Proceedings of ICAR 2003
The 11th International Conference on Advanced Robotics
Coimbra, Portugal, June 30 - July 3, 2003

Fault-Tolerant Certainty Grid

W. Elmenreich

Vienna University of Technology
Institut für Technische Informatik
Treitlstraße 1-3/182-1
wil@vmars.tuwien.ac.at

Abstract

World modelling for mobile autonomous robot is usually a process that uses sensor data as input and provides a model of the robot's environment as output. In this paper we investigate on sensor fusion methods for robustness and fault tolerance. We evaluate three methods according to their performance, memory consumption, and required sensor configurations.

The algorithms have been implemented in a four-wheeled autonomous mobile robot that uses a set of three infrared sensors to build the world model. We present a performance analysis of the new algorithms based on simulation and experimental data.

1 Introduction

Autonomous mobile robots belong to a class of applications whose operability depends on sensor data. Sensor, as physical devices converting a physical property into a measurement that can be interpreted by a computer system, are affected by several sources of error, like sensor deprivation, limited spatial or temporal coverage, imprecision, cross-sensitivity, and uncertainty. Thus, dependable applications may never depend on a single sensor. In order to overcome these problems, the inputs from several sensors are combined to form a dependable representation of the environment, the *world model*.

For mobile robots, usually the world model is represented as a two-dimensional or, especially for non-flat outdoor environments, a three-dimensional map of the robots surrounding. Typical approaches to generate such a map are the grid-like division of the environment, like the occupancy grid approach of Elfes [2]. Based on this grid, the navigation and path planning decides on the robot's actions. Erroneous or ambiguous sensor readings are detected and solved by processing redundant sensor information. Redundant information can be evaluated at different levels of abstraction. In general, if the evaluation is performed at sensor level, the system complexity can be kept low at the cost of hardware expenses. If evaluation of redundant information is made at application level, the performance is better, since at application level has more knowledge about the reasonableness of a particular result. However, system complexity increases since normal processing functions become intertwined with error-detection and fault-tolerance functions [7].

It is the objective of this paper to investigate on sensor fusion methods for robustness and fault tolerance for a grid-like representation of a mobile robot's world model.

The remainder of the paper is organized as follows: Section 2 describes the system architecture of a sensor grid application and examines the possibilities and benefits of applying sensor fusion at particular levels in this model. Section 3 presents certainty grid algorithms that can handle faulty measurements. Section 4 presents the results from the evaluation, while Section 5 discusses the results. Section 6 concludes the paper.

2 Architectural Considerations

An autonomous robotic system contains at least a set of sensors and actuators and a control application. Sensors and actuators are the interface to the process environment and belong to the *transducer level* of the robotic system, while the control application belongs to the *control level*.

In general, a system is composed out of components, whereas components are subsystems like sensors, actuators, processing nodes, and communication channels. As a matter of fact, every single component of a system will eventually fail [1]. Requirements for highly dependable systems can only be met, if these failures are taken into account.

Some systems, fail in a manner that they still provide a service, however at a degraded level. For example, a sensor may be affected by cross-sensitivity and fail to render a measurement with the specified accuracy – however, the measurement contains still information that can be exploited.

In the following sections we focus on sensor fusion

methods in order to exploit the sensors' information under the presence of faults. Failures of microcontrollers and communication lines will be handled by fault-tolerant mechanisms that are beyond the scope of this paper.

A dependable system needs a redundant sensor configuration for the purpose of competitive sensor fusion. Competitive sensor fusion can be used to achieve *robustness* or *fault tolerance*. Robustness means that effects of single sensor faults are attenuated in the result. Fault tolerance means that a defined set of faults does not affect the result at all, thus faults are masked out. The defined set of faults is called the *fault hypothesis*. Fault-tolerant systems do not make any guaranty about their behavior when faults occur that are not defined in the fault hypothesis.

For the implementation of the necessary tasks to handle faults, the designer of a system has two main options:

Transducer level: Each transducer can be equipped with a set of redundant sensors and a voting mechanism. Voting can be seen as the simplest method of sensor fusion. The advantage of this approach is that the other parts of the application are not aware of the extra sensors and faults of single sensors are masked out. The disadvantage is the great amount of extra hardware, which results in increased cost, weight, and power consumption, and possible problems of mutual sensor inference and, since the sensors are geometrically not exactly at the same place, parallax errors.

Application level: The application-specific approach uses reasonableness checks that use application knowledge to judge whether a value is correct or not. Since the dependability operations are integrated with (parts of) the application, this approach leads to increased design effort and application complexity.

However, this approach can be more efficient, since dependable behavior could be achieved with less hardware expenses [11].

In order to achieve a high efficiency and keep the system complexity low, we propose a compromise that performs the sensor fusion at an intermediate level between transducer and application level, the fusion/dissemination level. Thus we propose a system model that decomposes the real-time computer system of a mobile robot into three levels [4]: First, a transducer level, containing the sensors and the actuators, second, a fusion/dissemination level that gathers measurements, performs sensor fusion respectively distributes control information to the actuators, and third, a control level where a control program makes control decisions based on environmental information provided by the fusion level. The interface between fusion/dissemination level and control level can be made transparent to the transducer configuration, such that the control application can be implemented independent of the employed sensors and actuators [5].

3 Robust World Modelling

We assume a set of sensors that measure the distance to the next obstacle at particular angles from the robot. The goal is to get a map of the robot's environment containing obstacles and free space.

The sensors are swept around by a motor for each sensor in order to cover a segment of the robot's surroundings over time. The segments overlap partially or fully, hence providing some redundancy in the coverage of the environment. However, although our architecture is capable of synchronizing all sensors and motors, it is not feasible to turn any two sensors into the same or at least approximately the same direction because of interference problems. Thus, it is impossible to directly compare sensor readings made at the same time.

Furthermore, from the view of hardware architecture it is almost impossible to mount sensors in a way that the viewpoint angle from a sensor to an object is perfectly identical to the angle of the replicated sensor. A replicated sensor will thus always be located slightly offside, thus viewing objects from different angles. Another problem is that often measurements may not be synchronized due to sensor inference problems. Even if two sensors are working correctly, they may produce different results. Figure 1 depicts an example for an object that yields ambiguous sensor readings. Although both sensors are working according to their specification, sensor B detects an object for the given region while sensor A does not.

Besides these problems, we assume that a sensor may degrade the quality of its service up to the case where it permanently delivers faulty measurements. For example, one distance sensor could refuse to detect any object and always reports "no object nearby".

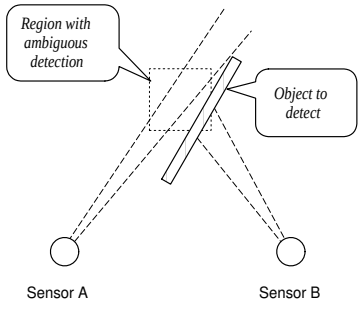


Figure 1: *Discrepancy between sensor A and sensor B due to object shape*

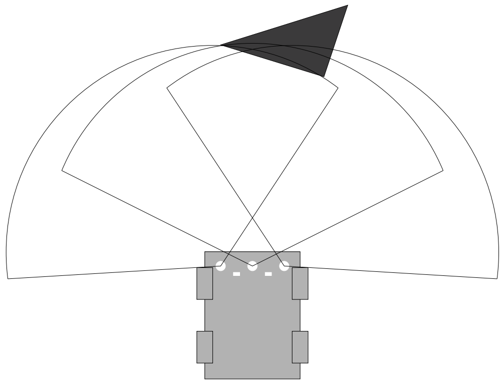
Note, that the existing certainty grid method based on Bayesian fusion can only handle the effects of occasional sensor faults on the grid [9]. Permanent faults – as assumed in our fault hypothesis – would result in a significant deviation of the representation in the grid from the actual environment.

For the purpose of handling sensor failures where a sensor permanently submits measurements with incorrect values, we have derived a robust certainty grid algorithm (also presented in [6]) and two fault-tolerant algorithms for grid generation.

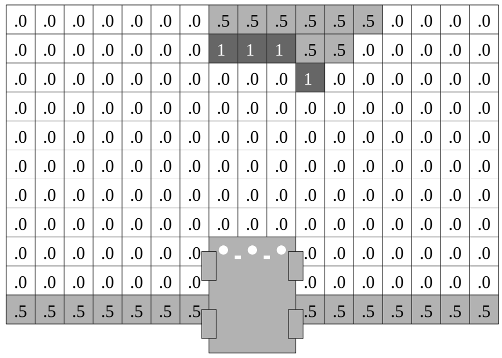
The problem is solved by analyzing the redundant parts of the certainty grid. The certainty grid is a two-dimensional array of grid cells that corresponds to the robot's environment. Each grid cell contains a probabilistic value *occ* ranging from 0 to 1 corresponding to the believe that this cell is occupied by an object:

$$cell.occ = \begin{cases} 0 & \text{free} \\ \vdots & \\ 0.5 & \text{uncertain} \\ \vdots & \\ 1 & \text{occupied} \end{cases}$$

Figure 2 gives an example for a certainty grid. Figure 2(a) depicts a mobile robot and an obstacle, while Figure 2(b) depicts the certainty grid. The values in the grid cells are the occupancy values. For regions that are not seen by a sensor, the resulting occupancy value is 0.5, that is unknown territory.



(a) Obstacle in front of robot



(b) Representation in certainty grid

Figure 2: *Representation of the robot environment in a certainty grid*

3.1 Fault-Tolerant Certainty Grid

The fault-tolerant certainty grid uses a separate entry of an occupancy value for each sensor. Thus, each sensor produces its own grid independently of the other sensors. The concise world model is built by fusing all sensor grids. The proposed fault-tolerance algorithm is similar to the Fault-Tolerant Average algorithm for clock synchronization [8].

The fault-tolerant fusion is performed as follows: First the measurements from all sensors for each grid cell are gathered. If the sensors do not update the grid cells simultaneously, the gathering of the measurement will take some time. It is assumed that the environment does not change significantly during this time.

Then the set of proposed certainty values for each grid cell are sorted and the t lowest and t largest values are removed from the set. t represents the maximum expected number of faulty sensors per measurement. If the set contains at least t faulty measurements, these will either be removed from the set, or the value of the faulty measurements lie between two correct measurements, which will not worsen the result.

The remaining measurements are then fused by Bayesian fusion. Assuming *conditional independence* and *maximum entropy* [10], the fusion formula for n values can be expressed as follows:

$$\frac{1}{\mathbb{P}(cell.occ|S_1,\ldots,S_n)} - 1 = \prod_{i=1}^{n} \left(\frac{1}{\mathbb{P}(cell.occ|S_i)} - 1 \right)$$

There must be at least $2t+1$ sensors contributing to each grid cell in order to provide enough data to keep at least one measurement after removing the t extreme values.

The algorithm tolerates at least t faulty measurements at a time, however drops also many correct measurements. This results in an information loss in the fused result, which may degrade the result in the average case.

3.2 Fault-Tolerant Median Selection

If the number of expected faults should be maximized, we propose the implementation of fault-tolerance by selecting the median value from the set of measurements for each grid cell. Thus, if a cell is updated by n sensors, at least $\lfloor \frac{n-1}{2} \rfloor$ faulty measure-

ments are tolerated. The median method drops even more correct measurements than the fault-tolerant average algorithm, however has the big advantage of being adaptive to the number of measurements. Thus if the number of contributing grid cells is not constant for all grid cells, the median method is easily applied to this situation, while the fault-tolerant average algorithm is not.

3.3 Robust Certainty Grid

The robust certainty grid algorithm fuses the sensor inputs immediately into a single grid. Together with the grid occupancy value *occ* each cell stores the main contributor (e. g., the sensor that updated this cell most recently) of the *occ* value with the cell. This property of each cell is named the current *owner* of the cell:

$$cell.owner = \begin{cases} 0 & \text{unknown} \\ 1 & \text{sensor 1} \\ \vdots & \\ \mathrm{n_{sensors}} & \text{sensor n} \end{cases}$$

All grid cells are initialized with $cell.occ = 0.5$ and $cell.owner = unknown$. When a new measurement has to be added to the grid, the following *AddToGrid* algorithm is executed (Figure 3 lists the algorithm in pseudocode):

- If the particular grid cell has no contributor listed in its owner field or the cell owner is identical with the contributing sensor, the measurement of the sensor is taken as is and the cell stores the index of the sensor as new owner.

```
procedure AddToGrid( sensor, cell )
begin
  if (cell.owner = unknown) or (cell.owner = sensor) then
    cell.occ := sensor.measurement;
    cell.owner := sensor;
  else
    comparison := 4*(cell.occ-0.5)*(sensor.measurement-0.5);
    weight1 := abs(cell.occ-0.5)*cell.owner.conf;
    weight2 := abs(sensor.measurement-0.5)*sensor.conf;
    if weight1 = weight2 then
      cell.occ := (cell.occ+sensor.measurement) / 2;
    else
      cell.occ := (cell.occ*weight1+sensor.measurement*weight2)
                        / (weight1 + weight2);
    if comparison > CONFIRMATIONTHRESHOLD then
      inc(cell.owner.conf);
      inc(sensor.conf);
    if comparison < CONTRADICTIONTHRESHOLD then
      dec(cell.owner.conf);
      dec(sensor.conf);
    contribution := 4*(cell.occ-0.5)*(sensor.measurement-0.5);
    if contribution > CONTRIBUTIONTHRESHOLD then
      cell.owner := sensor;
    else
      cell.owner := unknown;
end
```

Figure 3: *Pseudocode of the AddToGrid algorithm*

- If there is a different contributor, the measurement is first compared to the cell value *cell.occ* by calculating a value named *comparison*. If *comparison* is above a particular *confirmation threshold*, we speak of a *confirmation* of cell value and new measurement. If *comparison* is below a particular *contradiction threshold*, we speak of a *contradiction* of cell value and new measurement. In case of a confirmation, the confidence values of the new sensor and the owner are both increased. In case of a contradiction, the confidence values of the new sensor and the owner are both decreased. If *comparison* is not significant, it does neither yield a confirmation nor a contradiction.

- The new occupancy value of the cell is calculated as a weighted average between old value and measurement. The weights are derived from the respective confidence values and the significance of the measurement. A measurement is more significant if it has a greater absolute distance to the *uncertain* state (0.5).

- Thereafter, a new owner is selected. Therefore, a value *contribution* is derived. This value is calculated the same way as the comparison value, but it uses the new *cell.occ* value.

- The *contribution* is a measurement of the consistency of the sensor measurement with the new *cell.occ* value. If the *contribution* is above a certain threshold, the contributing sensor becomes the new owner of the cell. Otherwise the *cell.owner* value is reset to *unknown*.

4 Evaluation

We used two different test environments to evaluate the proposed algorithms, a real mobile robot and a simulation environment.

We implemented the robust certainty grid in a mobile robot for demonstration purposes. The mobile robot comprises a model car ("smart car") equipped with a suit of pivoted distance sensors, two ultrasonic sensors pointing straight forward, an electric drive, and a steering unit (see Figure 4).

The certainty grid is built from the input of the Sharp GP2D02 infrared sensors. These type of sensors provide a rather narrow sensor beam, however it delivers measurements with a significant amount of error. The quality of the infrared sensor data has been analyzed and presented in [3]. Figure 5 depicts the measured distribution of sensor errors for a sensor sample.

The proposed algorithms have been evaluated versus Bayesian fusion in a simulation program that emulates an arbitrary number of sensor that are swept around in order to map a given artificial environment.

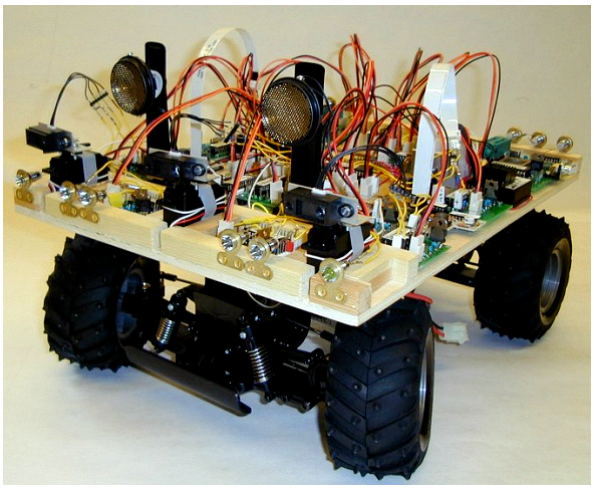


Figure 4: *Smart Car: Autonomous mobile robot with pivoting sensors*

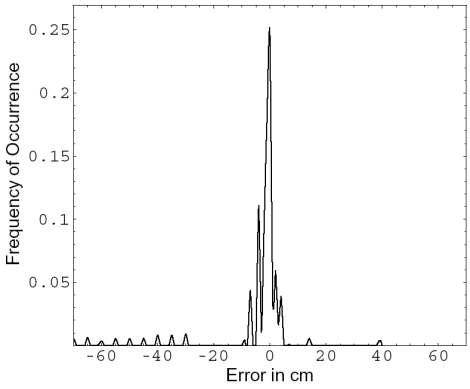


Figure 5: *Error histogram for Sharp GP2D02 infrared sensor (from [3]*

For each sensor we simulated the measured behavior of the sensor error of a real GP2D02 sensor. We have assumed that there is no correlation between any two measurements of different sensors.

In both test environments, we used 8-bit integer values to express the probability values between 0 and 1. Thus, a value of 0 corresponds to the $free$ state, 128 means the *uncertain* state while 255 is used to express the *occupied* state of a grid cell. The certainty grid had a size of 17 times 11 cells whereas each cell corresponds to a 10 cm times 10 cm square.

Figure 6 shows a comparison of the average results from the simulation. The grid has been generated several times using the proposed algorithms. The fault-tolerant median selection achieved the lowest deviation, i. e., best performance. All three proposed algorithms achieved a better performance than the standard Bayesian fusion.

The results obtained from the experiment with the smart car are depicted in Figure 7. Figure 7(a)

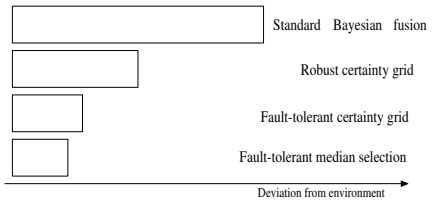


Figure 6: *Comparison of average deviation of generated grid from environment*

shows a photograph of the test environment, while Figures 7 b–d depict the certainty grids generated from the sensor data using different algorithms. Since the scanning ranges of the smart car's sensors do not exactly overlap, the fault-tolerant certainty grid algorithm algorithm was not tested with the smart car.

5 Discussion

All approaches need at least three sensors in order to compensate a single faulty measurement. In contrast to the fault-tolerant certainty grid algorithm, the robust certainty grid algorithm and the median selection gain extra sensor space, because the sensor views must overlap only partially.

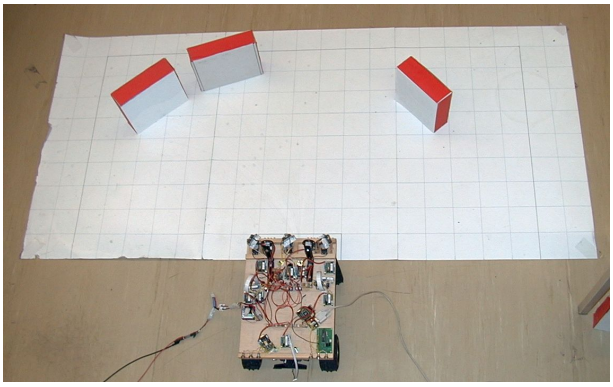
The fault-tolerant algorithms need to separately store the grid for each sensor (the resulting grid may produced on demand), thus require $n_{sensors} \cdot gridheight \cdot gridwidth$ memory elements where each element stores one certainty value. The robust certainty grid needs $gridheight \cdot gridwidth$ memory elements for the certainty values and $\frac{\lceil log_2(n_{sensors}+1) \rceil}{8} \cdot gridheight \cdot gridwidth$ extra bytes of memory for the storage for the owner values. The memory requirements for the confidence values can usually be neglected, since the number of sensors normally is remarkably lower than the total number of cells in the grid. Thus, the memory requirements of the robust certainty grid algorithm are considerable lower than the memory consumption of the fault-tolerant approach.

In contrast to Bayesian fusion and the fault-tolerant algorithms, the *AddToGrid* procedure of the robust certainty grid is sensitive to the ordering of measurements. Thus, when a grid cell is updated by subsequent measurements, the order of updates makes a difference in the result.

6 Conclusion

We have presented and evaluated two fault-tolerant and one robust method to handle faulty measurements when building a world model in form of a certainty grid map.

In the presence of faulty measurements, all methods show a better behavior than the Bayesian fusion.



(a) Experiment setup

0.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.0
0.0	0.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.0	0.0
0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.0	0.0	0.5
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.0	0.0
0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5
0.5	0.5	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.0	0.0	0.0	0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5

(b) Bayesian fusion

0.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.0
0.0	0.0	0.5	0.0	0.4	0.5	0.6	0.0	0.0	0.0	0.0	0.0	0.3	0.0	0.5	0.0	0.0
0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.0	0.0	0.5
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.5	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.0	0.0
0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5
0.5	0.5	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.0	0.0	0.0	0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5

(c) Robust certainty grid method

0.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.0
0.0	0.0	0.5	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.0	0.0
0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.0	0.0	0.5
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.5	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.0	0.0
0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5
0.5	0.5	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.0	0.0	0.0	0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5

(d) Fault-tolerant median method

Figure 7: *Comparison of algorithms on real data*

From the examined performance and the resource requirements, the robust certainty grid algorithm and the fault-tolerant median selection are most promising. While the fault-tolerant median selection shows the best performance, the robust certainty grid algorithm needs a significantly less amount of memory to store the certainty grid. However, if the certainty grid is only generated for a limited space while a second data structure is used as a general map, the resource requirements for the certainty grid will not be so stringent.

Acknowledgments

This work was supported by the Networks of Excellence European IST project ARTIST under contract No. IST-2001-34820.

References

[1] G. Bauer. *Transparent Fault Tolerance in a Time-Triggered Architecture*. PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2001.

[2] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *IEEE Computer*, 22(6):46–57, 1989.

[3] W. Elmenreich. *Sensor Fusion in Time-Triggered Systems*. PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2002.

[4] W. Elmenreich and S. Pitzek. The time-triggered sensor fusion model. In *Proceedings of the 5th IEEE International Conference on Intelligent Engineering Systems*, pages 297–300, Helsinki–Stockholm–Helsinki, Finland, September 2001.

[5] W. Elmenreich and S. Pitzek. Using sensor fusion in a time-triggered network. In *Proceedings of the 27th Annual Conference of the IEEE Industrial Electronics Society*, volume 1, pages 369–374, Denver, CO, USA, November–December 2001.

[6] W. Elmenreich, L. Schneider, and R. Kirner. A robust certainty grid algorithm for robotic vision. In *Proceedings of the 6th IEEE International Conference on Intelligent Engineering Systems (INES)*, pages 25–30, Opatija, Croatia, May 2002.

[7] H. Kopetz. *Real-Time Systems, Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, Boston, Dordrecht, London, 1997.

[8] J. Lundelius and N. Lynch. A new fault-tolerant algorithm for clock synchronization. In *Proceedings of the 3rd Annual ACM Symposium on Principles of Distributed Computing*, pages 75–88, Vancouver, Canada, August 1984.

[9] M. C. Martin and H. P. Moravec. Robot evidence grids. Technical Report CMU-RI-TR-96-06, The Robotics Institute, Carneghie Mellon University, Pittsburgh, PA, USA, 1996.

[10] B. Moshiri, M. R. Asharif, and R. Hosein Nezhad. Pseudo information measure: A new concept for extension of Bayesian fusion in robotic map building. *Information Fusion*, 3(1):51–68, 2002.

[11] S. Poledna. Fault tolerance in safety critical automotive applications: Cost of agreement as a limiting factor. In *Proceedings of the 25th IEEE International Symposium on Fault-Tolerant Computing*, pages 73–82, Pasadena, California, USA, June 1995.