensor* in both cases).

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 alternates 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). An overview of the steps and representation in the algorithm is provided in Fig. 8.

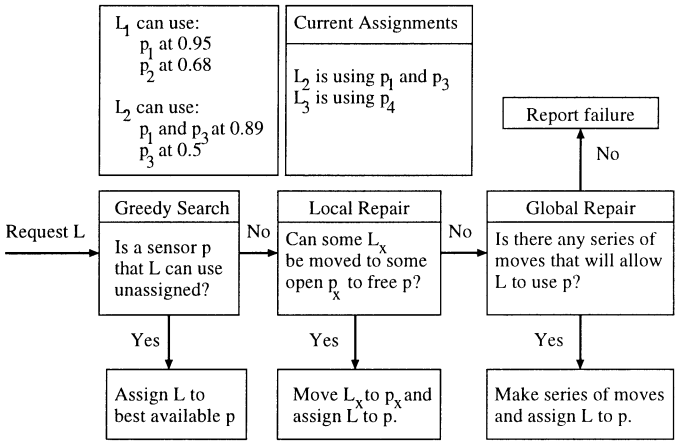

Fig. 8. Overview of the MCH assignment process, in terms of the individual steps and representation used.

B. Step 1: Greedy Allocation

This is the first step in the allocation of a physical sensor p to a logical sensor l, and begins when a request is made on behalf of a logical sensor. The list of alternates A for the logical sensor l (that is, all of the physical sensors that l could use) is created, and hard constraints are applied (meaning that if a sensor is broken or otherwise unsuitable, it is removed from consideration). The remaining alternates A are sorted by their utility and checked in order for availability. The logical sensor l is provided with the first physical sensor p_x (or set of fused sensors) available. If none of the alternates are available, 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. Note that the ASSIGN-TO-BEST-FREE-SENSOR function may either return a single physical sensor or a set of fused sensors.

C. Step 2: Local Repair

In the original Min-Conflict algorithm, conflicts that exist after the initial assignment phase are resolved by choosing a variable in conflict and randomly reassigning it in such a way as to reduce its conflict. In the domain of mobile robots, this approach is inappropriate, as there are costs involved in changing an assignment, and not all solution states are equal (some have a higher utility than others). The first step of MCH follows Min-Conflict, but the repair steps that follow are tailored to the mobile robot domain, which requires a departure from Min-Conflict.

For each physical sensor p assigned in conflict in Step 1, a list $(l_1, \ldots, l_m)$ of the logical sensors that p is assigned to 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.

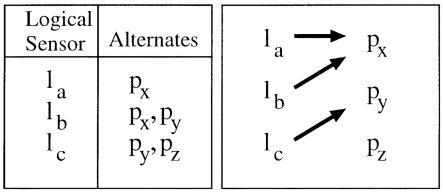

Fig. 9. Case where Global Repair is needed.

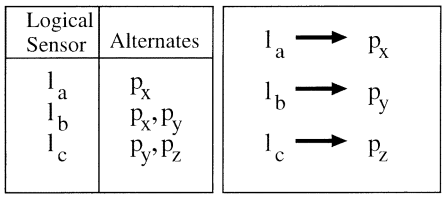

Fig. 10. Solution after Global Repair.

This ordering also reduces checks for alternates for any logical sensor 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, for 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).

D. Step 3: Global Repair

The final step of repair is by far the most complex, time-consuming, and powerful of any step. 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 happens to be free. This is illustrated in Fig. 9. Given this scenario, although the first two steps 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 Fig. 10). 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* step 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 step begins by choosing a physical sensor p_c with conflicting assignments after the *Greedy Allocation* and *Local Repair* steps have finished. It also creates a list, *takenSensors*, of physical sensors that have been tested already (initially empty) and adds p_c to it (GLOBAL-REPAIR, line 3). This will prevent the search from running in cycles. Note that in the case of p_c belonging to a fused set of sensors, only p_c should be added to the list, not the entire fused set (as this would eliminate otherwise untested alternates). The logical sensors using p_c are placed in a list, called *flexList*, sorted by the number of alternates each can use from most to least (see GLOBAL-REPAIR, line 2). Next, the set of sensors that each l_i can use is built into a new list of alternates called *sensorList* (FIND-PATH, lines 8–9). Physical sensors that have already been tested are removed, resulting in *possibleSensors* (FIND-PATH, lines 10–11). If *possibleSensors* is empty, or there is already a conflict on the alternate physical sensor being considered, the next logical sensor is tested recursively (lines 12–18).

At this point, there were no conflicts on the physical sensor under consideration, meaning that p_c has no conflicts because it is unused, or p_c has no conflicts because the assignment(s) on it do not constitute a conflict (in the event that a sensor sharing mechanism were present).

If there was no assignment to p_c, then a base case has been reached (FIND-PATH, lines 19–23). Otherwise, if there was an assignment, the search recurses, checking the next alternate (lines 25–30). The result of this recursion is stored; if it leads to a solution, then the solution is returned (lines 31–36). If it did not find a solution, then the search recurses on the next alternate physical sensor for the current logical sensor being tested (lines 37–42).

Upon success, this step will produce a solution in the form of an ordered list 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 creates 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.

E. Step 4: Opportunistic Repair (MCH+)

As it is written, the MCH algorithm focuses on making assignments only when physical sensors are requested. That is, it is entirely event-driven. 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.

A variant on MCH, called MCH+, allows for optimization between requests. MCH+ adds an *Opportunistic Repair* step, 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. 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.

The issue of the optimality of MCH+ includes not only whether a particular set of assignments is optimal (in terms of utility), but also how much overhead must be incurred by rearranging assignments in order to reach that optimality. In fact, MCH+ may not discover completely optimal solutions (those of the highest utility), but for those cases, it would be necessary to break existing assignments (incurring overhead) to increase utility further. Other approaches apply nonlinear methods to reach optimal solutions (such as stochastic differential equations [19], [20]), but the application of those methods to robots is untested.

F. Complexity

The worst case performance of the MCH algorithm is as follows. The basic operation for this analysis will be a test for whether a resource is available, p refers to the number of physical sensors on the robot, and l refers to the number of logical sensors in the system. The *Greedy Allocation* procedure traverses the sorted list of alternates for a logical sensor, stopping when it finds one that is free or when it exhausts the list (which happens only when all alternates are in use). For this procedure, the worst case running time is $O(p)$. The *Local Repair* procedure operates on the logical sensors L assigned to a single physical sensor, and considers the different physical sensors that the sensors L could use. In a pathological case (where all logical sensors are assigned to one physical sensor, and all alternates are occupied), this procedure can take $O(l \times p)$ time. Finally, the *Global Repair* procedure traverses multiple physical sensors, but never visits one physical sensor more than once (by keeping track of what sensors it has already traversed). At worst, it will traverse the list of logical sensors on each physical sensor, resulting again in a running time of $O(l \times p)$, but with a larger constant term since each logical sensor may be examined multiple times. Opportunistic repair for MCH+ simply traverses each logical sensor and tests the alternates for each, requiring $O(l \times p)$ time.

Unfortunately, we do not yet have a formal specification of the problem complexity. Given that any particular logical sensor can be matched with any combination of physical sensors, this is not trivially a bipartite matching or network flow problem. It appears that the sensor fusion aspect of the assignments produces a combinatorial complexity (as in $O(l \times p!)$), and sensor sharing would increase this further to $O(l! \times p!)$. However, we are not prepared to present the actual complexity.

G. Suitability of Algorithm

The Min-Conflict with Happiness algorithm as presented above meets the criteria for scheduling of sensing resources described in Section I as follows.

- The scheduler does not change or control the order of execution of behaviors. It allows the behaviors to become active through some external mechanism, attempts to fit the new demands for sensing into the existing schedule, making repairs as needed. No relaxation of the plan is performed.
- The scheduler relies upon the perceptual schema (a logical sensor) to construct a preference ordering over its possible sensing strategies. This preference ordering (which can also be derived based on the utility of each possible assignment) is then used to assign the most preferred available physical sensor or sensors.
- The scheduler assumes that the agents are uncooperative. That is, no negotiation takes place between behaviors. In fact, the behaviors do not have knowledge of each other. The scheduler allows the behaviors to make conflicting requests, which it then repairs.
- The Min-Conflict with Happiness algorithm is capable of repairing a broken schedule (in which sensors are assigned in conflict), which enables it to find solutions even when there is high contention.
- This algorithm first attempts to perform assignments in a *greedy* fashion, which involves a linear search across the physical sensors (a strategy that has been shown to work about 85% of the time in simulation). If this approach fails, the algorithm resorts to more exhaustive searches only when necessary. This allows the algorithm to function quickly, which makes it appropriate when there is limited time to devote to scheduling.
- The scheduler does not require any knowledge of future sensing needs. All requests are handled at the moment a request for sensing is made, based only on the current state of the sensors.
- The scheduler does not refuse to provide resources to a behavior in order to find a solution. That is, no relaxation of constraints is performed by the Min-Conflict with Happiness algorithm.

IV. Simulation Experiments

This section describes tests of the Min-Conflict with Happiness heuristic algorithm in simulation that show a 63% improvement over a *greedy* scheduler and a 142% improvement over a *random* scheduler (detailed in Section IV-A) in terms of successful assignments, and an improvement in overall utility using the Opportunistic Repair step (MCH+) of 8.88% over MCH on average (with a maximum of 140%), 7.01% over *greedy*, and 24.63% over *random* on average.[1] Other tests validated support for sensor fusion.

The first test of MCH was performed in simulation with single-sensor assignments (without sensor fusion), and the second test was performed with sensor fusion. The purpose of these tests was to determine how MCH performed compared to a *greedy* algorithm and a *random* scheduler. 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 heuristics to be compared on a robot and the time needed to run a robot through enough tasks to adequately test the heuristics 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.

[1]These values differ from those cited in [26] and [27] due to a tallying error.

TABLE I
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

TABLE II
NUMBER OF ASSIGNMENTS COMPLETED IN SIMULATION BY EACH ALGORITHM. MAXIMUM POSSIBLE WOULD BE 200, 20 FOR EACH OF 10 RUNS

Method	1	2	3	4	5	6
MCH	16	20	20	7	9	20
greedy	3	18	4	7	8	3
random	4	9	10	5	8	3

	7	8	9	10	Total
MCH	17	12	20	19	160
greedy	13	4	19	19	98
random	7	9	4	7	66

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

A. Simulation Setup

Two tests of MCH were conducted in simulation. The first measured the performance of MCH for cases without sensor fusion. The second set of tests measured the performance when allocating for sensor fusion, and the discussion of those tests will follow the cases without sensor fusion.

The purpose of the algorithm is to satisfy requests for physical sensors by logical sensors, so for the first test, a set of (fictional) heterogeneous logical sensors was fabricated (shown in Table I). The set of alternates for each logical sensor was unique, such that the allocation problem was nontrivial, 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 and corresponding acceptable physical sensors were chosen based on what a robot might be expected to do. 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 I. 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.

B. Simulation Results

The results of this experiment show that MCH was able to satisfy a significantly greater number of requests than the other methods before failing, and to maintain a higher level of utility. The *length of successful assignments* metric will be discussed first, followed by a discussion of the *level of utility* metric. Compared to the *greedy* allocation, MCH handled 63% more requests on average (with a statistical significance of $p = 0.0317$, given a null hypothesis that MCH averages the same number of successful allocations as *greedy*), and compared to the *random* allocation, MCH handled 142% more requests (with a statistical significance of $p = 4.08 \times 10^{-5}$ given the analogous null hypothesis). The actual number of requests satisfied in each case are shown in Table II. In each case where MCH/MCH+ failed to allocate for a new request, no solution could be found by hand, so a *prescient* algorithm would have done no better.

The data are also shown graphically in Fig. 11, 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 Fig. 11 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.

In terms of the utility achieved across all assignments, the Min-Conflict with Happiness algorithm also proved successful. In all cases except for one, MCH and MCH+ matched or

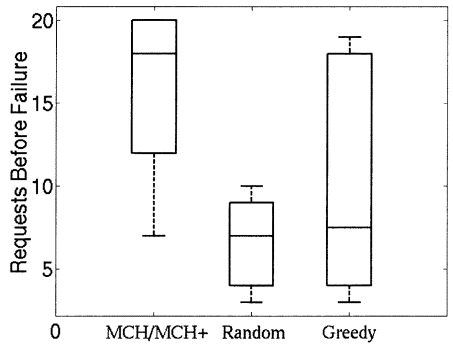

Fig. 11. Box-whisker plot of number of requests handled before failure by each algorithm. The error bars (whiskers) indicate that MCH/MCH+ allocated for as few as seven requests before failing, and in that particular case, *random* and *greedy* allocated for five and seven, respectively.

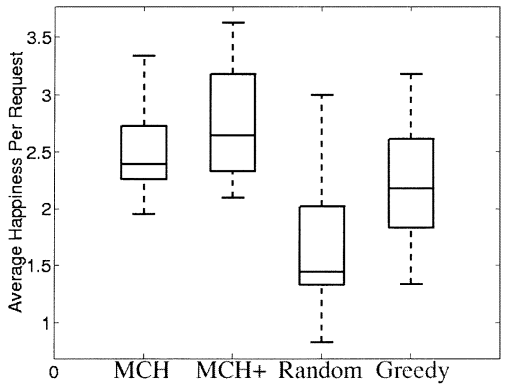

Fig. 12. Box-whisker plot of happiness before failure. Higher happiness is preferable. The error bars (whiskers) indicate that at their worst, MCH and MCH+ perform as well as the other heuristics perform on average.

TABLE III
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
MCH+	1	2	1	2	2	2.66	2.66
MCH	1	2	1	2	2	2.66	2.16
random	0.5	1.5	0.5	1	2	2.33	1.83
greedy	1	2	1	2	-	-	-

Request	8	9	10	11	12	13
MCH+	3.66	2.66	3.16	2.66	3.66	-
MCH	3.16	2.66	3.16	2.66	3.66	-
random	2.33	1.83	-	-	-	-
greedy	-	-	-	-	-	-

TABLE IV
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

outperformed *greedy* and *random* assignment at every step. In the exceptional case, the random approach managed a slightly higher utility than MCH and MCH+ for a single assignment by making a suboptimal assignment previously. The logical sensor for *Follow-Hall* could either use vision (utility 1), sonar (utility 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 and MCH+ use as the 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+ (which uses the *opportunistic repair* step) provided further improvement above MCH in terms of overall utility, reaching as high as 140% improvement in global utility as resources became available (that is, in one case, MCH+ 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 III, where the sequence of activations causes the global utility to vary according to the different heuristics. On average, MCH+ provided 8.88% greater global happiness than MCH, 7.01% over *greedy*, and 24.63% improvement over the *random* approach. These values are reflected in Fig. 12.

However, while MCH+ provides an 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. Further discussion of this issue can be found in Section VI.

The time taken to handle each event was between 11 and 17 ms on an Intel Pentium 233, with higher resource contention requiring the longer solution times.

The ability of the algorithm to perform assignments including fused sensors was tested in the second set of experiments, again using the *greedy* and *random* schedulers for comparison. These tests used the logical and physical sensors shown in Table IV, and requests consisting of all 24 permutations of those logical sensors were tested. The same metrics (average number of requests handled and average utility) were used, and the results are shown in Table V. As expected, MCH outperformed the *greedy* and *random* schedulers when allocating with fusion.

V. ROBOT RESULTS

Once the MCH heuristic 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 (winning a technical achievement award in 1999, and

TABLE V
TEST RESULTS FOR MCH AND OTHERS FOR SENSOR FUSION, USING 24 PERMUTATIONS OF POSSIBLE REQUESTS. THE THIRD 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 IV

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

third place and the Nils Nilsson award for technical merit in 2000), and was demonstrated at the Tampa Museum of Science and Industry (MOSI). This section describes the robots used and the tasks they performed, as well as performance data for the algorithm. The algorithm performed favorably in these tests; at MOSI, MCH handled 71 changes over 40 min (an average rate of 106.5 changes per hour), requiring approximately 14 ms each, or a total of approximately 0.99 s of processing over 40 min. A discussion of the MOSI tests appears in Section V-B.

A. Implementation

Two Nomad200 robots, Butler and Leguin, were used to compete in these competitions. Both robots are holonomic, capable of moving in any direction from a stop and are 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; Butler has two rings of sonar, whereas Leguin only has one, and Butler uses a SICK planar laser ranger. Butler may also be outfitted with a thermal sensor.

MCH was incorporated into a prototype of the SFX architecture, which includes a *Sensing Manager* and an *Effector Manager* to schedule the robot's resources. This integration was accomplished by adding a socket interface between the LISP code and the *Sensing Manager* process, such that the *Sensing Manager* would receive requests and pass them 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).

B. Waiter Test Domain

Tests of the MCH algorithm in a robotic task came in the form of the annual "Hors d'oeuvres, anyone?" competition of the American Association for Artificial Intelligence (AAAI) which requires the robot entrants to serve appetizers to guests of the conference. Both Nomad robots were entered in the 1999 and 2000 competitions and used a similar strategy each year. As Butler had a larger variety of sensors available, she was designated the primary waiter, and Leguin was tasked to bring a refill tray of appetizers from the refill station to Butler when needed. We did not collect quantitative data during the AAAI competitions so a demonstration of the robots was held at the Tampa Museum of Science and Industry (MOSI) to collect data. These data correspond to Butler, because Butler was equipped to simulate sensor failure, and because Butler required more changes of sensing state than Leguin.

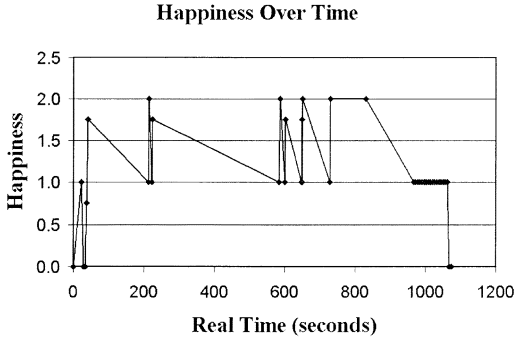

Fig. 13. Plot of the level of happiness at each request during a typical run at MOSI. The maximum possible in that configuration was 2.0; the vertical axis shows 2.5 to set an appropriate scale.

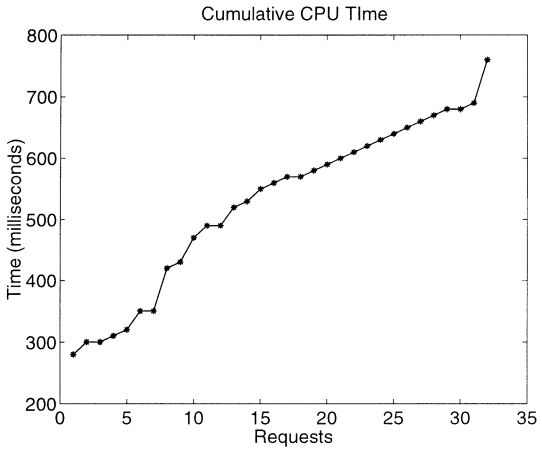

Fig. 14. Plot of the cumulative CPU time used throughout a typical run at MOSI. The vertical axis is time in milliseconds, while the number of requests is shown on the horizontal axis.

In the MOSI tests, the robots were run for 40 min (limited by their battery life). In this time, MCH received 71 total messages consisting of 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 simulated using a panel of switches regulating power to the robot's hardware). Nine messages indicated a sensor failure and 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. Over four runs, the average happiness ranged from 0.57 to 1.44 out of a maximum possible 2.0 (see Fig. 13).

In this test, the *Global Repair* stage was never needed as the combination of behaviors and sensors did not produce much contention. Over the 40 minute run, the MCH implementation in LISP required a total of 990 ms (meaning that on average, handling each message required approximately 14 ms, 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 ms, which is a one-time cost. The CPU time for a representative run is plotted in Fig. 14.

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.

VI. Discussion

From the experiments discussed in Section IV, V, we claim that the Min-Conflict with Happiness heuristic algorithm (MCH) is viable for a robotic task, and that it can handle numerous requests for sensing given incomplete information, while also dealing with unpredictable events and sensor failure. MCH satisfies these requests for a number of different perceptual processes that compete for resources, without complete information, which distinguishes it from other known approaches. Interesting aspects of the algorithm are that it is simple and efficient, operating in $O(l \times p)$ time (see Section III-F); event-driven, running only when changes to the sensing state are needed; it exhibits the property of *least-disturbance* of existing assignments to avoid overhead caused by changing the sensing strategy; it optimizes sensor certainty over all behaviors through its use of sensor utility; and it can allocate for sensor fusion. MCH also biases its search based on the *flexibility* of logical sensors, and tests computationally inexpensive solutions before attempting more exhaustive means. Tests in simulation show that MCH outperforms obvious schedulers in two metrics: length of successful sequences of requests, and overall utility while satisfying requests. In particular, MCH satisfied 63% more requests than *greedy* and 142% more than *random*, while maintaining higher utility. The time taken for each assignment was less than 20 ms, which indicates that this approach is valid for an on-line or real-time 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 handle sensor allocation, including sensor fusion and sensor failure. The real-world tests required that MCH perform 71 assignments over 40 min, which were performed in 0.99 s on a processor that was concurrently performing other robotic tasks, including computer vision.

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 to test sensor fusion when there is more contention (and thus, not always a solution) may be illustrative (as there was always a solution in the sensor fusion simulation). However, the existing simulation results should serve to verify the algorithm's correctness. Finally, 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 and MOSI, it appears that the underlying basis is sound.

An issue that has been introduced but not explored is whether it is appropriate to use opportunistic repair as part of the Min-Conflict with Happiness scheduler. On one hand, performing opportunistic repair violates the event-driven nature of the scheduler, and may also violate the principle of *least disturbance* of the schedule by breaking existing assignments for the sake of optimization. On the other hand, opportunistic repair does offer an improvement in the overall "happiness" of the system (as much as 142%), which translates to a more certain state of sensing, and may lead to higher utilization of the robot's physical sensors. Whether opportunistic repair is appropriate and the question of when to perform this repair will depend on the particular domain in which a robot is operating. A robot using Min-Conflict with Happiness may choose to perform opportunistic repair if it has Level I information (that is, it can predict both the sequence and duration of future requests) and thus knows when the next request will take place. The robot can ensure that the overhead associated with opportunistic repair is merited, meaning that enough time will pass before the next request that the improved sensing state is worth the overhead. The robot can also perform opportunistic repair if its current state of sensing proves inadequate (meaning that all logical sensors are satisfied, so MCH has done its job, but the robot may be unable to perceive precisely enough for a task).

Another issue that has been mentioned is that of sensor utility. For this work, two utility measures were used in order to sort alternates for MCH. One measure was qualitative, and was based simply on the number of alternates that a logical sensor had, such that the logical sensor's first choice would be given a utility of 1.0, and subsequent choices would be given a lower utility (depending on the total number of alternates). This was an ad hoc method, and relied on the programmer to choose an appropriate preference order for the alternates of each logical sensor. A quantitative method was to combine the measured true-positive and true-negative rates from each of a logical sensor's alternates, and to combine these values using a *t*-norm function [28], [29]; in this case, the algebraic product. Other utility metrics, such as those mentioned in Section II (especially the *reward* used in [15]) are compatible with our approach. While *t*-norms were satisfactory for expressing the expected performance of a logical sensor in terms of false positives and false negatives, any utility measure that can be used to sort sensing strategies according to certainty will work with MCH, with no change to the algorithm.

It is interesting to consider Fig. 15 and the contribution of each step of the MCH algorithm. *Greedy Allocation* is an effective method for making assignments quickly and without disturbing previous assignments, handling 88.7% of the requests without generating conflicts. *Local Repair* can fix many of the cases where *Greedy Allocation* fails (an additional 9.4%), but it does so by disturbing existing assignments (starting with the most *flexible*). However, this disturbance is limited to the logical sensors assigned in conflict to a single physical sensor. *Global Repair* was only called to satisfy 1.9% of the allocation requests in simulation, and took 17 ms of processor time on average (on the robot's Pentium 233). It builds on *Local Repair*, but does so

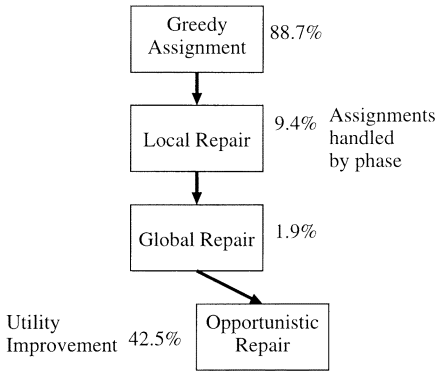

Fig. 15. Steps of the MCH Algorithm. To the right is the fraction of cases that each step successfully solved in simulation. To the left is the experimental gain in utility provided by MCH+.

in a recursive manner, and has the potential to disturb any prior assignment in the schedule.

VII. Conclusions and Future Work

This article contributes a simple heuristic algorithm for allocating sensing resources in a hybrid deliberative/reactive architecture and reports experimental and empirical results on its effectiveness under Level III of incomplete information (that is, no prediction of the sequence or duration of active behaviors). Existing resource allocation schemes used in robotics, most notably [13], do not work with incomplete information, making this approach unique. This article also contributes a categorization of the types of contention over sensing resources and a taxonomy of available information for the sensor scheduling task.

Some classical scheduling algorithms, such as Min-Conflict, do not optimize over the entire set of activities. The Min-Conflict with Happiness (MCH) algorithm was shown to provide this optimization of sensing quality for all active behaviors using a utility measure, both in simulation and on robots. Furthermore, MCH does not impose any restriction on the utility metric except that each logical sensor have a single metric. In this work, both a qualitative relative ranking and a quantitative expected performance measure based on t-norms were used with no change to the algorithm. In robot implementations, MCH used measures of the sensor uncertainty to allocate sensing resources which maximized the overall sensing quality.

The MCH algorithm is appropriate for autonomous mobile robots operating under a hybrid deliberative/reactive architecture. It makes no assumptions contrary to the characteristics in Section I. Also, it operates sufficiently fast to support real-time control activities of the robot (11–17 ms using a Pentium 233, which was simultaneously performing other tasks, such as computer vision). The algorithm is event-driven and operates under the principle of *least disturbance*, thereby avoiding thrashing. It is consistent with schedulers in hybrid architectures, and does not relax behavioral assignments; it merely optimizes sensing allocation to satisfy those assignments.

The article has shown through the algorithm's design and implementation on two autonomous mobile robots with different sensor suites that it is portable and applicable to a wide variety of sensors. In this work, sonars, cameras, laser range finder, thermal, and contact sensors were used. Also, the algorithm was shown to work with behavioral sensor fusion, where the *face-find* logical sensor generated percepts extracted from a camera fused with a thermal probe.

It should be noted that the current algorithm does not cover all categories of sensor contention and is restricted only to Level III type of incomplete information. The algorithm does not cover sensor sharing; it operates only with the second type of sensor contention, in which multiple perceptual schemas compete for the same perceptual resource (however, through a modification to the *CONFLICT-EXISTS* function, the algorithm could allow sharing of sensors). Additional information about the future does not obviate this method, as it can be used alongside a technique such as that used in [13].

Directions for future work include adapting MCH to function in distributed robot systems where one robot may request and make use of sensors belonging to another robot; developing a mechanism to determine whether two or more logical sensors may share a physical sensor without conflict; and finding a sensor utility metric that spans all sensing modalities.

Acknowledgment

The authors would like to thank the anonymous reviewers, and S. Patel for proofreading.

References

[1] R. Murphy, *Introduction to AI Robotics*. Cambridge, MA: MIT Press, 2000.
[2] M. Arbib, "Perceptual structures and distributed motor control," in *Handbook of Physiology—The Nervous System II*, E. Brooks, Ed. Bethesda, MD: Amer. Physiol. Soc., 1981, pp. 1449–1465.
[3] T. Henderson and E. Shilcrat, "Logical sensor systems," *J. Robot. Syst.*, vol. 1, no. 2, pp. 169–193, 1984.
[4] L. P. Kaebling, "An architecture for intelligent reactive systems," in *Reasoning About Actions and Plans*, M. P. Georgeff and A. L. Lansky, Eds. San Francisco, CA: Morgan Kaufmann, 1987, pp. 395–410.
[5] R. Simmons, L. Lin, and C. Fedor, "Autonomous task control for mobile robots," in *Proc. the 5th IEEE Int. Symp. Intelligent Control*, 1990, pp. 663–668.
[6] S. Minton, M. Johnston, and P. Laird, "Solving large scale constraint satisfaction and scheduling problems using a heuristic repair method," in *AAAI Proc.*, Menlo Park, CA, 1990, pp. 17–24.
[7] S. Minton, M. Johnston, A. Philips, and P. Laird, "Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems," *Artif. Intell.*, vol. 58, no. 1–3, pp. 161–205, 1992.
[8] M. Johnston and S. Minton, *Intelligent Scheduling*. San Francisco, CA: Morgan Kaufmann, 1994.
[9] M. Fox, *Intelligent Scheduling*. San Francisco, CA: Morgan Kaufmann, 1994, pp. 3–28.
[10] R. Korf, "Real-time heuristic search," *Artif. Intell.*, vol. 42, no. 2–3, pp. 189–211, 1990.
[11] S. Matsubara and T. Ishida, "Real-time planning by interleaving real-time search with subgoaling," in *Proc. 2nd Int. Conf. Artificial Intelligence Planning Systems*, 1994, pp. 122–127.
[12] J. Pemberton and R. Korf, "Incremental search algorithms for real-time decision making," in *Proc. 2nd Int. Conf. Artificial Intelligence Planning Systems*, 1994, pp. 140–145.
[13] E. Ephrati, M. Pollack, and J. Rosenschein, "A tractable heuristic that maximizes global utility through local plan combination," in *ICMAS-95 Proc.*, V. Lesser, Ed., 1995, pp. 94–101.
[14] M. Youssefmir and B. Huberman, "Resource contention in multiagent systems," in *ICMAS-95 Proc.*, V. Lesser, Ed., 1995, pp. 398–403.
[15] G. Hovland and B. McCarragher, "Dynamic sensor selection for robotic systems," in *Proc. 1997 IEEE Int. Conf. Robotics and Automation*, vol. 1, 1997, pp. 272–277.

[16] J. Evans and V. Krishnamurthy, "Optimal sensor scheduling for hidden Markov model state estimation," *Int. J. Contr.*, vol. 74, no. 18, pp. 1737–1742, 2001.
[17] V. Krishnamurthy, "Algorithms for optimal scheduling and management of hidden markov model sensors," *IEEE Trans. Signal Processing*, vol. 50, pp. 1382–1397, June 2002.
[18] A. Lim and V. Krishnamurthy, "Risk-sensitive sensor scheduling for discrete-time nonlinear systems," in *Proc. 37th IEEE Conf. Decision and Control*, 1998, pp. 1859–1864.
[19] A. Savkin, R. J. Evans, and E. Skafidas, "The problem of optimal robust sensor scheduling," in *Proc. 39th IEEE Conf. Decision and Control*, 2000, pp. 3791–3796.
[20] H. Lee, K. Teo, and A. Lim, "Sensor scheduling in continuous time," *Automatica*, vol. 37, no. 12, pp. 2017–2023, 2001.
[21] C. Chen and M. Trivedi, "Task planning and action coordination in integrated sensor-based robots," *IEEE Trans. Syst., Man, Cybern.*, vol. 25, no. 4, pp. 569–591, 1995.
[22] F. Noreils and R. Chatila, "Plan execution monitoring and control architecture for mobile robots," *IEEE Trans. Robot. Automat.*, vol. 11, no. 2, pp. 255–266, 1995.
[23] R. Murphy and A. Mali, "Lessons learned in integrating sensing into autonomous mobile robot architectures," *J. Exper. Theoret. Artif. Intell.*, vol. 9, no. 2, pp. 191–209, 1997.
[24] D. Musliner, E. Durfee, and K. Shin, "Circa: a cooperative intelligent real-time control architecture," *IEEE Trans. Syst., Man, Cybern.*, vol. 23, no. 6, pp. 1561–1573, 1993.
[25] B. C. M. Carlsson and G. Ottosson, "An open-ended finite domain constraint solver," in *Proc. Programming Languages: Implementations, Logics, and Programs*, 1997.
[26] A. Gage and R. Murphy, "Allocating sensor resources to multiple behaviors," in *Proc. IROS 99*, 1999.
[27] ——, "Sensor allocation for behavioral sensor fusion using min-conflict with happiness," in *Proc. IROS 2000*, 2000.
[28] K. Menger, "Statistical metrics," in *Proc. Nat. Acad. Sci.*, vol. 28, Dec. 15, 1942, pp. 535–537.
[29] G. Klir and B. Yuan, *Fuzzy Sets and Fuzzy Logic: Theory and Applications*. Upper Saddle River, NJ: Prentice-Hall, 1997.

Aaron Gage is pursuing the Ph.D.degree in computer science and engineering at the University of South Florida, Tampa. He received the B.S. degree in mathematical and computer sciences from the Colorado School of Mines in 1997 and a M.S. degree in computer science and engineering from the University of South Florida in 2001.

His research interests include mobile robots, sensing, artificial intelligence, fault tolerance, cryptography, and computer graphics.

Robin Roberson Murphy received the B.M.E. degree in mechanical engineering and the M.S. and Ph.D degrees in computer science in 1980, 1989, and 1992, respectively, from Georgia Tech, Atlanta, where she was a Rockwell International Doctoral Fellow. Her Ph.D. dissertation pioneered cognitive approaches to robust sensor fusion.

Since 1998, she has been an Associate Professor with the Computer Science and Engineering Department at the University of South Florida, Tampa, with a joint appointment in cognitive and neural sciences in the Department of Psychology. Her research interests include teams of heterogeneous multiple robots, sensor fusion, and human–robot interaction. She is the author of over 70 publications in these areas as well as the textbook *Introduction to AI Robotics*. She is Director of the Center for Robot-Assisted Search and Rescue at the University of South Florida.

Dr. Murphy is a recipient of an NIUSR Eagle Award for her participation at the World Trade Center.