MULTI-SENSOR (IR) DATA FUSION FOR MOBILE ROBOT NAVIGATION USING OCCUPANCY GRID METHOD

by

CAROL GAL

MAY 1994

A thesis submitted to the
School of Graduate Studies and Research
in partial fulfillment of the
requirements for the degree of
Master of Applied Sciences
in Electrical Engineering

OTTAWA-CARLETON INSTITUTE FOR ELECTRICAL ENGINEERING

Department of Electrical Engineering
Faculty of Engineering

University of Ottawa, Ontario, CANADA

ISBN 0-612-00461-9

Canada

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

I hereby declare that I am the sole author of this thesis.

I authorize the University of Ottawa to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Carol Gal

I further authorize the University of Ottawa to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Carol Gal

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First, I would like to thank Dr. Emil Petriu, my thesis supervisor, for his direction, guidance and advice. His ideas, knowledge, judgment, and experience have been leading me through my work. I was also fortunate having his support and advice.

Next, I wish to thank the collective lead by Dr. Sadia Elgazzar from National Research Council for providing excellent working conditions and access to mechanic and computer facilities. I would like to thank in particular Mr. Larry Korba and Mr. Doug Taylor for their advice, time and help.

Next, I would like to thank the Canadian Space Agency for the good collaboration in providing the necessary mechanical support and expertise.

Next, I would like to thank Mr. Niculaie Trif for his helpful opinions, companionship, and many long discussion we had together.

Finally, I would like to thank my family for support and understanding and especially my daughter for her patience during my work.

ABSTRACT

The main topic of the thesis is the **multi-sensor data fusion** in the context of **mobile robot navigation.** The work presented has been part of a continuous research done in the field of mobile robots. In that respect, a mobile robot platform with an onboard manipulation capability has been developed as an experimental platform for a multi-sensor system for teleautonomous applications in an unstructured environment. Different types of sensors have been provided to gather information about the environment: infrared (IR) range finders, vision, tactile, position. While vision and tactile sensors were approached by other related work, this thesis is essentially aimed to solve the following problems:

- *Mobile robot navigation*, in terms of electronic interface and computerized control with real-time range data acquisition for obstacle avoidance, using a GUI (Graphical User Interface)
- *Occupancy grid method implementation* using *Bayesian analysis* for the case of an IR ranging sensor
- *Unstructured environment mapping*, in the context of *multi-sensor data fusion* of mobile robot views taken from different given positions
- *Global map integration* with other types of sensory information (vision)

PART 1

INTRODUCTION

The thesis presents an approach to the problem of *mobile robot navigation in an unstructured environment using multisensor data fusion*, based on the Bayesian estimation fusion model. As a solution to this updating problem does not exist in closed form, an approximation algorithm to the optimal solution based on the probability occupancy grid method was chosen.

The multi-sensor data fusion problem is presented in Part Two: different approaches reported in literature for frameworks, control structures, world models, fusion methods. As a conclusion to this part, examples of multi-sensor based mobile robots are described.

Part Three first introduces the theoretical approach of Bayesian estimation fusion model. Based on a framework proposed by Hager ([11], [16]) it is presented how a world representation in terms of geometric parametric functions can be employed to solve the problem of task directed sensor fusion. The occupancy grid method is described next with application in robot mapping and navigation. The experimental results presented in paragraph 3.3 deal with the following problems:

- the mobile robot electronic/software interface *design* in order to control the robot movement and the real-time obstacle avoidance based on IR range sensors information fusion
- the IR sensors *calibration and modeling* as a prerequisite in the Bayesian estimation fusion model
- the occupancy grid method *implementation* for the given IR sensors, global map representation
- an *experimental scene set-up* as an example of global map integration with vision in a virtual-reality interactive environment representation

The thesis conclusions are presented in Part Four.

PART 2

MULTI-SENSOR DATA FUSION

Multi-sensor data fusion, as defined in [19], refers to the process of actual combination (or fusion) of different sources of sensory information into one representational format. This definition would also apply to the fusion of information from a single sensory device acquired over an extended time period. Multisensor integration, as a more general notion, refers to the synergistic use of the information provided by multiple sensory devices to assist in the accomplishment of a task by a system. Although the distinction between fusion and integration is not standard in the literature, it serves to separate the more general issues involved in the integration of multiple sensory devices at the system architecture and control level, from the more specific issues involving the actual fusion of sensory information.

This chapter presents different aspects of multisensor integration and fusion that are quite general in terms of their range of applicability:
- an overview of paradigms, frameworks, and control structures that have been proposed;
- the problem of world models;
- a survey of general multisensor fusion methods;

The chapter also details the critical role played by multisensor fusion in enabling mobile robots to operate in uncertain or unknown environments. A variety of proposed high-level representations for multisensory information are presented that are suitable for mobile robot control and navigation. The chapter concludes with a disscusion of different sensor combinations that have been used in mobile robots, and a short description of multisensor fusion in the navigation and control of a number of existing mobile robots.

2.1. General Approaches to Multi-Sensor Integration and Fusion

2.1.1. Frameworks for Integration

1) *Hierarchical Phase-Template Paradigm*

Proposed by Luo [23], this general paradigm is based upon four distinct temporal phases in the sensory information acquisition process (fig. 2.1):

- far away
- near to
- touching
- manipulation

Each phase is distinguished by the range over which sensing takes place, the subset of sensors typically required, and most important, the type of information desired. In the first phase, "far away", only global information concerning the environment is obtained by detection, localization, object recognition. Likely non-contact sensors are to be used in this stage, as video cameras and range finding sensors. In the second phase, "near to", proximity sensors can be used, while in the next one, "touching", contact sensors as tactile sensors. If is needed, one can proceed to the fourth phase, "manipulating", in which sensors such as force/torque, slippage and weight would be typically used.

The information acquired in each phase is represented in a form of a distinct frame-like template, that stores information both common to all phases (e.g. position and orientation of an object) and specific to the particular phase. During each phase of operation, the information acquired by each sensor is stored as an instance of that phase's template. The information from each sensor can be fused into a single instance of the template.

2) *Neural Networks*

Neural networks provide a fairly well-established formalism with which to model the multi-sensor integration process. Neurons can be trained to represent sensory information and, through "associative recall", complex combination of the neurons can be activated in response to different sensory stimuli. From the many introduced methods I mention here the "simulated annealing" method that is used in finding optimal global paths for mobile robot navigation and the "self-organizing features maps", method developed by Kohonen [39], that can be used to reduce dimensionality of the sensor signals while preserving their topological relationships.

3) *Logical Sensors*

A "logical sensor" [40] is a specification for the abstract definition of a sensor that can be used to provide a uniform framework for multi-sensor integration and fusion. Through the use of an abstract definition of a sensor, the unnecessary details of the actual physical sensor are separated from their functional use in a system, thus providing both portability and the ability to adapt to technological changes in a manner transparent to the system.

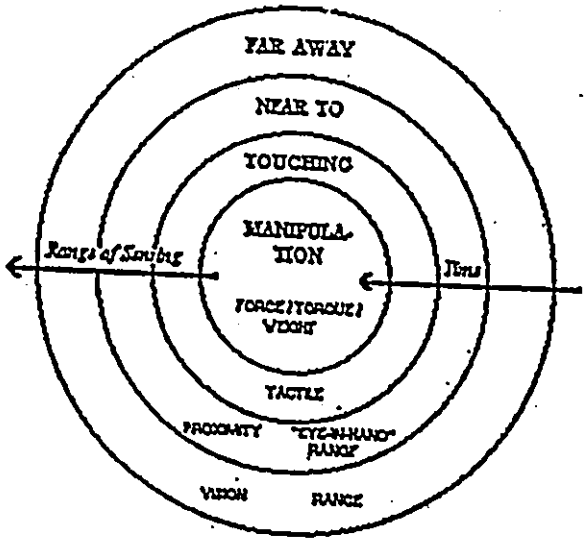


Figure 2.1. The hierarchical phase-template paradigm

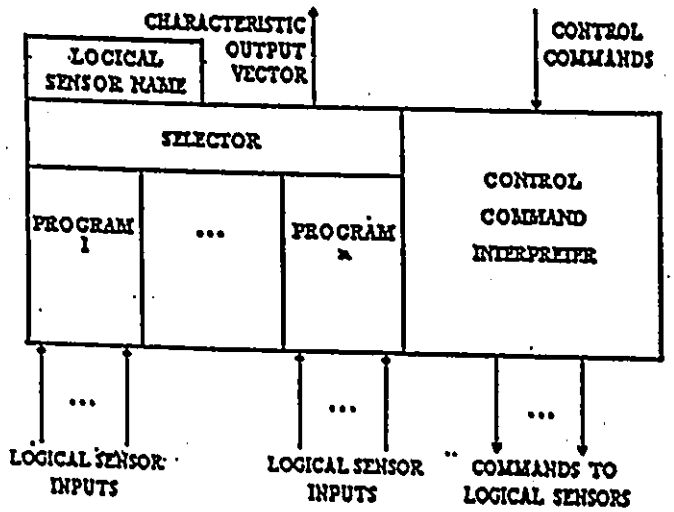


Figure 2.2. Logical sensor

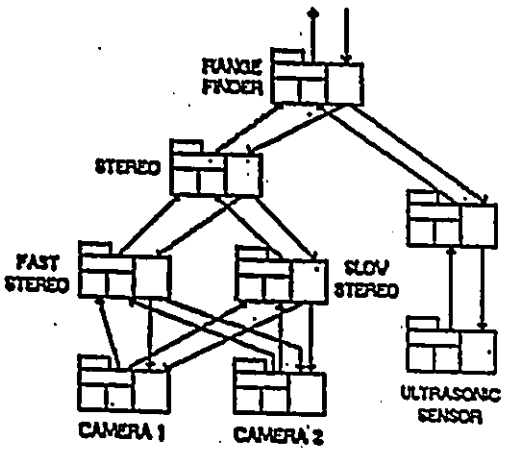


Figure 2.3. Logical-sensor based range finder

Fig.2.2 shows the essential elements of a logical sensor. Each logical sensor can serve as an element in a network of logical sensors, which itself can be viewed as a logical sensor. As exemplification, a hypothetical logical sensor-based range finder is shown in fig.2.3. Both cameras are used as input to a fast and a slow stereo logical sensor. Each of these logical sensors, which differ in terms of the speed and accuracy of their processing algorithms, are used as input to an overall stereo logical sensor. The entire network of logical and physical sensors can provide for a range finder that is both robust in terms of the lighting conditions in which it can operate (i.e. the ultrasonic sensor for poor lighting conditions) and, depending on time constraints, the speed at which it can operate. The range information could possibly be made more accurate if the redundant information from the stereo and ultrasonic sensors is fused at the top-level logical sensor in the network.

4) *Object-Oriented Programming* [24]

In most object-oriented multi-sensor applications, each sensor is represented as an object. From conceptual point of view, is a similar method to that of logical sensor. Objects communicate by passing messages that invoke specialized sensor processing procedures, called methods, based on the sensor's attributes and behavior. Each method is transparent to other objects, allowing possibly different physical sensors to be used interchangeably. Object recognition and multi-sensor robotics tasks are examples of the object-oriented programming method implementation.

2.1.2 Control Structures

Different structures have been used to control the overall integration and fusion process. A distinguished class of control structures is that based on artificial intelligence and therefore it is not mentioned in the following.

1) *The NBS Sensory and Control Hierarchy*

The Center for Manufacturing Engineering, National Bureau of Standards (USA), introduced a multi-sensor interactive hierarchical robot control system based on the mathematical formalism called the cerebellar model arithmetic computer [19,29]. The structure of the control system consists of an ascending "sensory processing" hierarchically coupled to a descending "task-decomposition" control hierarchy via "world models" at each level. The use of multiple levels is motivated by the observation that the complexity of a control program grows exponentially as the number of sensor and the associated processing increases. By isolating related portions of the required processing at one level and assuming that the processing at each level can

be done in parallel, the addition of more levels will not result in an exponential increase in complexity. The amount of processing time at each level is further reduced by the use of a priori knowledge from the world model, that provides predictions to the sensory system. Based upon current sensory information, the world models are updated. The use of a world model promotes modularity because the specific information requirements of the sensory and control hierarchies are decoupled.

2) *Distributed Blackboard*

A blackboard architecture allows economical communication between distributed sensory subsystems in an integrated multisensory system. Each subsystem can send time-stamped summary output to a blackboard where it becomes available to any fusion process as well as the integration functions. The time stamp allows for sensor information to be made commensurate before being fused. Any number of different fusion methods can be implemented using the output from the blackboard. Harmon *et al.*[35] have been used a blackboard architecture for an autonomous vehicle control.

3) *Adaptive Learning* [32]

Adaptive learning is a method of control in which the system discovers the appropriate signals for control based on the output of the sensors. The system is taught a representative sample of correlated control signals and associated sensory outputs over the range of signals and sensory outputs encountered by the system. Based on the associations developed during the teaching phase, it is possible to have the system respond to any combination of sensory outputs with an appropriate control signal. The system requires no a priori knowledge of the world, nor of the structural kinematics of the robot. It is this feature that makes the adaptive learning attractive when there are possibly multiple sensors interacting to produce complex output. It was applied for instance to the Multi-Sensor-based control of robotics manipulators.

2.1.3. World Models

World models, usually defined in terms of high-level representation, enables a Multi-Sensor system to both store and reason with previously acquired sensory information. Two basic type of approach are used: sensory information can add to predefined model of the world, or are used to dynamically create one during the operation. Two general representations are mentioned in the following, many other, closely tied to the fusion methods, being described along with those methods later on.

1) *The Multi-Sensor Kernel System* [41]

This representation is compatible with the specification of logical sensors. Object features are extracted from low-level sensory data and organized into a three-dimensional "spatial proximity graph", that makes explicit the neighborhood relations between features. Subsequent sensory data can then either be matched in terms of the spatial proximity graph, or a "k-d tree" (a binary tree with k-dimensional keys that allows the nearest neighbors of one of k features to be found) is constructed, using the proximity graph, for faster processing.

2) *The NBS Sensory System* [29]

This world model was proposed for the NBS hierarchy control structure (see paragraph 2.1.2). World models at each level in the hierarchy are used to create initial expectations about the form of the sensory information available at that level and then to generate predictions for the task control units in the hierarchy so that they do not have to wait for sensory processing to finish. Errors between the sensed information and the world model are used initially to register the model and later to maintain the consistency of the model during operation of the system.

2.1.4. General Fusion Methods

Most methods of multi-sensor fusion make explicit assumptions concerning the nature of sensory information. The most common assumptions include the use of a measurement model for each sensor that includes a statistically independent additive Gaussian error or noise term and an assumption of statistical independence between the error terms for each sensor. Many of the differences in the fusion methods included here center on their particular techniques (calibration, thresholding) for transforming raw sensory data into a form so that the above assumptions become reasonable and a mathematically tractable fusion method can result. Table I summarizes for comparison the relevant aspects of each general fusion method presented in this section.

1) *Weighted Average*

Is one of the simplest and most intuitive fusion methods. It consist on taking the average of redundant information provided by a group of sensors and use this as the fused value. This method allows for real-time processing of dynamic low-level data and was used as example in the mobile robot HILARE [34], after first thresholding the sensory information to eliminate the spurious measurements.

2) *Kalman Filter* [42]

Being also applied for real-time processing of the raw data, this method is usually preferred to the weighted average method because it uses the statistical characteristics of the measurement model to determine estimates recursively for the fused data that are optimal in statistic sense. If a system can be described with a linear model and both the system and sensor error can be modeled as white Gaussian noise, the Kalman filter will provide unique statistically optimal estimates for the fused data. Further more, the recursive nature of the filter makes it appropriate for use in systems without large storage capabilities. Examples of the use of the filter for Multi-Sensor fusion include: object recognition, robot navigation, inertial navigation, multi-target tracking.

3) *Bayesian Estimation using Consensus Sensors* [23]

The main idea of this method is first to eliminate from the consideration the sensor information that is likely to be an error and to use the information from the remaining "consensus sensors" to compute the fused value. The optimal fusion of the information is determined by finding the Bayesian estimator that maximizes the likelihood function of the consensus sensors. The method was used for example within the hierarchical phase-template paradigm.

4) *Multi-Bayesian*

Durrant-White [28] has developed a model of a Multi-Sensor system that represents the task environment as a collection of uncertain geometric objects. Each sensor is described by its ability to extract useful static descriptions of these objects. An "ε-contaminated" Gaussian distribution is used to represent the geometric objects. The sensors in the system are considered as a team of decision-makers. Together, the sensors must determine a team-consensus view of the environment. A multi-Bayesian approach, with each sensor considered a Bayesian estimator, is used to combine the associated probability distributions of each respective object into a joint posterior distribution function. A likelihood function of this joint distribution is then maximized to provide the final fusion of the sensory information. The fused information, together with an a priori model of the environment, can then be used to direct the robotics system during the execution of different tasks.

5) *Statistical Decision Theory*

This is a two-step method for the fusion of redundant location data from multiple sensors, using statistical decision theory. Location data refer to sensor measurements that are modeled as

additive sensor noise translated by the parameter of interest being sensed. Sensor noise is modeled as the ε-contamination of a variety of possible probability distributions. Initially, the data from different sensors are subject to a robust hypothesis test as to its consistency. Only data that passes this preliminary test are used in the second step for fusing, using a class of robust minimax decision rules.

6) *Shafer-Dempster Evidential Reasoning* [44]

This method is an extension to the Bayesian approach that makes explicit any lack of information concerning a proposition's probability by separating firm support for the proposition from just its plausibility. In the Bayesian approach all propositions for which there is no information are assigned an equal a priori probability. When additional information from a sensor becomes available and the number of unknown propositions is large relative to the number of known propositions, an intuitively unsatisfying result in the Bayesian approach is that the probabilities of the known propositions become unstable. Instead, the Shafer-Dempster approach assigns to "ignorance" the unknown propositions; ignorance is reduced only when supporting information becomes available. The method has been tried in Multi-Sensor target identification and military applications.

7) *Fuzzy Logic* [19]

Fuzzy logic, a type of multiple-value logic, allows the uncertainty in Multi-Sensor fusion to be directly represented in the fusion process by assigning to each proposition a real number between 0 and 1 that would indicate its degree of truth. Consistent logical inference can take place if the uncertainty of the fusion process is modeled in systematic fashion.

8) *Production Rules with Confidence Factors*

This method has been used for object recognition [37]. Production rules represent symbolically the relation between an object feature and the corresponding sensory information, by associating a confidence factor with each rule to indicate its degree of uncertainty. Fusion take place when two or more rules, referring to the same object, are combined during logical inference to form one rule. The major problem is that the confidence factor of each rule is defined in relation to the confidence factors of other rules in the system, making it difficult for example, to add new sensors that require additional rules.

2.2. Multi-Sensor-Based Mobile Robots

Mobile robots are one of the immediate applications of Multi-Sensor data integration and fusion in developing features such as mapping, navigation and control. Different combinations of sensor types have been used and some examples are summarized in Table II, along with the operating type of environment, world model representation and fusion method used. A key role is played by the high-level representation of the world, usually closely tied to the fusion method used. In the following the most used high-level representations are briefly discussed:

1) *Spherical Octree* [33]

A sperical octree is an 8-ary tree structure that at its first level separates a spherical perspective view of the environment into eight octants corresponding to the children of the root node (the entire spherical environment perceived by the robot). This is a recursively division method representing the objects at increasingly finer resolution. Information from each different sensor (usually range sensors), is used to reconstruct three-dimensional surfaces using a knowledge base of typical patterns for the sensor. The reconstructed surfaces are than fused together as part of the process of being represented in the octree structure.

2) *Occupancy Grids*

Originally developed for use with sonar equipped mobile robots [18,27], this representation has been extended to allow for the fusion of information from many different types of sensors. It is a cellular world model representation that uses Bayesian estimation to fuse together each sensor's probabilistic estimate as in order to determine weather a cell in the grid is occupied by an object. The resulting grid can be used to determine paths through unoccupied areas of the grid.

3) *Neural Networks*

This method divides the environment into equal-size volumetric cells and associates each cell with a neuron. In a manner similar to the occupancy grid, the magnitude of each neuron's activation corresponds to the probability that the cell it represents is occupied. The neurons are trained using sensory information from different perspectives. Associative recall can be used to recognize objects in the field and simulated annealing can be used to find optimal paths for navigation.

4) *Graphs*

Graph structures have been used to represent the local and topological features of the environment to avoid having to define a global metric relation between non-adjacent nodes in the graph. When landmarks or beacons are not used to correct cumulative position error, a global metric would contain too much position error to be useful. Graph structures allow the topological features to be represented and reasoned with in an efficient manner. Each point in the graph, represented as an "uncertainty manifold", corresponds to the location of a robot in configuration space at which sensory information was acquired; each arc is labeled with a local measurement of the distance traveled between endpoints. The cumulative uncertainty of the robot as it moves from point to point in the graph is taken into account through the cascading of successive uncertainty manifolds.

5) *Labeled Regions*

Sensory information can be used to segment the environment into regions with properties that are useful for spatial reasoning. The known characteristics of different types of sensory information can be used to label some useful property of each region so that symbolic reasoning can be performed at higher levels in the control structure. Sequence of maps, created as a robot moves, are integrated into a global map by overlaying pairs of maps and then replacing the labels of corresponding regions with a label determined according to a precedence procedure (e.g. if a region is labeled as unexplored in one map and as an obstacle in an other, the global map might label the region as occupied). The global map is then used for obstacle detection and path finding.

6) *Production Rules*

The use of production rules in a control structure [36] allows for a wide range of well-known artificial intelligence methods to be used for path planning and learning purposes. Network hierarchies are created from the schema resulted from production rules that allow inference and matching procedures to take place at multiples levels of abstraction, with each level using an appropriate combination of the available sensors.

As a conclusion of this chapter, I will present in the following a selected list of examples of different mobile robots reported in the literature, their world representation and sensor combination (from [19]):

1) *Hilare:* it was the first mobile robot to create a world model of an unknown environment using information from multiple sensors, in particular contact, acoustic, 2-D vision and laser range-finding. Acoustic and vision sensors are used to create a graph partitioned into a hierarchy of locations, then vision and laser-finding sensors are used to develop an approximate 3-D representation of different regions in the environment. The shape of each object is represented as a polygon. Depending on the features of an object and its distance from the robot, an appropriate group of redundant sensors is selected to measure the object. The uncertainty of each sensor is modeled as a Gaussian distribution. If the standard deviations of all the sensor' measurements have the same magnitude, a weighted average of their value is used as a fused estimate of a vertex of the object, otherwise the measurement from the sensor with the smallest standard deviation is used. The estimated vertices of the object can then be matched to known regions of the world model by finding an object in the model that minimizes the weighted sum of the distance between corresponding vertices.

2) *Crowley's Mobile Robot:* Crowley developed a mobile robot with a rotating ultrasonic range sensor and a touch sensor capable of autonomous navigation in a known domain. Information from a pre-learned global world model is integrated with information from both sensors to maintain dynamically a composite model of the local environment. Obstacles and surfaces are represented as connected sequences of line segments in two dimensions and the confidence as to their actual existence is represented by an integer ranging from one (transient) to five (stable and connected). Information from the global model and the sensors is matched to the composite local model by determining which line segment in the local model has the best correspondence with a given line segment from either the global model or sensors. The best correspondence is found by performing a sequence of tests of increasing computational cost based on the position, orientation, and length of the line segments relative to each other. The results of the matching process are then used to update the composite local model by either adding newly perceived line segments or adjusting the confidence value of the existing ones.

3)*Ground Surveillance Robot:* is an autonomous vehicle designed to transit from one known location to another over unknown natural terrain. Vision and acoustic ranging sensors are used for obstacle detection, while a laser range finder together with a high resolution gray-level camera and a low resolution color camera are used for distant terrain and landmark recognition. A distributed blackboard is used both to control the various subsystems of the vehicle and a mechanism through which to integrate and fuse various types of sensor data. Data is organized into a class tree representation with inheritance properties.

4)*Stanford Mobile Robot:* uses tactile, stereo vision, and ultrasonic sensors for navigation in unstructured man-made environments. A hierarchical representation is used in its 2-D world model. The uncertainty as to the location of the robot and the features in the environment is modeled with a Gaussian distribution. A Kalman filter is used to fuse the measurements from a sensor as the robot moves.

5)*CMU's Autonomous Land Vehicles:* as part of the research on autonomous land vehicles at Carnegie-Mellon University's Robotics Institute, two vehicles were developed, NAVLAB and Terregator. Each of them is equipped with a color TV camera, laser range finder, and sonar sensors. The design of an architecture able to support parallel processing and the development of Multi-Sensor integration and fusion techniques have been major goals of the research. Data in a local world model are represented as tokens with attribute-value pairs. Tokens representing physical objects and geometric locations consist of a 2-D polygonal shape, a reference coordinate frame that can be used to transform the location to other frames, and time stamps that record when the token was created and the time at which sensor data were received that led to its creation. When range data, measured by the camera and laser range finder at different times and locations on the vehicle are to be fused, the coordinate frames or the tokens created by each sensor for these data are first transformed to a common vehicle frame and then transformed forward to the same point in time. The data are now fused, resulting in the creation of a new token representing the fused data.

6)*The DARPA Autonomous Land Vehicle:* built as part of the Defense Advanced Research Projects Agency's Strategic Computational Program, the vehicle is intended to be a demonstration of the state of the art in autonomous vehicle research. A hierarchical control system is used to provide the ALV with the flexibility needed for operation over natural terrain. At the lowest level in the hierarchy, "virtual sensors" and "reflexive behaviors" are used as real-time operating primitives for the rest of the control system. Virtual sensors combine information from physical sensors with appropriate processing algorithms to provide specific information to associated reflexive behaviors. Combinations of behavior and virtual sensors are used to handle specific problems that are part of the overall navigation task. A laser range-scanner, together with orientation sensors to determine the pitch, roll, and x and y position of the vehicle, were used to provide information needed to create overhead map view representations of the terrain called Cartesian elevation maps (CEM). Other range sensors such as stereo vision could also be used to create CEM's. Smoothing procedures are applied to the CEM's to fill in detail not provided by the

sparse laser range information. As the ALV travels over the terrain, CEM's are fused together to provide a means for selecting traversable trajectories for the vehicle.

TABLE 2.1 (from [19])

Method	Operating Environment	Type of Sensory Information	Information Representation	Uncertainty	Measurement Consistency	Fusion Technique
Weighted average	dynamic	redundant	raw sensor readings	-	thresholding possible	weighted average
Kalman filter	dynamic	redundant	probability distribution	additive Gaussian noise	thresholding cali-bration	filtering of system model
Bayesian estimate using consensus sensors	static	redundant	probability distribution	additive Gaussian noise	largest digraph in relation matrix	maximum Bayesian of consensus sensor
Multi-Bayesian	static	redundant	probability distribution	additive Gaussian noise	ε -conta-mination	maximum Bayesian estimate
Statisti-cal decision theory	static	redundant	probability distribution	additive noise	ε -conta-mination	robust minimax decision rules
Evidential reasoning	static	redundant and com-plementary	proposition	level of support versus ignorance	-	logical inference
Production rules	static	redundant and com-plementary	proposition	confidence factor	-	logical inference

TABLE 2.2 (from [19])

Mobile Robot	External Sensors	Operating Environment	World Model Representation	Fusion Method
HILARE	vision, acoustic, laser range finder	unknown man-made	polygon objects in graph of locations	weighted average
Crowley's mobile robot	rotating ultrasonic, tactile	known man-made	connected sequences of line segments in 2-D	best correspondence using integer valued confidence factors
Ground surveillance robot	high-resolution gray level vision, low-resolu-tion color vision, acoustic, laser range finder	unknown natural terrain	triangular segments in blackboard	variety possible (data put in spatial and temporal correspondence)
Stanford mobile robot	stereo vision, tactile, ultrasonic	unknown man-made	hierarchical sensor measurements and symbols	Kalman filter
CMU's ALV's NAVLAB and Terregator	color vision, sonar, laser range finder	unknown roadway	polygons tokens with attribute-value pairs in whiteboard	variety possible (data put in spatial and temporal correspondence)
DARPA ALV	color vision, sonar, laser range finder	unknown natural terrain	Cartesian elevation maps (CEM's)	average elevation change over small CEM areas

PART 3

OCCUPANCY GRID METHOD FOR MOBILE ROBOT NAVIGATION

The problem to be solved is the mobile robot navigation using multi-sensor information. This is a typical example of task-directed information gathering problem.

Hager and Mintz have presented in [11] a general mathematical framework for modeling the problem and have proposed methods for computing the solution. The proposed method is based on developing a decision-theoretic model of task-directed sensing in which sensors are modeled as noise contaminated, uncertain measurement systems, and sensing tasks are modeled by a transformation describing the type of information required by the task, a utility function describing sensitivity to error, and a cost function describing time or resource constrains on the system. The mathematical framework is a standard Bayesian decision-making model where the value of information ("payoff") of an estimate is defined as the average utility (the expected value of some function of decision or estimation error) relative to the current probability distribution; therefore, the best estimate is that which maximizes the payoff, and the optimal sensor viewing strategy is that which maximizes the net payoff (decision value minus observation costs) of the final estimate. As solutions to this upgrading problem do not exist in closed form, an approximation algorithm based on a grid implementation of Bayes' theorem was developed [11].

One main advantage of this solution is its generality - it is independent of a particular sensing modality or a particular sensing problem: the former implies that, by definition, it solves the multi-sensor fusion problem; the latter implies that the same techniques can be employed in a diverse set of applications and therefore provide a unified treatment of many different sensor-based systems.

Another important advantage is that the described method deals with the inherent statistical aspect of the measuring process, modeling the sensors as noise contaminated. Not only that this approach proves to be flexible and close to reality, but also allows the user to deal with many, less expensive sensors.

Finally, by explicitly representing the costs of information gathering and the effect of decision errors, the method is able to determine how much information to gather and therefore where to stop.

All the above advantages make the proposed method appealing for the robot navigation problem and therefore was used in this thesis. In the following is described the mathematical framework and the grid based method, with exemplification for the robot mapping and navigation tasks. Its particularization and implementation for the concrete, given multisensory mobile robot platform is described in the next chapter.

3.1. Sensor Models and Sensing Tasks

3.1.1 Geometric Models

Geometric model representation employs a model of sensor observations in terms of uncertain geometry [16,11]. All geometric objects (features, locations, relations) can be described by parametric, geometric surface descriptions of the following form:

$$g(x,p) = g(x,l;s) = 0, \qquad p \in P,\ x \in X,\ g \in G \tag{3.1}$$

In this description, p is a vector of parameters that describes the essential structure of the system and x is a vector of observable characteristics of the object; in case of geometry, p can be decomposed into a vector representing location l, and a vector describing size and shape, s. Thus

the function $g(x,p)$ can be interpreted as a model of the physical geometric object that maps a compact region in Euclidean space and its observable characteristics to a point p in the parameter space. Each function g describes a particular type of geometric object belonging to a set G: all straight lines, or the family of quadratic surfaces, for example. Each value of p specifies a particular instance of a geometric object modeled by g. For example, all plane surfaces can be represented by the equation:

$$g(x,p) = p_x x + p_y y + p_z z + 1 = 0 \qquad \text{with } x=[x,y,z]^T \text{ and } p=[p_x,p_y,p_z]^T \tag{3.2}$$

A specific plane can be represented as a point $p \in R^3$ in the parameter space. A sensor can be considered as observing values of p. The uncertain event that a sensor "sees" a specific instance of this geometric object can be described by taking the parameter vector p to be a random variable. The likelihood that a particular instance of a geometric feature is observed can now be described by a probability density function (p.d.f.) $f_g(p)$. For a specific type of feature represented by equation (3.1), the p.d.f. describes the probability or likelihood of observing a particular instance of the associated geometric object.

The advantage of describing information in terms of an uncertain parametrized function is that geometric description itself can easily be transformed between different coordinate systems and different object representations, providing a simple but effective means of communicating information between different sensors. The information communications is then reduced to the transformation of stochastic geometric functions of the form of equation (3.1). This geometric description of sensor information provides two essential components: first, a natural way to embed noise models into the structure, and second a mechanism to communicate information between different sensors in the common language of geometry.

3.1.2. Sensor Models

A sensor is considered to be both the hardware and the software used to extract specific properties from the environment. A sensor model is an abstraction of the physical sensing process whose purpose is to describe the ability of a sensor to extract descriptions of the environment in terms of the information available to the sensor itself. Sensor model should provide a quantitative ability to analyze sensor performance, allowing capabilities to be estimated a priori and decision procedures developed in line with information requirements.

A complete model of the sensor would include a description of the effect of all influences on the output of the sensor. These effects can be summarized in the following:

- Statistical noise: the physical implementation of the transducer and its attendant elements lead to the corruption of the sensor signal that can be modeled using probability measures;

- Quantization: in digital systems, the design of the sensor and the associated algorithms have an inherent limited resolution;

- Model uncertainties: errors may occur due to inaccuracies in the sensor placement, mechanical backlash, incomplete or partial information;

Thus, a sensor model of the following general form [16], allows the expression of a large class of sensor errors:

$$z_i = H(p,w,e)+V_i, \qquad H \in H,\ V_i \in V \tag{3.3}$$

The first term, H, describes the ideal relationship between observed features and sensor observations. The behavior of H depends on the world geometry - through the system parameters p, and the choice of sensor control parameters w. There may be additional calibration parameters, e, used to describe the sensor system itself. Slight variations in the actual behavior of the sensor

cause it to depart from the idealized model in unpredictable ways (modeling errors, mechanical backlash, quantization, communication delay). Because H is almost never known with complete certainty, it is allowed to vary within an envelope H and requires fusion methods to tolerate such variations.

The second term, V_i, expresses the fact that observations may be corrupted by additive noise. V_i belongs to a specified class of random variables V, rather then assuming a single description for it. The intent is that we usually are not in a position to state a single model of statistical noise, though we can usually place bounds on the form of its distribution. It is usually assumed that V_i are independent from observation to observation, and that they are independent of the unknown parameters.

3.1.3. Task Models

Within the geometric framework used for the environment, information gathering and fusion consist of choosing a parametric representation and determining the values of the unknown model parameters. This can be viewed as a directed activity, task-oriented one, and therefore a way of optimizing the use of limited computational resources. The information gathering process is governed by three factors:

- what information we are seeking (geometric features)
- the value we place on that information (utility)
- the cost of gathering the information

This point of view naturally suggests a decision-theoretic approach [36]: maximizing the net gain of information based upon the cost of information processing.

Geometric Transformations

In many cases, robotics tasks need to use information in a different form from the given geometric parametrization. To express the relationship, an auxiliary transformation is needed, $I(p)$, indicating how requested information is related to model geometry. Two particular formulations of $I(p)$ are of special interest:

1. The projection function, in which case $I(p)$ restricts attention to a subset of the parameter space. For example, if the requested information is weight and a rectangular representation is employed for a box of density d, the relationship is obvious:

$$I(p) = I(l;a) = d \cdot a_1 \cdot a_2 \cdot a_3 \tag{3.4}$$

In the above example, we may be interested only in the shape description of the object a, even though the geometric model includes position information, l.

2. The indicator function, in which case I(p) encodes a proposition. It is then possible to formulate the problem so that the result of estimation is an indication of whether that proposition is true, false, or not completely decidable, based on sensor information. For example, consider a parallel gripper with jaw travel between 2 and 4 cm, manipulating boxes on the table. To find out if an observed box is of a manipulable size, a following prepositional mapping can be employed:

$$I(p) = \begin{cases} yes; & \text{if } a_1 \text{ or } a_2 \text{ is between } 2 \text{ and } 4 \text{ cm}, \\ no; & \text{otherwise} \end{cases} \tag{3.5}$$

The two exemplified formulations above show that we can use the same framework to determine quantitative (point based) and qualitative (prepositional) quantities.

Utilities

An important element of an information request is some quantification of the effect of errors on task performance. Therefore we require that an information request make reference to a function $u(p,\hat{p})$, where $\hat{p}$ is an estimate of the unknown parameters, and interpret this function as a decision-theoretic utility. Meanwhile, u can be extended to a function of the form $u(p,a)$ where a represents a generic action from a set of possible actions A. From the variety of utility formulations that have been appeared in the literature, I mention here two of the most employed: the 0-1 and the quadratic utility:

1. The 0-1 utility. Referring to the previous example of a parallel gripper manipulating boxes and assuming that the consequences of both types of wrong decisions are equal (trying to manipulate a too big box or deciding not to attempt to manipulate a box that is in fact manipulable), a utility function of the following form can be employed:

$$u(I(p),a)=\begin{cases}1; & I(p)=a, \qquad p\in P, \qquad a\in\{yes,\ no\}\\ 0; & \text{otherwise}\end{cases} \tag{3.6}$$

In this type of formulation, either the constraint is satisfied and the action succeeds, either is not satisfied and the task fails. In some cases, when the effects of the two errors are not equal, we would adjust the weights so that the case {no, yes} has a value between 0 and 1.

2. The quadratic utility is expressed by the following formulation:

$$u(I(p),I(\hat{p}))=-(I(p)-I(\hat{p}))^2 \tag{3.7}$$

As opposed to the 0-1 utility, the quadratic utility express a performance degradation with no notion of success or failure.

The Cost of Gathering Information

Always an estimate can be refined by having more observations or more computation. To express the trade-off of this process, the cost of gathering the needed information has to be introduced. Then the value of the estimate (in terms of its utility) must be weighed against the cost. Hager & Mintz ([11]) define the cost of gathering information in term of time cost of taking another observation w_{n+1}, denoted $c(w_{n+1}, p)$:

1. Time to select a control sequence
2. Time to move to the specified location
3. Time to gather and integrate new information

As an example, a natural model for time costs is a deadline model, in which we specify a nominal maximum time and also how important it is to meet that deadline:

$$c(w,p) = \left(\frac{t_e + t(w,p)}{t_p} \right)^h, \qquad h \geq 1 \tag{3.8}$$

where t_p is the deadline for the sensing task, t_e is the current elapsed time, $t(\cdot,\cdot)$ is the time taken to execute w when the unknown parameters are p, and h is a factor of how "hard" the deadline is. For large h, the deadline acts as a barrier, whereas for h=1 the growth is strictly linear.

3.2. Bayesian Techniques for Data Fusion and Experimental Design

3.2.1. Bayes' Theorem

Bayes theorem is the cornerstone in keeping track of evidence for and against alternatives. The procedure is the following:

When we make our analysis we have at hand a probability distribution for the alternatives which expresses our accumulation of knowledge to that point. This is called the prior distribution, the one that comes before the observation or the experiment. Then we make an observation intended to tell us something about the relative merits of our alternatives. On the basis of this information we modify the prior probability distribution and obtain a new one, the posterior distribution, the one that comes after the observation. In the following experiment we make, this posterior distribution becomes the prior distribution for the next step in the analysis. Therefore, the words prior and posterior are relative words: they refer to the states before and after any experiment. Bayes' theorem gives the relationship between the posterior and prior distributions on the basis of an observation :

If

i) the A_i's are a set of mutually exclusive and exhaustive alternatives,

ii) $P_0(A_i)$ is the prior probability of A_i,

iii) X is the observation, and

iv) $P(X|A_i)$ is the probability of the observation given that A_i is true,

then the posterior probability of A_i is:

$$P(A_i|X) = \frac{P_0(A_i) \cdot P(X/A_i)}{\sum_j P_0(A_j) \cdot P(X/A_j)} \tag{3.9}$$

In the case of a continuous parameter θ instead of the given set of alternatives A_i, the probabilities become distributions and Bayes theorem can be written in the equivalent form:

$$\pi(\theta|x) = \frac{f(x|\theta) \cdot \pi(\theta)}{\int_P f(x|\theta) \cdot \pi(\theta) \cdot d\theta} \tag{3.10}$$

where $\pi(\theta)$ is the prior distribution over parameter θ

x is the observation having a distribution $f(x|\theta)$

$\pi(\theta|x)$ is the posterior distribution

It is said to have a statistical bayesian distribution whenever a parametrized statistical distribution $f(x|\theta)$ and a prior density over the unknown parameter, $\pi(\theta)$ are given.

3.2.2. Data Fusion

A standard Bayesian decision-making framework takes the following general form [36]:

Given a fixed sensor model of the form (3.3) - that means the uncertainty envelope contains only one geometric model and one sampling density, the sensor model gives rise to a conditional probability distribution:

$$f_z(z|p,w,e) = f_V(z - H(p,w,e)) \tag{3.11}$$

From now on, assuming that w, e are known parameters for the fixed sensor model, they will be no more explicitly indicated and therefore, equation (3.4) can be rewritten:

$$f_z(z|p) = f_V(z - H(p)) \tag{3.12}$$

Given a prior density, $\pi(\theta)$, over unknown model parameters, Bayes' theorem describes how to compute the new probability density over the unknown parameters:

$$\pi(p|z) = \frac{f_z(z|p)\cdot\pi(p)}{\int_p f_z(z|p)\cdot\pi(p)\cdot dp} \tag{3.13}$$

This updating process can be iterated over time using independent observations, over sensor configuration by adjusting w, and over sensors by substituting different sensor descriptions into (3.11).

3.2.3. Decision Making

In Bayesian decision theory, decisions are made by finding that action or estimate that has the maximal expected payoff relative to the current parameter uncertainty. It means that given a density π on P and a utility u, we can compute the expected payoff of a decision $\hat{p}$ as:

$$\rho(\pi, \hat{p}) = E^{\pi}[u(p, \hat{p})] = \int_P u(p, \hat{p}) d\pi(p) \tag{3.14}$$

The optimal decision is that having the maximum expected payoff:

$$p^* = \arg\max_{\hat{p}} \rho(\pi, \hat{p}) \tag{3.15}$$

In the case of a nontrivial transformation I, the expected payoff becomes:

$$\rho(\pi, I(\hat{p})) = E^{\pi}[u(I(p), I(\hat{p}))] \tag{3.16}$$

in which case the optimal decision is $I(p^*)$. For convenience, for a fixed task, the following two functions can be defined [11]:

1. a simple decision rule mapping probability distribution representations to decisions:

$$\delta(\pi) \rightarrow I(p^*) \tag{3.17}$$

This simply means that the decision rule would be whichever of *yes* or *no* has higher probability of being correct and the payoff is the probability of being correct. If the weights are changed, than the payoff becomes weighted probability and the optimal decision is the choice with highest weighted probability.

2. a decision rule with the payoff of a decision with respect to a given distribution:

$$r(\pi) = \rho(\pi, \delta(\pi)) \tag{3.18}$$

In this second case, the decision rule is the vector with highest probability of succeeding, and the payoff is that probability.

The decision rule for the final task is the average weight and the payoff is the negated variance of the estimation error, and in most cases more observations result in a better (lower mean-square error) estimate.

3.3. Occupancy Grids

The previously described methods have the intuitive appeal of mathematical simplicity, clarity and generality. In essence, by describing the sensor, the geometric representation and the task, we determine the solution to a problem. But these method's generality and computability depend largely on the representation of probability functions. The probability distribution representations should meet the following requirements:

- be flexible enough to describe both prior and posterior distributions after updating with a nonlinear sensor description;
- can be easily transformed and integrated to accommodate a variety of tasks descriptions;
- be computationally tractable;

The occupancy grid framework addresses the above requirements by employing probabilistic sensor interpretation models and random field representation schemes. It adopts the class of probability distributions that can be described by piecewise-constant density functions. Intuitively, such densities are defined by choosing a partition of the parameter space and defining a probability associated with each set. Densities that are not inherently piecewise-constants are approximated so, as in fig. 3.1, from which it can be seen that piecewise-constant densities can represent skewed, multimodal, and bounded distribution. Furthermore, Bayes' theorem, estimate calculation, and payoff calculation of this class of distributions are relatively simple. In so doing, occupancy grid framework supports robust mapping and navigation strategies and allows a variety of robotics tasks to be addressed through operations performed directly on the occupancy grid representation.

3.3.1 Occupancy Grid Representation

The occupancy grid representation employs a multidimensional (typically 2D or 3D) tessellation of space into cells, where each cell stores a probabilistic estimate of its state. Formally, an occupancy field $O(x)$ is a discrete-state stochastic process defined over a set of continuous spatial coordinates $x = (x_1,...,x_n)$, while the occupancy grid is a lattice process, defined over a discrete spatial lattice. The state variable s(c) associated with a cell C of the occupancy grid is defined as a discrete random variable with two states, occupied and empty, denoted OCC and EMP. Therefore, the occupancy grid corresponds to a discrete-state binary random field. Since the cell states are exclusive and exhaustive, the following relationship applies:

$$P[s(C) = \text{OCC}] + P[s(C) = \text{EMP}] = 1 \tag{3.19}$$

With all these set, the approximation method for estimating the occupancy grid can be defined by formulating Bayes theorem for the grid based representation.

3.3.2 Occupancy Grid Method

The occupancy grid method is the process of updating the probabilistic estimate of the cells based on actual information acquired through new measurements (observations); in doing so, the method makes use of Bayes' theorem.

Given a current estimate of the state of a cell C_i, $P[s(C_i) = \text{OCC} | \{r_t\}]$, called prior probability estimate - that is based on previous observations $\{r_t\} = \{r_1, \ldots, r_t\}$ and given a new observation r_{t+1}, the updated estimate $P[s(C_i) = \text{OCC} | \{r_{t+1}\}]$, called posterior probability estimate is given by (3.19):

$$P[s(C_i) = \text{OCC} | \{r\}_{t+1}] = \frac{P[r_{t+1} | s(C_i) = \text{OCC}] * P[s(C_i) = \text{OCC} | \{r\}_t]}{\sum_{s(C_i)} P[r_{t+1} | s(C_i)] * P[s(C_i) | \{r\}_t]} \tag{3.19}$$

In this recursive formulation, the prior estimate is obtained directly from the occupancy grid while the computed posterior estimate is subsequently stored again in the map. Depending upon the initial knowledge about the environment, the initial state of the cells can be fixed accordingly. When no initial information is available, the maximum entropy priors are used, which means that all cells have the same initial probability estimate of 1/2.

To illustrate the approach in the context of a measurement, lets consider a basic example. Suppose an existing obstacle and a given measurement $r_1 = 2.0$ to that obstacle, with a density support on the interval (-b to b) corresponding to 5 cells, fig. 3.2.a; the corresponding occupancy probability of the cells is presented in the figure 3.2.b and constitute the prior probabilities. The next measurement to the obstacle, taken on the same direction from another point is say $r_2 = 1.0$ with a density support on the interval (-b/2 to -b/2) is presented in the figure3.2.c. The occupancy probability grid method implies the following steps:

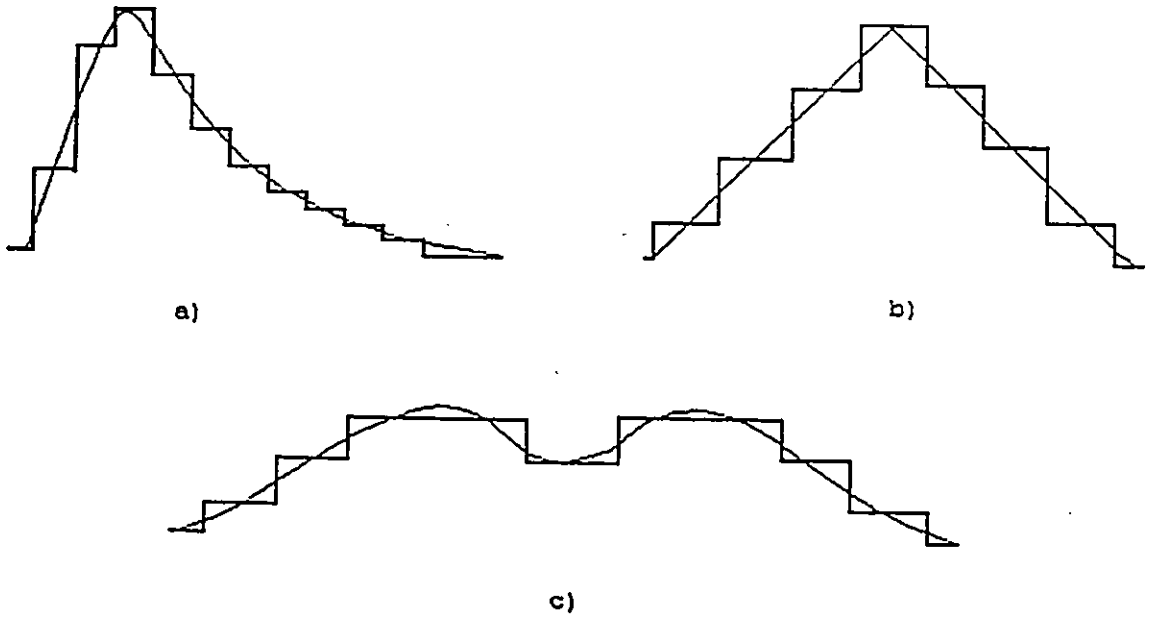


Figure 3.1. Approximation of given distributions in piecewise-constant density functions

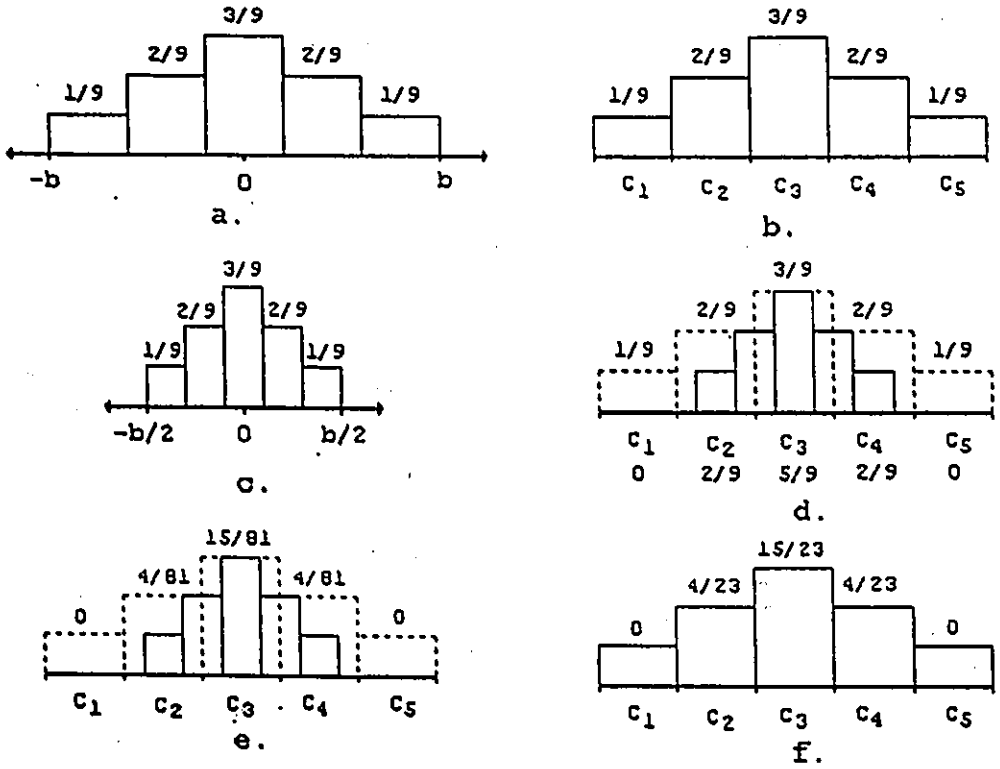


Figure 3.2. Occupancy grid method applied on an example

- compute the occupancy probability of each cell C_i, $i = \overline{1,5}$:

$P[r_{t+1} | s(C_i) = \text{OCC}]$;

given the probability densities, this reduces to compute $\int_{C_i} f(r|p)dp$ for each cell C_i; these values are written under the corresponding cells on figure 3.2.d;

- multiply the prior probability value for each cell by the value calculated in the last step. The result is equivalent to computing the numerator of Bayes rule. These values are written in figure 3.2.e.

- normalize the last result to one, giving the values on fig.3.2.f; with these, we completed the Bayes calculation.

The described method shows how to compute the occupancy probability distribution for any kind of given piecewise -constant density distribution. Furthermore, it can be proved that a repeated Gaussian distribution leads to the behavior of an ideal sensor. In doing so, a distinction has to be made between *obstacle surface range probability distribution* and *the grid (cell) occupancy probability distribution*. Let's assume an obstacle at the range x = 2.5 m, the grid size 0.2 m. The complete demonstration is presented in Appendix A and was done in MathCAD, here being presented only the results.

a) Ideal sensor:

In the case of an ideal sensor, the obstacle surface probability distribution will lead to all cells having occupancy probability zero, except the one at x = 2.5 m that will have probability one (fig. 3.3.a). But because beyond the obstacle's surface the occupancy state of the cells is unknown (probability 0.5), the grid occupancy probability has a different distribution, presented in fig. 3.3.b.

b) Sensor having a Gaussian distribution:

Given a Gaussian distribution:

$$f(r) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{(r-r_0)^2}{2\sigma^2}} \qquad (3.20)$$

with $r_0 = 2.5$ <m> and $\sigma = 0.15$, the obstacle surface probability distribution is presented in fig.3.3.a. A transformation is required to compute from the given distribution the grid occupancy probability distribution. The proposed method is easy to apply and general for any kind of distribution: in computing the occupancy probability for a cell "i" ($i \in 0,...,n$; n - max. nr of cells involved), one would add the sum of all previous cells probability multiplied with 0.5! The reason is that if a given cell has an occupancy probability "p", all the cells beyond it would have been occupied with a probability of "p" times 0.5 (0.5 represents the unknown state). It is easy to observe that the ideal sensor case is nothing else than the particular case of p=1. In figure 3.4.b is presented the computed grid occupancy probability distribution using the proposed method.

In the case of a Gaussian obstacle surface probability distribution (fig.3.4.a), repeated measurements lead, applying Bayes theorem, to the behavior of the ideal sensor, fig.3.3.a. In other words this means that a repeated Gaussian distribution will narrow continuously due to the time convolution process and will converge toward Dirac's function. It can be proved that the proposed method for computing the occupancy probability grid is consistent in the sense that a repeated measurement having the grid occupancy probability distribution in fig.3.4.b lead also to the behavior of the ideal sensor (fig. 3.3.b). In fig. 3.5 are represented with continuous line the initial distribution (probability 0.5) together with five successive measurements and with dashes line the Gaussian distribution of the sensor, having $r_0 = 2.5$ <m> and s $= 0.15$.

In conclusion, the specific steps involved in estimating the occupancy grid from sensor data are:

a. The world state gives a *range data* for a given sensor;

b. The real sensor is modeled by a *stochastic sensor model* defined by a probability density function of the form $p(r|r_0)$, which relates the reading r to the true parameter space range value r_0;

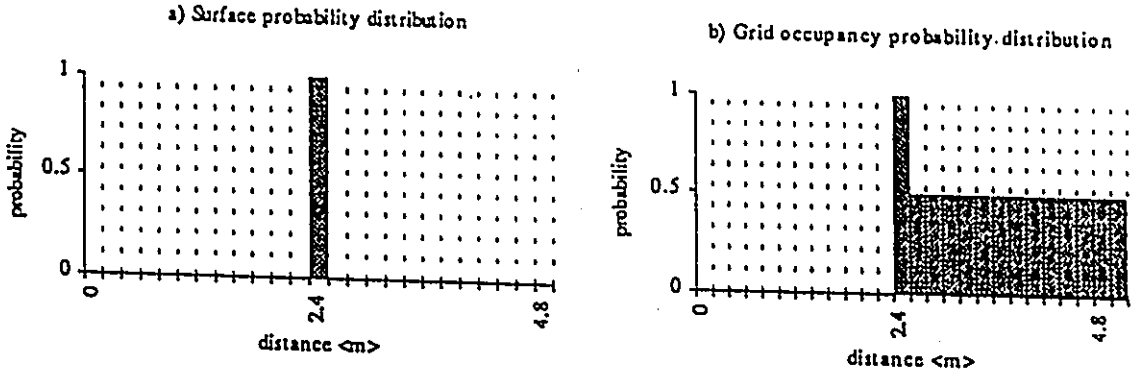


Figure 3.3. Ideal sensor probability distribution

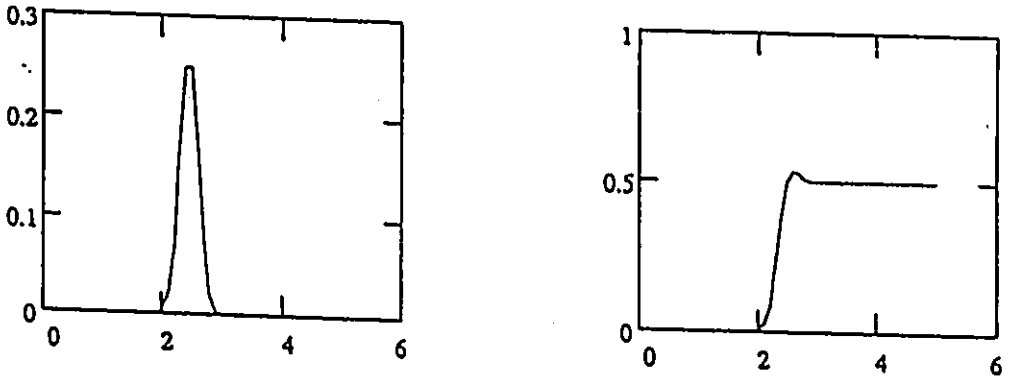


Figure 3.4. Probability distribution for a sensor described by a Gaussian probability density function

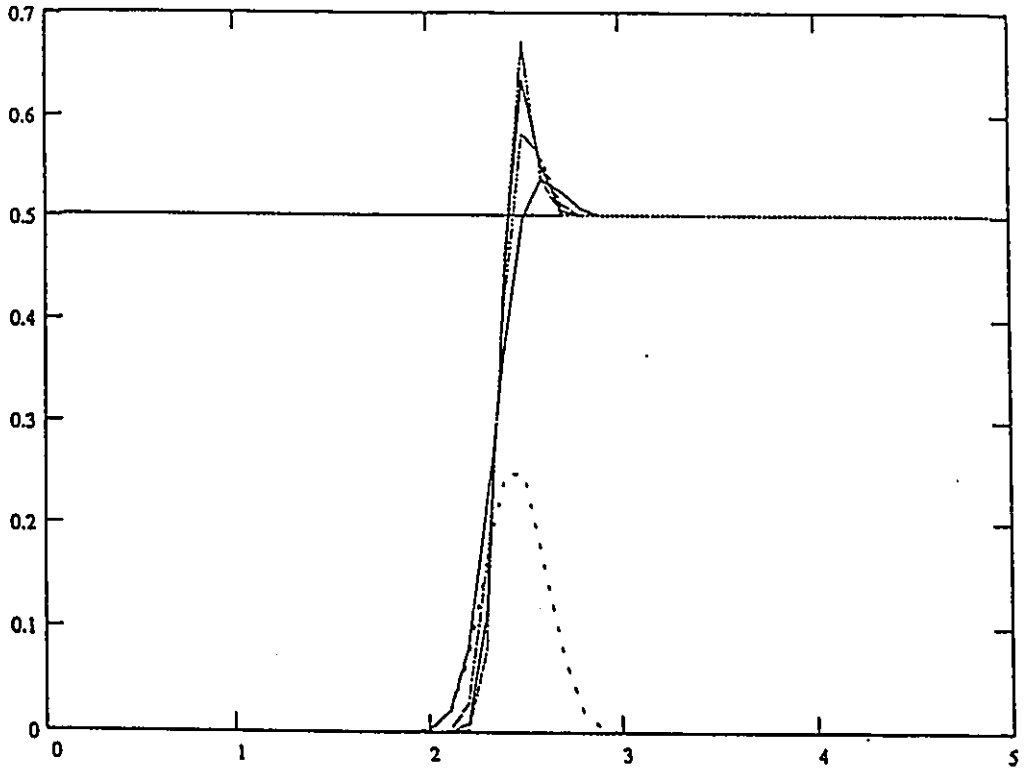


Figure 3.5. Repeated Gaussian distribution leads toward the behavior of the ideal sensor

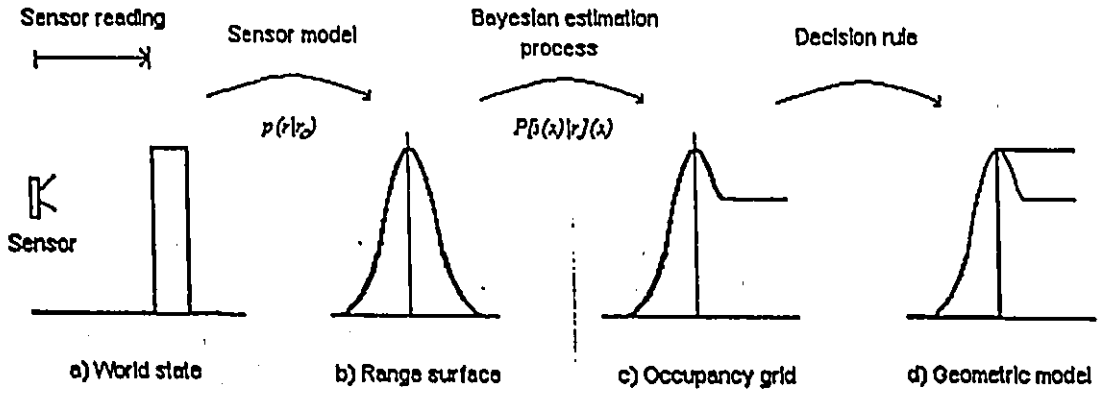


Figure 3.6. Steps involved in estimating the occupancy grid from sensor data

c. The obtained density function is subsequently used in a *Bayesian estimation procedure* to determine the occupancy grid cell state probabilities;

d. Using a *decision rule* to assign discrete state to the cells, labeling them empty, occupied or unknown, is obtained a deterministic world model;

This steps are sketched out in fig. 3.6.

3.3.3 Using Occupancy Grids for Mobile Robot Mapping and Navigation

Particularly important for the mobile robots is to provide improved world models with higher levels of safety and fault tolerance. The first step in achieving this is to use different sensor types that have different operational characteristics and failure modes in order to complement each other. The occupancy grid proves to be the ideal framework for sensor integration, decision making and robot mapping and navigation.

a) Sensor Integration

Given two sensors S1 and S2, the user will have to implement the two corresponding sensor models $p_1(r|r_0)$ and $p_2(r|r_0)$. By using a formula similar to equation (3.19), the same occupancy grid can be updated by the two sensors operating independently. This can be done simultaneously, for any number of independent sensors, of any type (e.g. sonar, vision, laser).

A different estimation problem occurs when separate occupancy grids are maintained for each sensor system and the integration into one framework is done at later stage by composing the corresponding cell probabilities. This is a case of combination of probabilistic evidence from multiple sources and can be addressed using an estimation method known as the independent opinion pool, in which case the evidence for each cell state are summed and a normalization is performed.

b) Decision Making

For applications like mobile robot mapping and navigation, it is necessary to assign specific states to the cells of the occupancy grid. An optimal estimate of the state of a cell is given by the maximum aposteriori (MAP) decision rule (an example of (3.17) mapping decision rule):

- a cell is occupied if its probability of being occupied is greater than its probability of being empty;
- a cell is empty if its probability of being occupied is smaller than its probability of being empty;
- a cell is in an unknown state if the two probabilities are equal;

Based on the fact that the cell states are exclusive and exhaustive (3.19), the MAP decision rule becomes a comparation criteria on the probability of the cell of being occupied:

$$\text{if } P[s(C) = OCC] \begin{cases} > 0.5, \text{ the cell is occupied} \\ = 0.5, \text{ the cell state is unknown} \\ < 0.5, \text{ the cell is empty} \end{cases} \tag{3.21}$$

Depending on the specific task and context, sometime is useful to define an unknown band, as opposed to a single thresholding value. Other decision criteria such as minimum-cost estimates can be also implemented.

c) Mobile Robot Mapping

Mobile robot mapping is a well-defined robotics task, in which the vehicle explores and maps its environment, acquiring information about the world. Data acquired from a single sensor reading is called a *sensor view*. Various sensor views taken from a single robot position can be composed into a *local sensor map*. Multiple sensor maps can be maintained separately for different sensor types, for example sonar and laser. To obtain an integrated description of the robot's surroundings, sensor fusion of the separate local sensor maps is performed to yield a *robot*

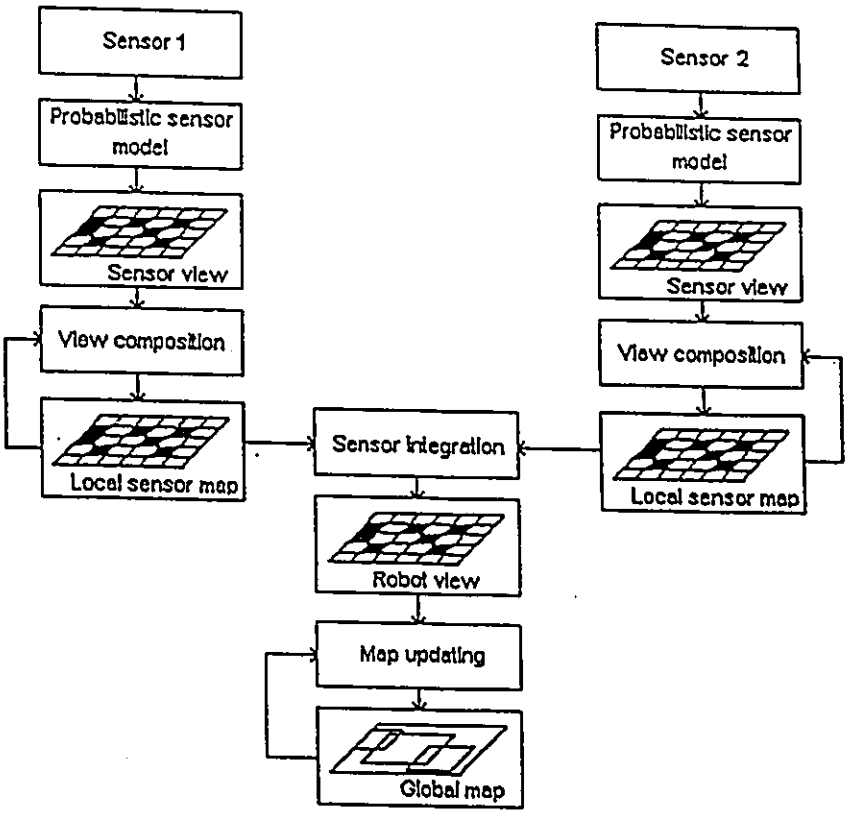


Figure 3.7. Local maps integration into the global map

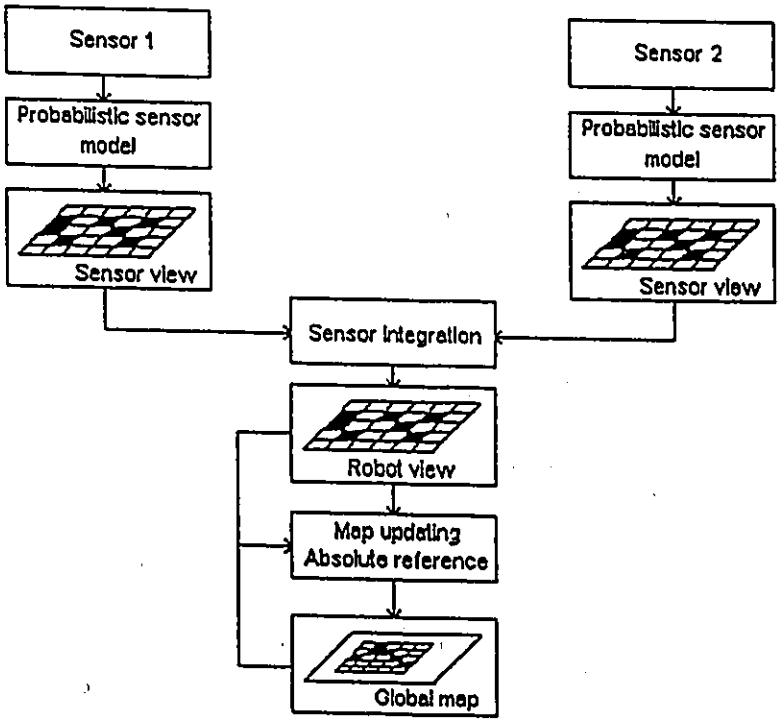


Figure 3.8. Global map direct updating from sensor views

view, which encapsulates the total sensor information recovered from a single sensing position. As the vehicle moves around exploring different parts of the terrain, the different robot views taken from multiple data-gathering locations are composed into a *global map* of the environment. The above flow of processing for the use of occupancy grids in mobile robots is presented in fig.3.7.

As it was mentioned earlier, the same occupancy grid can be updated by different sensors working independently. This case requires the continuous registration of the robot views to a common frame of reference, the only existing map being the global one, updated with each new sensing position. This scenario is presented in fig.3.8.

Although this work deals with the scenario in which the robot operates in unknown environment, it should be mentioned that in the cases that some prior knowledge about the environment exists, this can be provided through precompiled maps representing occupied and empty cells. These maps can subsequently be used as priors or can simply be treated as another source of information to be integrated with sensor derived maps.

d) Mobile Robot Navigation

In many applications, mobile robot navigation addresses issues that are not only related to the ability of moving around but also to aspects like path planning and position estimation and updating.

Usually the path planning and obstacle avoidance problem are performed using potential functions and search algorithms [18]. The latter one can operate directly on the occupancy grid, optimizing a path cost function that takes into account both the distance to the goal and the occupancy probabilities of the cells traversed.

For mobile robots that move around in unstructured environments, recovering precise position information poses major problems. That becomes important in the process of integrating multiple views acquired by the robot from different sensing positions. Over longer distances, dead

reckoning and inertial navigation estimates are many times unreliable. Two different approaches are usually implemented in solving the motion problem: landmark tracking and map matching (correlation method). The position uncertainty can be integrated in most cases in the map updating process as a blurring or convolution operation performed on the occupancy grid.

In the *global map*, the motion of the robot is related to an absolute coordinate frame, and the current view is blurred by the robot's global position uncertainty prior to composition with the global map. Over longer distances, this procedure has the effect that the new views become progressively more blurred, adding less and less useful information to the global map. That means that observations seen at the beginning of the exploration are "sharp", while recent observations are "fuzzy". From the point of view of the inertial observer, the robot eventually "disappears" into a cloud of probabilistic smoke.

For the *robot based mapping*, the registration of the global map becomes more and more uncertain due to the recent movements of the robot. Therefore, the global map is blurred by this uncertainty prior to the composition with the current robot view. A consequence of this method is that observations performed in the remote past become increasingly uncertain, while recent observations have suffered little blurring. In other words, from the robot point of view, the immediate surroundings, which are of direct relevance to its current navigational tasks, are "sharp".

To avoid losing the local spatial relationship and in order to solve the above duality, Elfes proposed in [5] a two-level spatial representation. on one level, the individual views are stored attached to the nodes of an "approximate transformation graph" that describes the movements of the robot. On the second level, a global map is maintained that represents the robot's current overall knowledge of the world. This two-level structure representation can be sufficient and effective for various navigation tasks.

3.4. Experimental Results

A mobile robot with an onboard manipulation capability has been developed as an experimental platform for a multi-sensor system for teleautonomous applications in an unstructured environment. The system provides information which reduces uncertainties about the environment and the robot state. Sensory data gathered from different sources such as vision, infrared (IR) range finders and wheel/joint encoders are integrated in a unique framework which allows to deal in a common way with the problems of position estimation, obstacle detection and avoidance and environment map construction. Tactile and force/torque sensory data are used for the control of object manipulation operations. Human oriented sensory interfaces allow a teleoperator to experience virtual reality feedback of a visual and tactile nature.

3.4.1. System Architecture [1,3]

The mobile robot being developed has the conceptual diagram illustrated in fig.3.9. It consists of a computer controlled four-wheel vehicle, shown in fig.3.10, with a frontal robot manipulator, also computer controlled. The two front wheels are steered with a DC motor and their position is measured by an absolute joint encoder built around a potentiometer. The rear wheels are driven by another DC motor. An incremental wheel encoder was developed to be used for vehicle's position recovery by dead reckoning. The vehicle has the overall dimensions 130 cm x 72 cm. An electronic interface has been developed for the two DC motors and the joint/wheel encoders, in such a way to allow their control either through the onboard computer or directly from a joystick.

Eight infrared (IR) range finders and a monocular vision system with grid encoded structured light were used as exteroceptors for the position control of this mobile robot. The limited perceptive capability of the onboard computer can be substantially complemented by the superior perception expertise of an operator provided with telepresence capabilities. The tactile interface consists of a tactile sensor array mounted in the jaws of the robotics manipulator and a tactile monitor placed in contact with the teleoperator's palm.

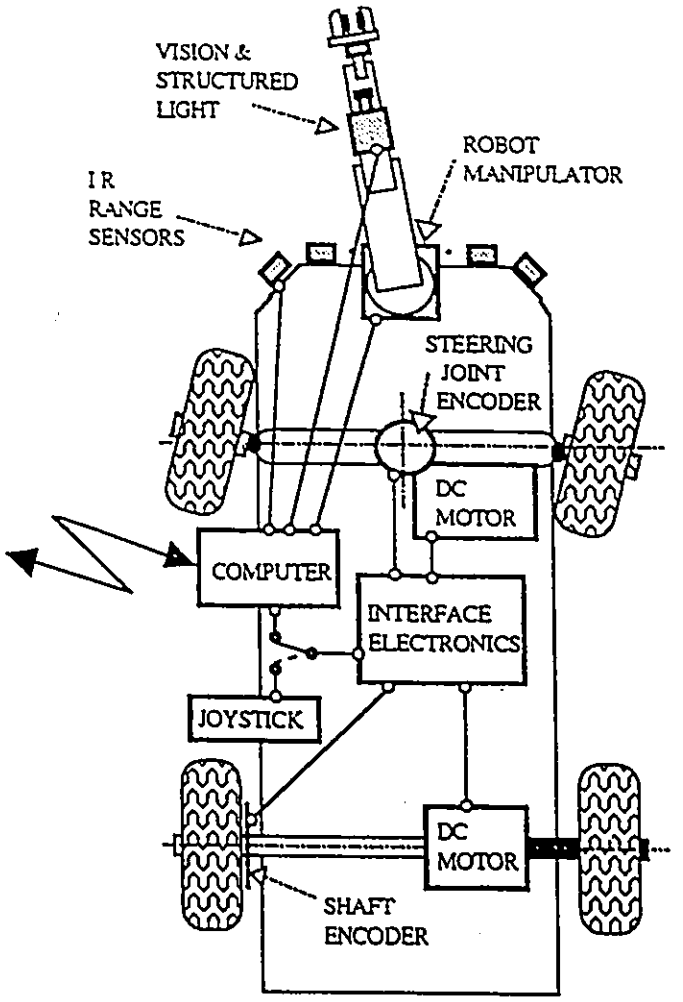


Figure 3.9. The mobile robot. Conceptual diagram

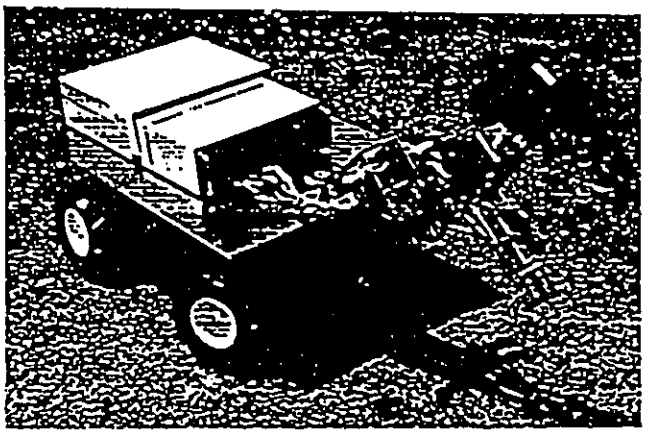


Figure 3.10. The mobile robot. Photo

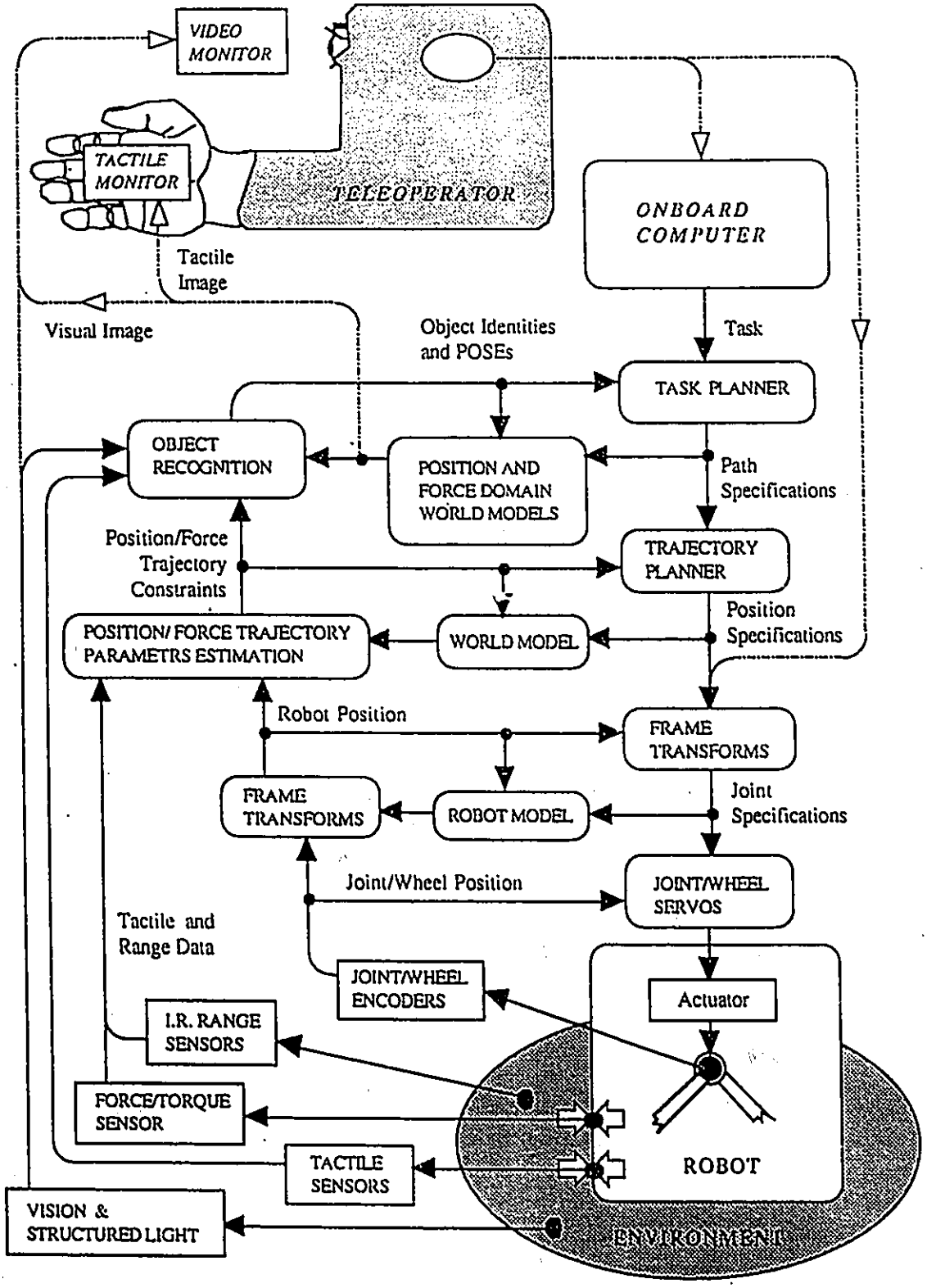


Figure 3.11. The multi-sensor data fusion system's hierarchical architecture

The multi-sensor data fusion system has a hierarchical architecture, shown in fig.3.11, based on the NBS Sensory and control hierarchy (see paragraph 2.1.2). The multi-sensor data fusion should solve sensor redundancy problems occurring when a plurality of sensors (usually of a different type) than are actually required are used to measure a given object parameter. Redundant sensor data are integrated at the lower levels of the architecture and complementary information is fused at the higher levels. In the following sections I will concentrate on those parts related to the problem of mobile robot navigation implementing the occupancy grid method, highlighting my personal contributions to the project.

3.4.2. Electronic Interface

The electronic interface was designed to control the speed and direction of the mobile robot platform by driving two brush DC motors. The key role is played by the MC33033 Brushless DC controller that drives a TMOS Power MOSFET H-Bridge, MPM3004, both circuits developed by Motorola [17].

The block diagram is presented in fig. 3.12 and consists of a voltage controller with a negative feedback loop (provided by the potentiometer joint encoder) for the steering and a differential amplifier for the rear wheels continuous speed control (forward/backward), as shown in fig. 3.13 and 3.14. A switch can chose the control from the onboard computer or directly from a joystick. The component placement diagram of the interface is presented in fig. 3.15.

The eight sensors are connected to a sensor controller (developed at NRC/ITT) that is interfaced to the PC through the serial port RS 232.

The hardware electronic interface is connected to the PC to the LAB-PC data acquisition card [7]. It consists of a 12-bit successive approximation ADC with eighth analog inputs, two 12-bit DACs with voltage outputs, 24 lines of TTL-compatible digital I/O, and timing features unused in our applications. The use of the card is briefly explained in the following by making reference to fig. 3.12:

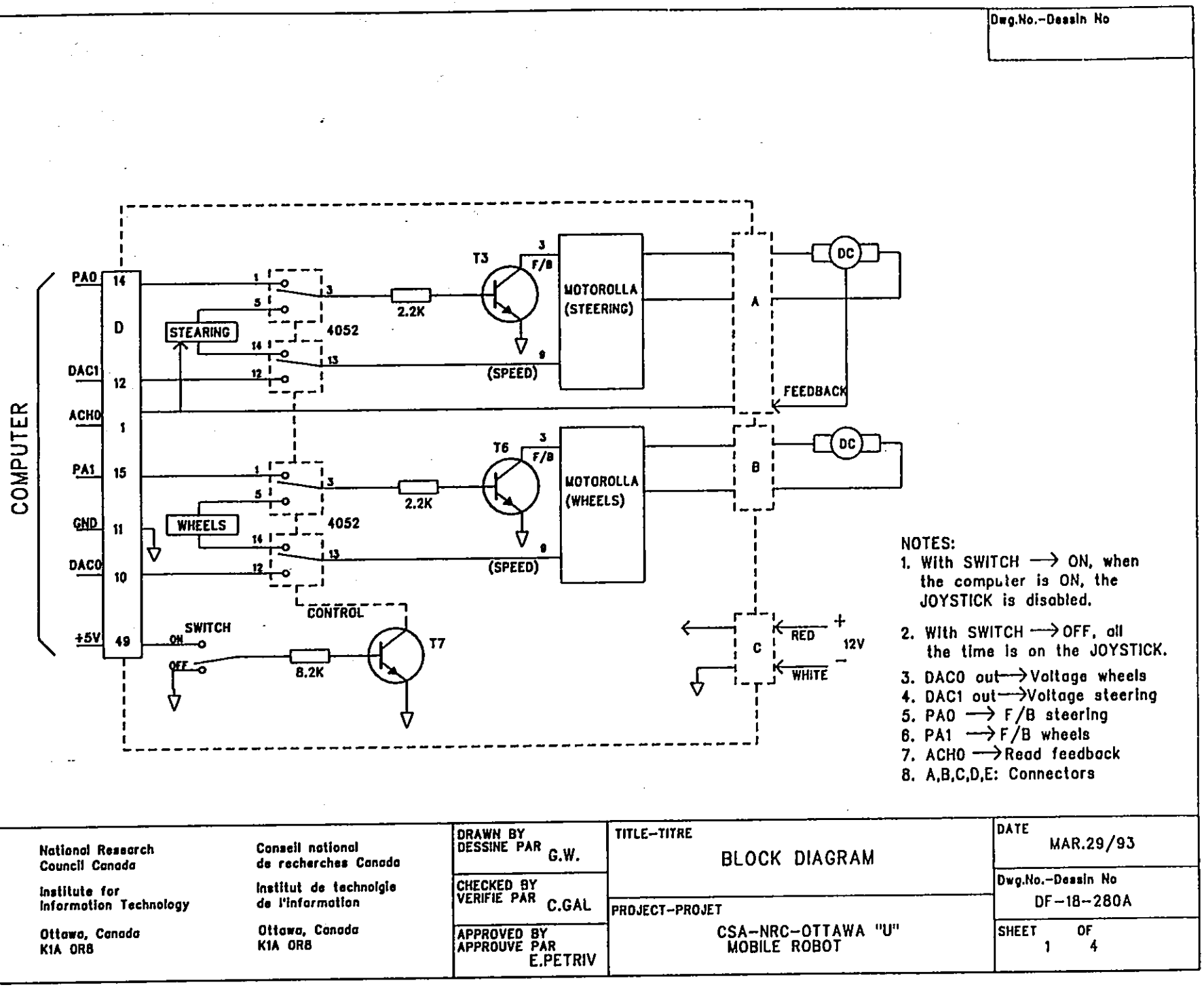


Figure 3.12. The electronic interface. Block diagram

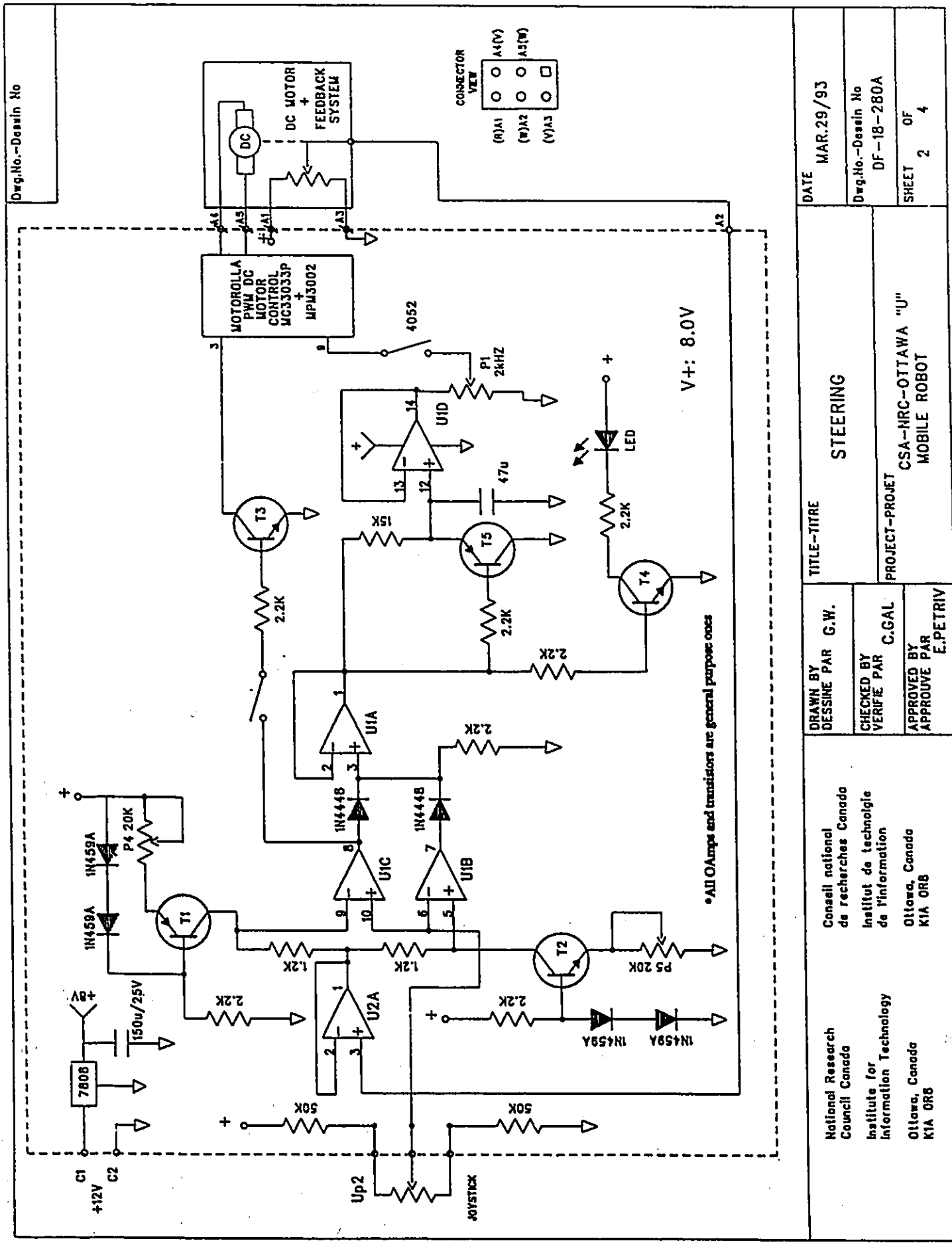


Figure 3.13. The electronic interface. Steering module

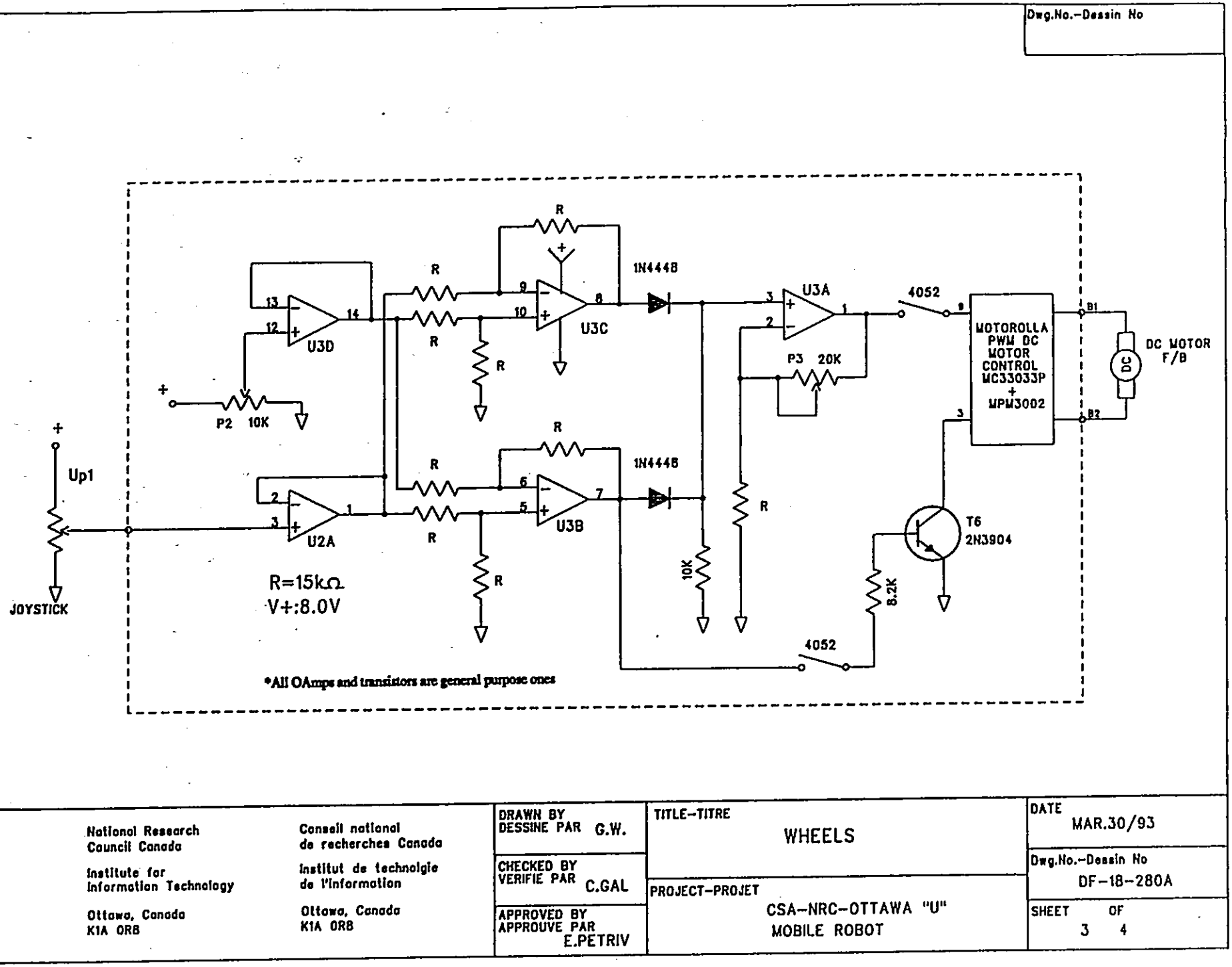


Figure 3.14. The electronic interface. Speed controller

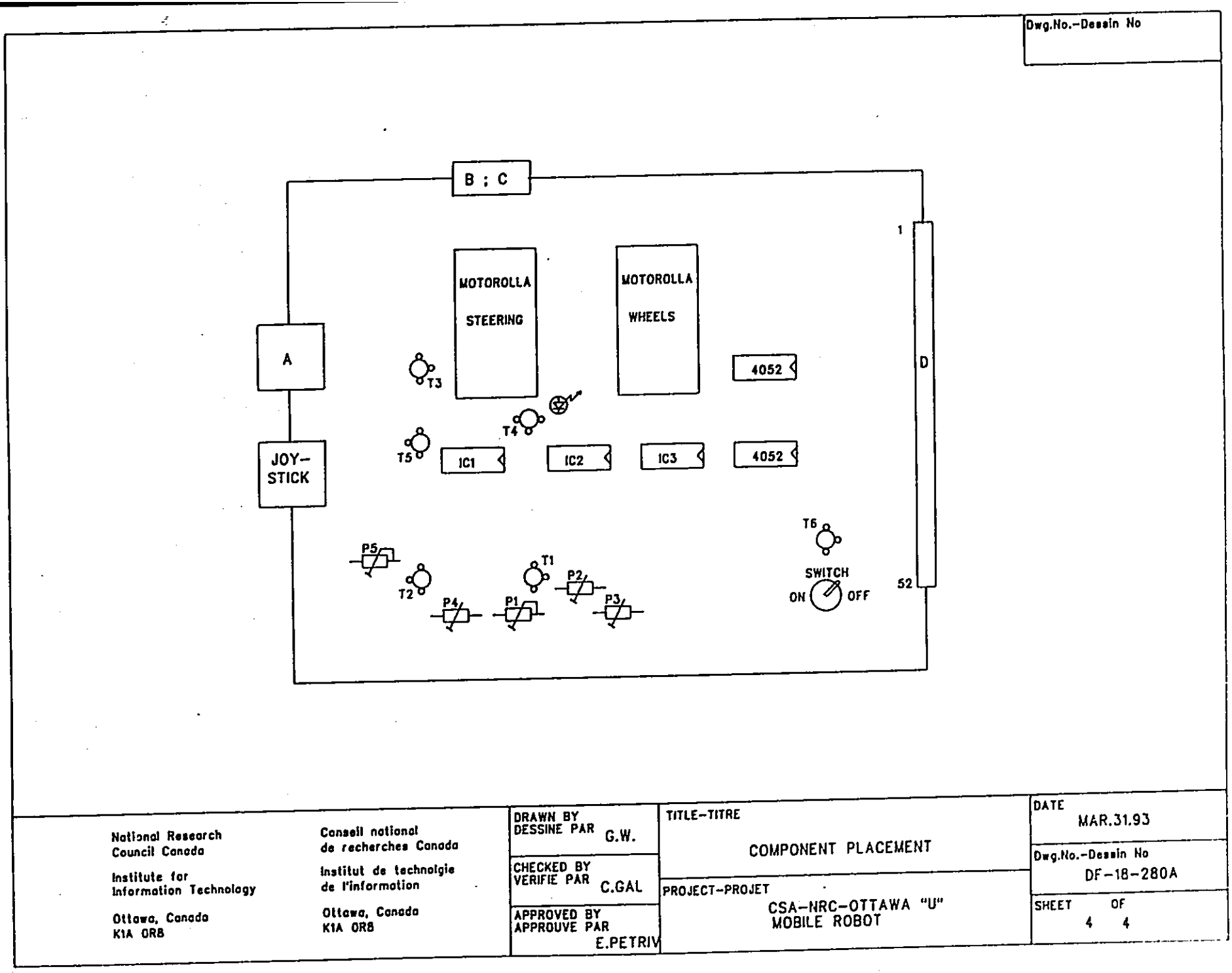


Figure 3.15. The electronic interface. Component placement

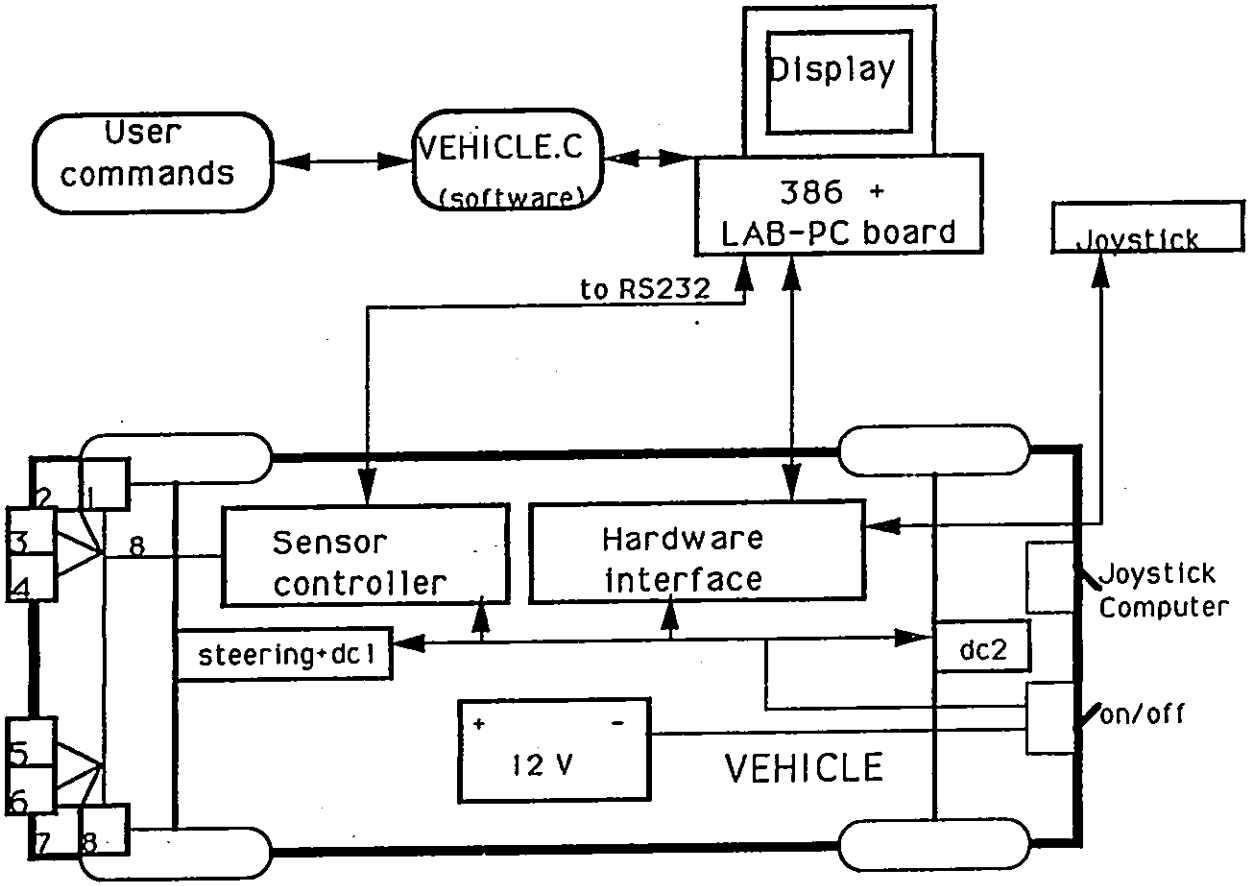


Figure 3.16. The mobile robot interface architecture

-the two DACs were used as inputs for the speed and steering voltage (DAC0 and DAC1,);

- two TTL lines (PA0 and PA1) were configured as outputs to switch the rotation sense of the dc motors (forward / backward, left / right);

- one ADC input (ACH0) was used for the negative feedback voltage coming from the steering, for the front wheels position controller;

The mobile robot interface architecture is presented in fig. 3.16. In Appendix B are listed the subroutines used to create the user-card software interface as well the GUI subroutines.

3.4.3. Infrared Range Sensing [1,4,6]

Robot system designers make compromises between the viewing depth, firing patterns and delay between firings to produce data gathering speeds that are considerably less than a second, but with a concomitant loss of viewing depth. From these points of view, the single transducer pulse-echo sonar systems have several disadvantages:

- the wavelength (for example, for the speed of sound 340m/sec that means 6.8 mm at a frequency of 50 kHz,) is in a range in which the object appear smooth, meaning that unless the sensor is pointed almost normal to the surface of the target, very little energy is returned to the transducer;

- the multiple reflections lead to erroneous readings in a multiple sonar sensor arrangement;

- the speed of sound in the medium (approx. 340 m/s) limits the speed of firing (for example at 7 m distance, 41 ms are required, plus an extra 50 ms to avoid interference);

These and other disadvantages have led to consider light-based ranging alternatives. In this project, the main goal was to find a cost effective alternative to the commercial sensors, that provides the following characteristics:

- a ranging system that would sense a volume in front of a vehicle with at least a 2.5 m depth over a 90 degree angle;

- the sensor output would indicate the distance of an object within one of at least 4 range steps within the 3 m sensing depth; target color would have a minimal effect on the response of the sensor;

- the time required to sense the coverage area would be as short as possible, with a maximum time of 300 ms;

- the system would easily be connected to and controlled by a computer;

- each sensor would be small enough to be easily mounted under the bumper of the experimental vehicle;

- the sensor would be light-based and capable of operation in bright sunlight;

Many different solutions were tested at NRC/IIT, the potential solution being searched among the IR auto-focus technology used in a number of video cameras and still cameras. The Pentax auto-focus module proved to have a very narrow field of view, along with a range measure with medium to high accuracy (depending on range, up to about 3 m), irrespective of target reflectivity, at a cost of only $50 as a spare part.

The Pentax auto-focus module, that comes together with the Pentax Zoom 60x camera ($190 CDN) is based on the active triangulation principle, fig. 3.17. The PSD provides a differential current output that varies depending on the position of the light spot reflected from the target, on the detector. The IR module uses one main circuit board that houses the key circuitry needed for the ranging sensor, power control, light level sensor and microcontroller. The sensor's effective beam width was found to be approximately six degrees. Changing the angle of the target to the principal axis of the sensor did not have a significant effect on the range response.

Based on the Pentax module, the IR ranging system , shown in fig. 3.18, consists of eight IR sensors arranged such to provide a 90 degrees field of view in front of the vehicle. The field of view of several sensors intersect, producing higher density coverage, fig. 3.19. The sensor interface was developed by NRC/IIT, and consists of a single chip microcontroller that interprets

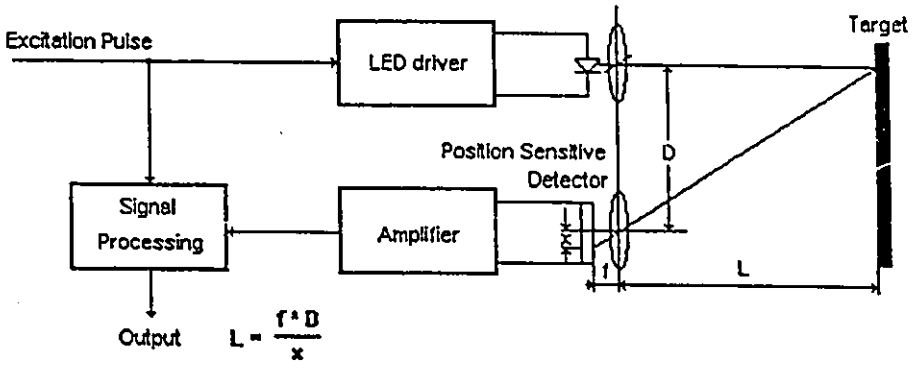


Figure 3.17. The Pentax Zoom 60x camera. Triangulation principle

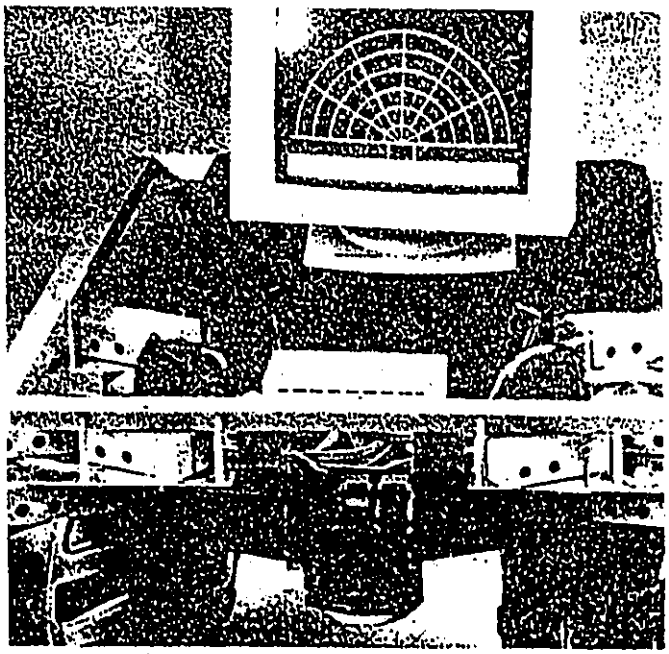


Figure 3.18. The mobile robot IR ranging system (photo from front)

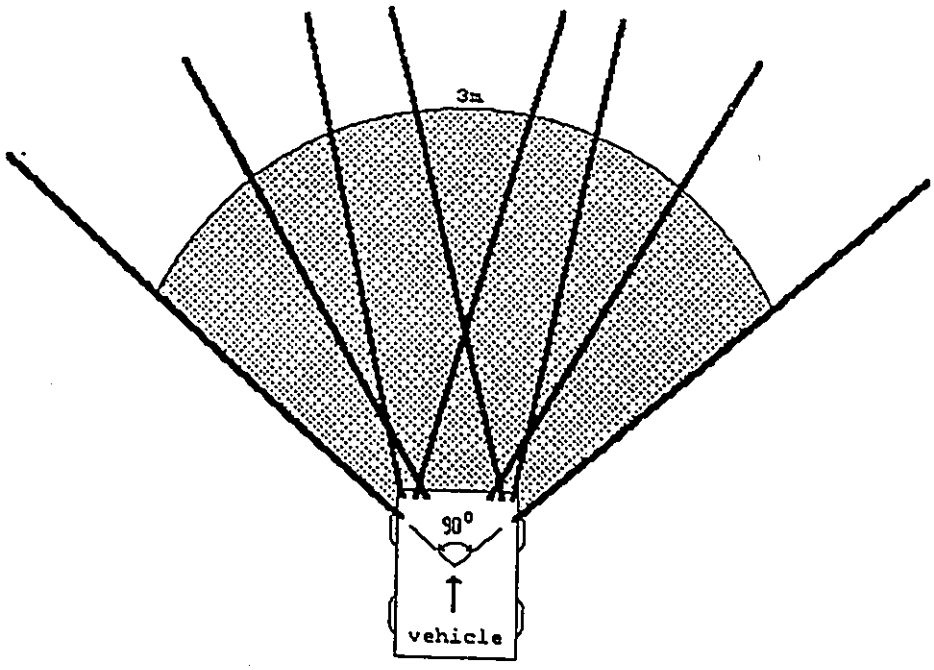


Figure 3.19. The mobile robot IR ranging system. Field of view in front

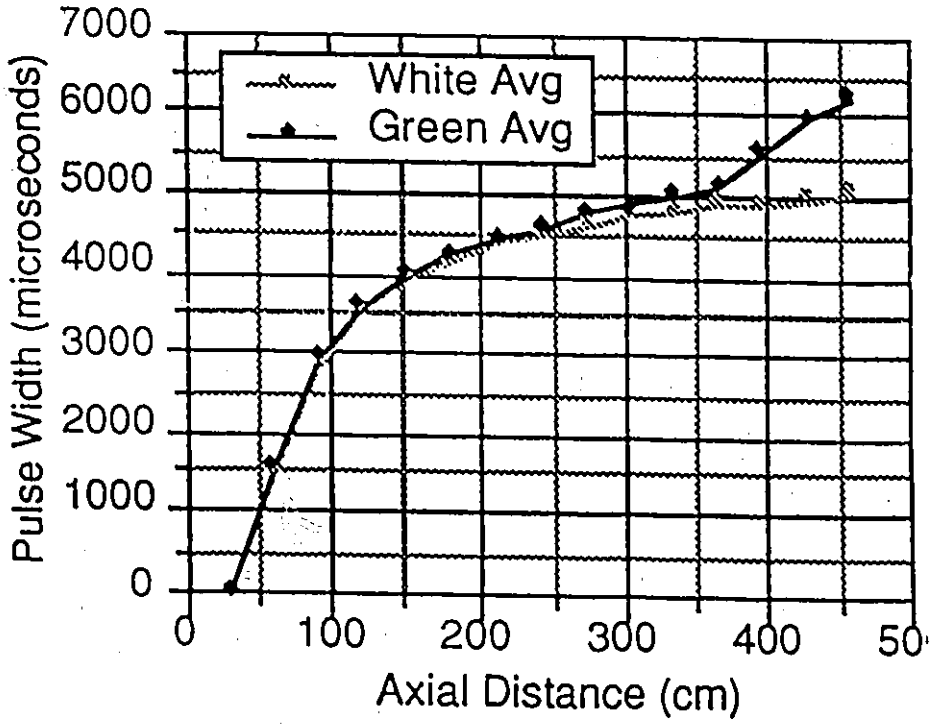


Figure 3.20. Typical axial response for an IR sensor

commands over an RS-232 port, controls, multiplexes and measures signals from the sensors and sends range data back to the computer over the serial port.

3.4.4. Infrared Sensor Calibration and Modeling [4,6]

The raw data returned by the IR sensors is a 16 bit number representing a pulse width that corresponds to a given range measurement. A typical axial response of a sensor is presented in fig. 3.20 [from 4], for a white and a green target. The axial response, repeated in bright sunlight showed no measurable performance degradation [4].

Calibration

The raw data is required when calibrating the sensors. Calibration requires compiling the sensor output for a white matte target at a number of different measured distance from the sensor. The resulting information maybe used to create a transfer function between the raw output data and the range reading of the target. The controller has the transfer function for performing the conversion directly. However, any slight geometric differences within the sensor result in significant variation of output range reading when using the conversion function. In applications where accuracy is important, a calibration table for each sensor must be created. The calibration table has output range values as entries with each entry separated by a constant time of 32 microseconds. The raw data from each sensor is used as an index into each its own unique calibration table to obtain the actual range. In fig. 3.21 is presented an example of such a calibration table for an IR sensor.

Modeling

The occupancy grid method makes use of probabilistic sensor models. A sensor model is defined as the conditional probability density function $p(r/d)$ relating the sensor reading r to the actual distance d. This model is determined experimentally from the error characteristics of each sensor, as shown in fig. 3.22. What is important is not only the maximum relative error measurement, but how the errors are distributed for a given range. Starting with the error characteristics, the curve can be segmented for different input ranges for which the error

Distance	Sensor number (average readings in 500 ns)							
	1	2	3	4	5	6	7	8
18'	158	712	668	932	609	948	709	878
2'	2682	2942	2826	3095	2822	3081	2861	2963
3'	5396	5794	5705	5965	5708	5956	5660	5620
4'	6741	7272	7156	7411	7178	7412	7086	7005
5'	7529	8113	8002	8250	8027	8246	7866	7793
6'	8061	8695	8569	8825	8603	8816	8419	8340
7'	8432	9098	8967	9215	9009	9210	8842	8717
8'	8712	9401	9264	9508	9304	9498	9160	9000
9'	8923	9625	9494	9738	9534	9727	9399	9219
10'	9099	9816	9674	9931	9719	9893	9581	9397
11'	9253	9979	9839	10088	9871	10031	9739	9548
12'	9347	10095	9955	10205	9987	10134	9846	9656
13'	9450	10213	10056	10300	10101	10223	9945	9742
14'	9535	10313	10149	10407	10196	10318	10038	9836
15'	9591	10366	10178	10447	10249	10363	10094	9886
16'	9654	10427	10241	10509	10325	10417	10129	9914

Figure 3.21. IR sensor calibration table

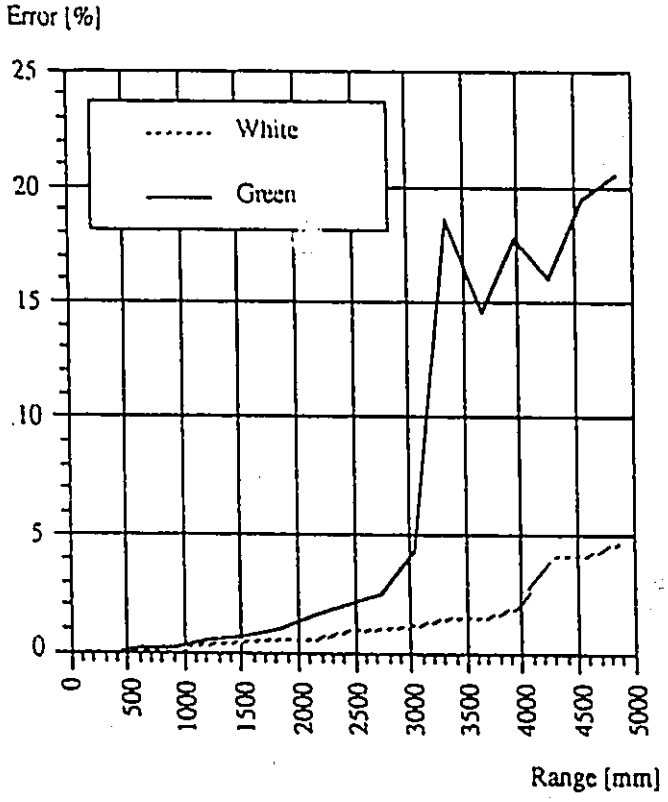


Figure 3.22. IR sensor error characteristic

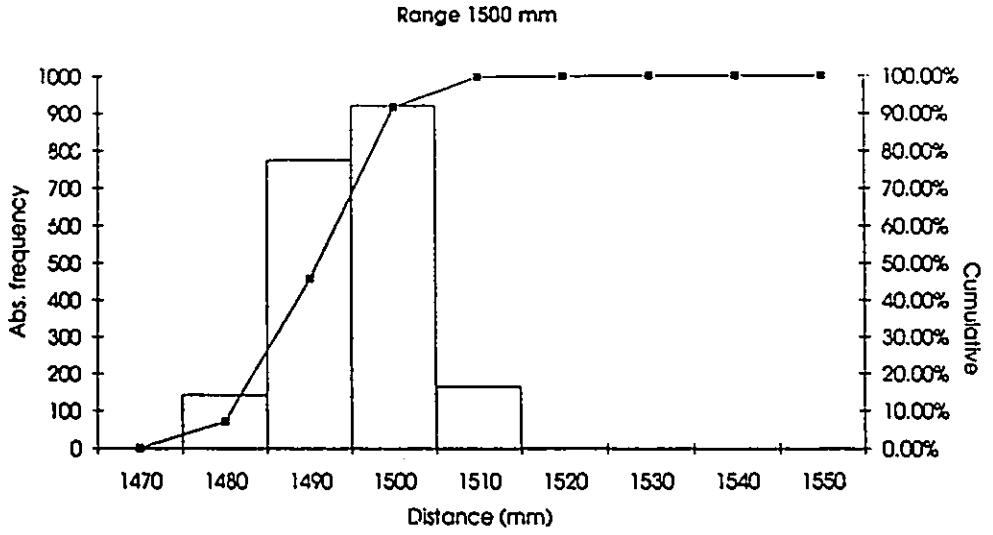
distribution can be experimentally determined. Of course that finer the segmentation, better, but the pay-off is the memory requirement and the computational increase that decreases the acquisition speed. For the IR sensor presented in fig. 3.22, four different segments were assigned with the errors distribution provided from measurements at 1.5 m, 3 m, 4m and 4.8m. The experimental results are presented in fig. 3.23-3.26 (a), and Table 3.1. Given these distributions experimentally determined, any other distribution with a different distance increment step can be done, by computing the corresponding histogram. Fig.3.23-3.26 (b) present the computed histograms for a distance step of 0.1 m, value that matches the grid size of the scene in the mobile robot navigation experiment.

Given the sensor models for the range surface and using the Bayesian estimation process and the occupancy grid method proposed in chapter 3.3.2, we can compute the occupancy probability grid for the IR range measurement.

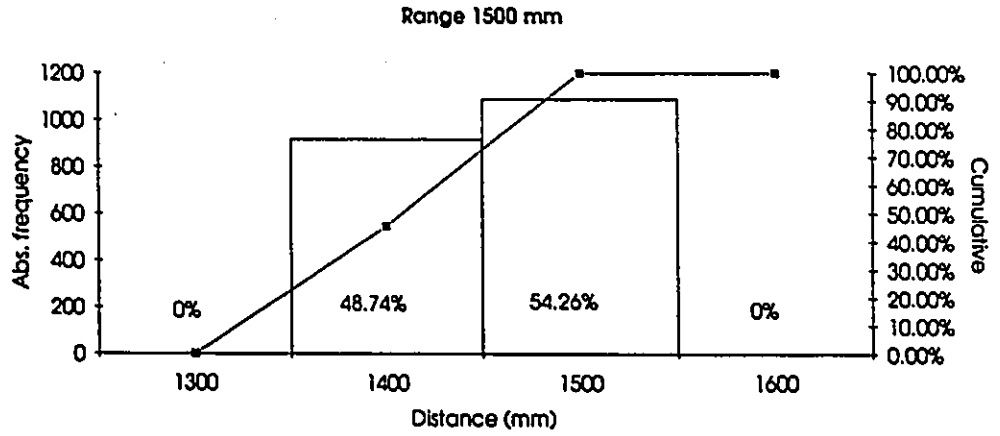
3.4.5 Occupancy Grids

The mobile robot's surrounding environment is stored by the main computer in the form of a grid (cell) based map, each cell being assigned a probability of being occupied. Given the overall dimensions of the mobile robot and the sensor's precision, the grid size was chosen to be 0.1 m. The IR sensors output characteristics was used to produce a sensor probabilistic model. Due to the fact that the IR sensor characteristic is a non-linear curve, a calibration table kept in memory gives the range measurement; furthermore, the characteristic was divided in four segments (0-1.5m, 1.5-3m, 3-4m, and over 4m), each of them having its own probabilistic model. These models were experimentally determined and computed for steps increments of 0.1 m, such that to match the grid size of the map (fig. 3.23-3.26).

The mapping method used is the one that updates directly the global map, which means that although local views (maps) are taken all the time, the probability occupancy grid estimation updating process is done directly on the global map (fig. 3.8). The local maps can be represented on the screen at the user option, and they can be used for navigation purpose. These maps are

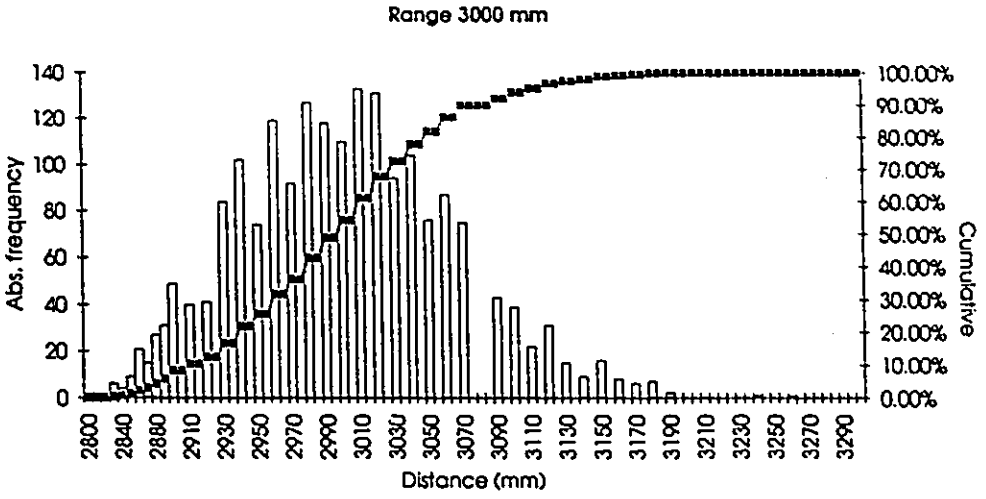


a) histogram for 10 mm step increments

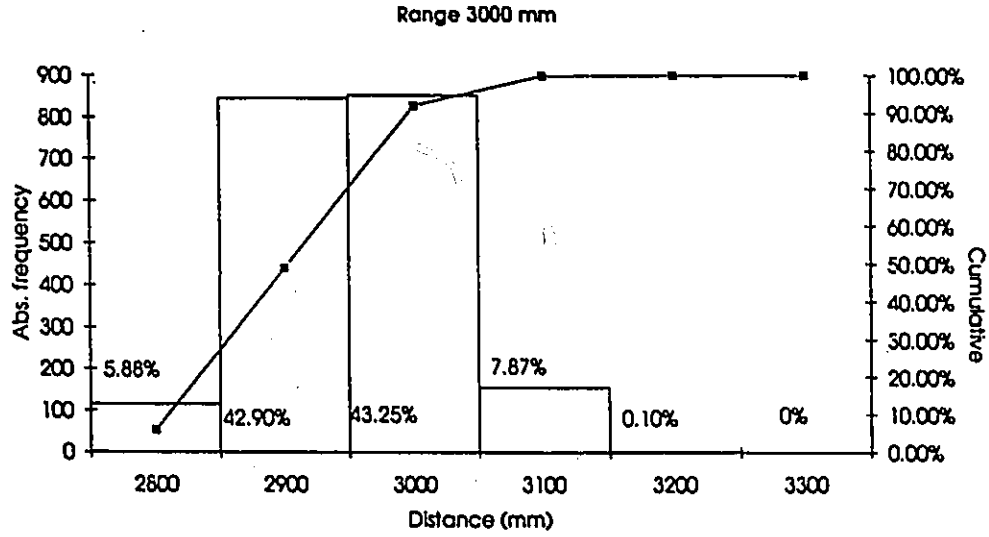


b) histogram for 0.1 m step increments

Figure 3.23. IR sensor probabilistic model for 1.5 m range

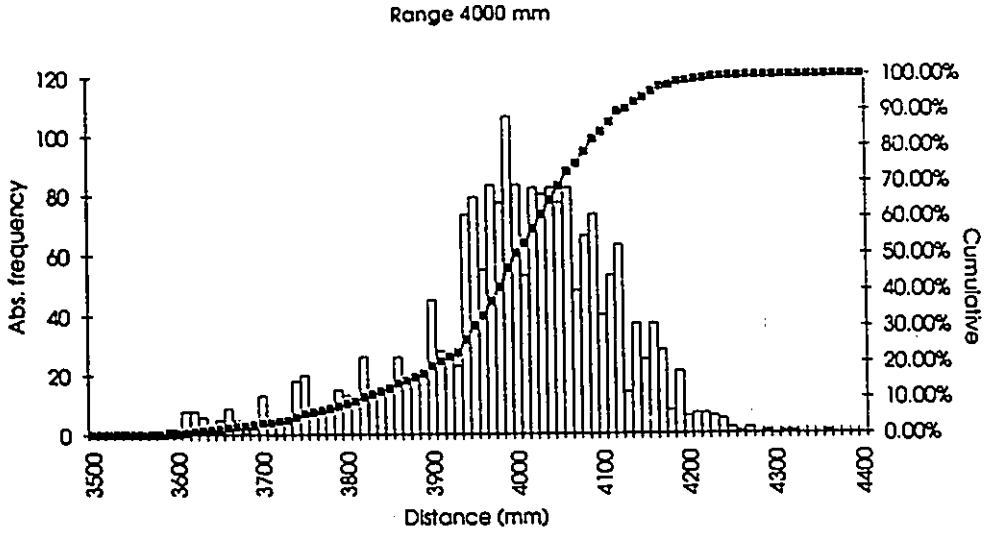


a) histogram for 10 mm step increments

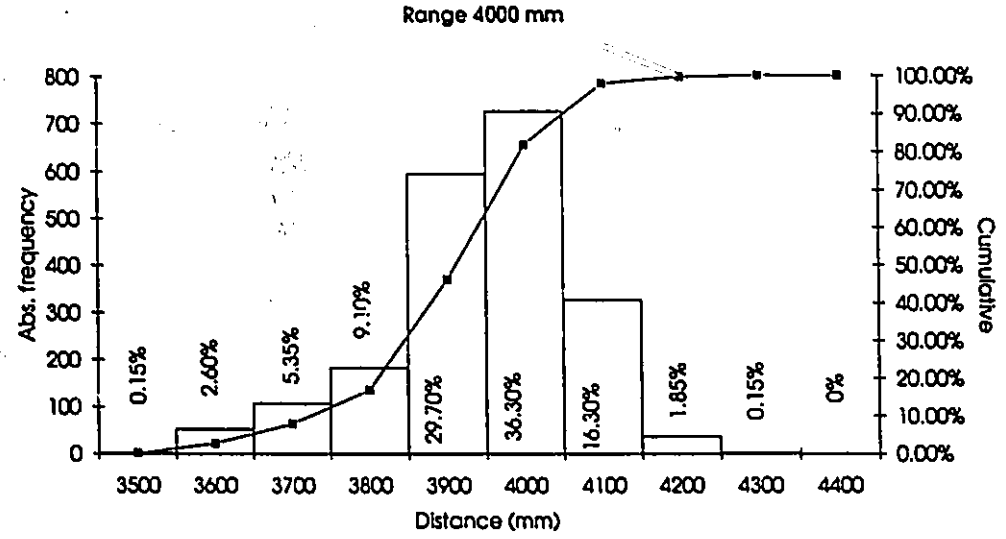


b) histogram for 0.1 m step increments

Figure 3.24. IR sensor probabilistic model for 3.0 m range

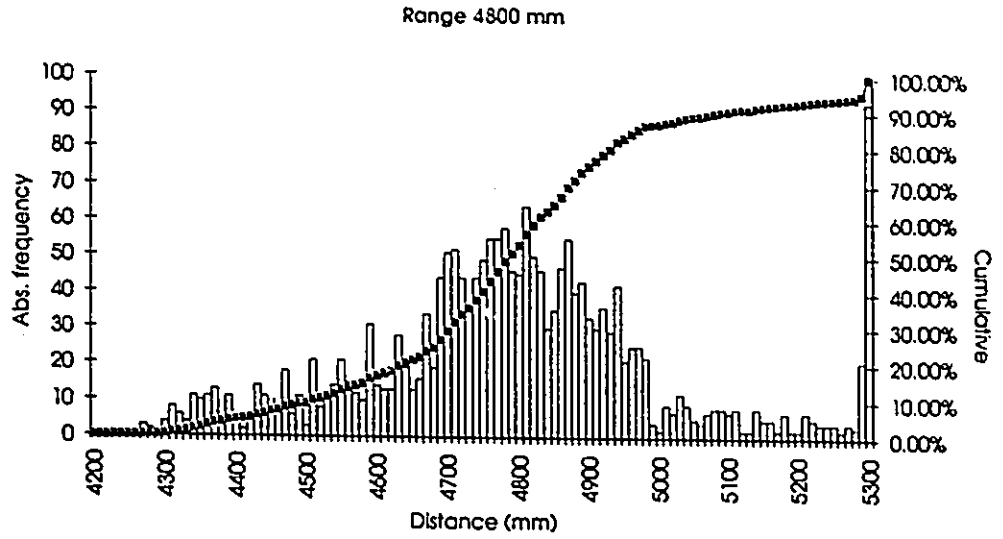


a) histogram for 10 mm step increments

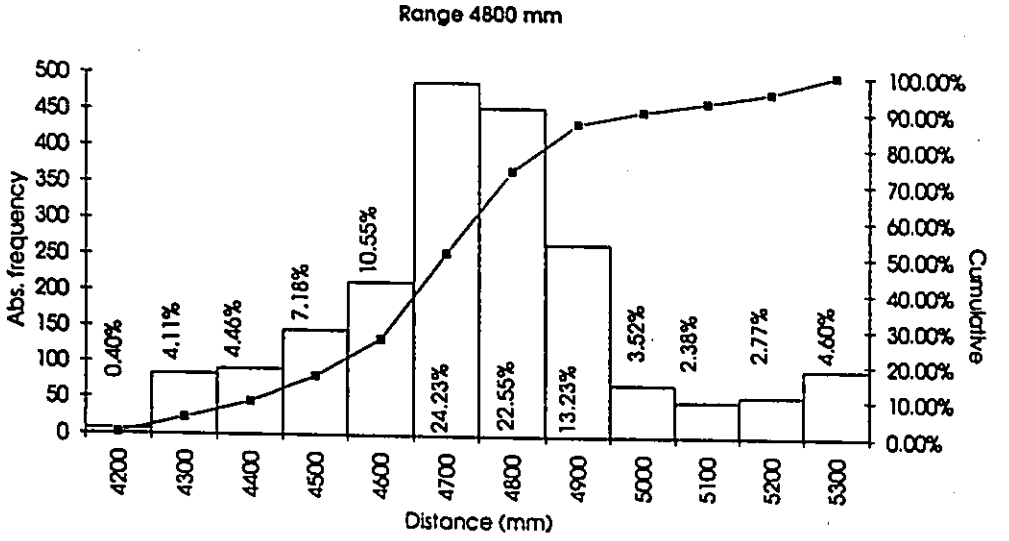


b) histogram for 0.1 m step increments

Figure 3.25. IR sensor probabilistic model for 4.0 m range



a) histogram for 10 mm step increments



b) histogram for 0.1 m step increments

Figure 3.26. IR sensor probabilistic model for 4.8 m range

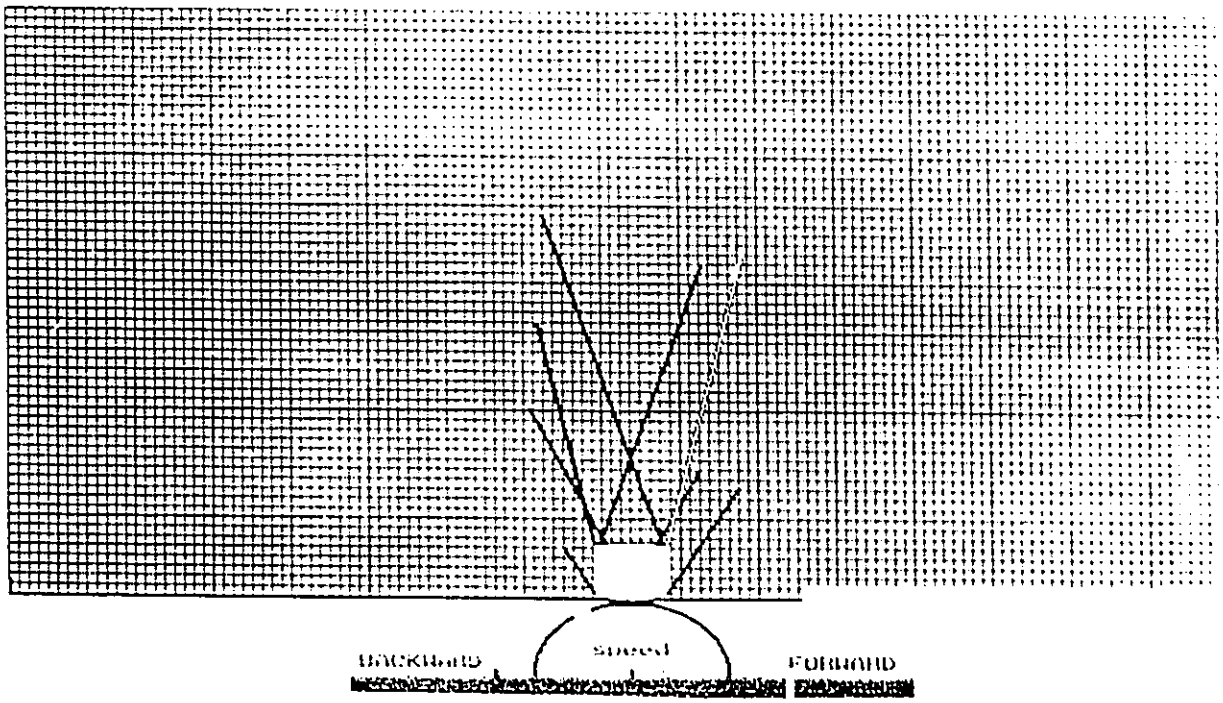
displayed in real time and are not stored "as they are" in the memory, therefore cannot be recalled. But important to remember however is the fact that the information contained in these maps is always used to update the "background" global map. The global map can be anytime recalled by the user, in which case the image on the screen is swapped.

The local maps are always represented having as reference the mobile robot front end. In such a display the bottom part of the screen uses a graphic user interface (GUI) display to show the current speed of the vehicle and the front wheels steering system orientation. An example of such a local map is presented in fig. 3.27.a, b. Using a menu and the arrow keys, the user can interact dynamically with the vehicle, changing its speed and direction (including the option to move forward or backward) and the depth of view in the front. For example, both pictures in fig.3.27 present the same environment, one having a view of 3 m in front of the vehicle, the other 5 m, and the speed and steering position differ.

The problem of updating the global map in real time is a computational intensive one and some concrete, implementation-related problems have to be addressed in applying the Bayesian estimation method. These problems arise mostly as a result of the fact that the sensor orientation on the map can have any value. That means that the sensor probabilistic model would have in general case a different step size than the grid size and a computational method based on that presented in (3.3.3) has to be implemented. In the following I will present briefly the computational method used in this work, making use of an arbitrary, concrete example and will highlight the advantages and disadvantages against other options.

Lets assume a range measurement R_0 having a probabilistic sensory model presented in fig. 3.28. A special mention is needed here, that the sensor's error probability distribution does not necessarily has to be symmetrical with respect to the mean value and from the conducted experiments (fig 3.23-3.26) this hypothesis is confirmed. The distance step increment size (R_i-R_{i-1}) is such to match the global map grid size, e.g. 0.1 m. Referring to the fig. 3.28, the information contained in the probabilistic sensory model is that given a measurement of range R_0 , the actual range can be anywhere in the five cells R_{-2}...R_2 with the corresponding probabilities: .12-.2-.35-

a)



b)

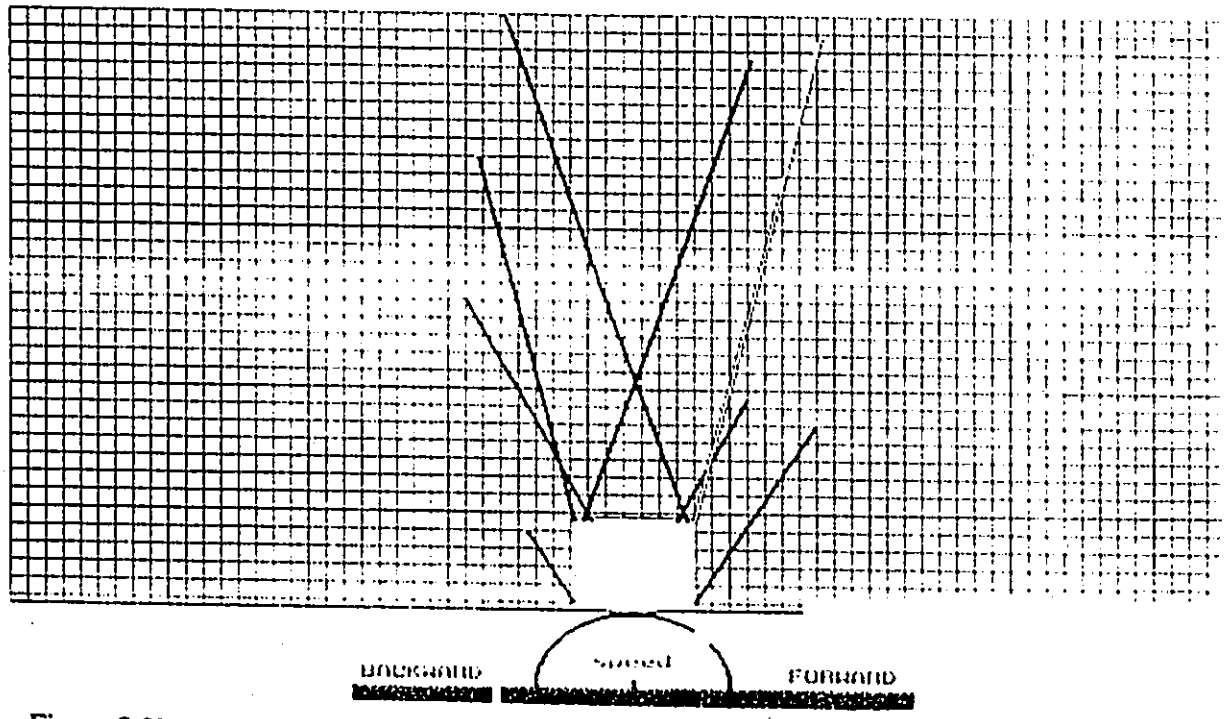


Figure 3.27. Example of a local map display with different settings (range, speed, steering)

Range [m]	min [mm]	average [mm]	max [mm]	No. readings	Standard deviation
1.5	1477	1500.14	1535	2007	6.95
3.0	2800	3002.21	3264	1972	66.17
4.0	3541	3997.12	4365	2030	122.33
4.8	4202	4802.29	5304	2018	219.42

Table 3.1. IR sensor modeling. Experimental results summary

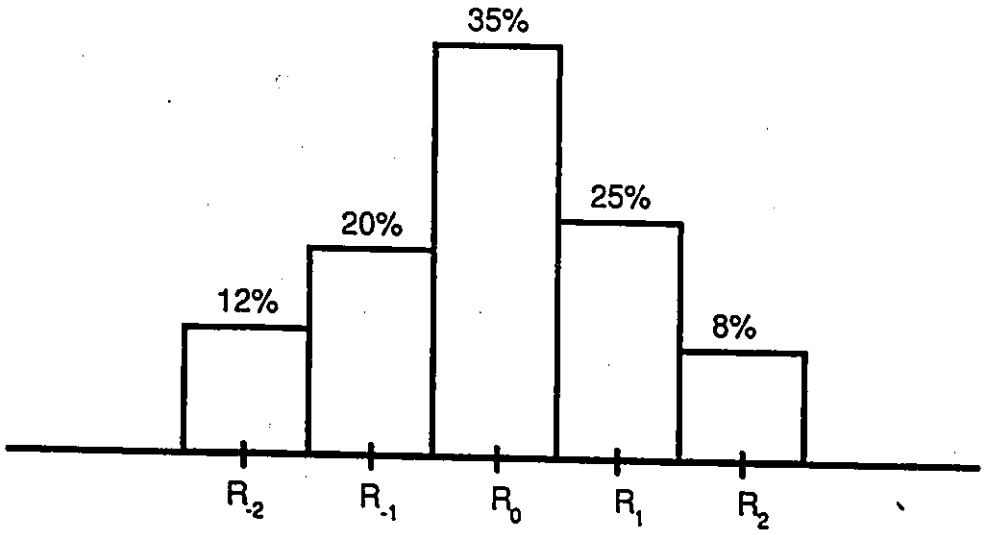


Figure 3.28. Example of a probabilistic sensor model

.25-.08. Being a bounded probability distribution, of course the sum of all probabilities has to be one. Further, lets assume a portion of the global map having the cells labeled by variables *i* and *j* on the rows and columns respectively and a measurement range R_0 of the previously described probabilistic distribution making an angle *alpha* with the rows direction. This scenario is depicted in fig. 3.29, with the dotted lines being perpendiculars on the range direction and going through the intersection points between the range direction and the cell borders.

Each cell that does contain a portion of the sensor probability distribution will be involved in the process of estimating an occupancy probability cell. In the above example, there are a number of six cells involved: (i+2,j+4), (i+2,j+5), (i+2,j+6), (i+2,j+7), (i+3,j+7) and (i+3,j+8). These cells are represented shadowed. With slanting lines are represented the six different portions from the sensor's probability distribution that are assigned to each individual cell. A computational method similar to that exemplified in fig.3.2 will lead to the occupancy probability grid for each cell:

- a) compute the occupancy probability of each cell: $\int_{C_i} f(r|p)dp$
- b) multiply the prior probability value for each cell (in this example 0.5) by the values calculated in the last step
- c) normalize the result to one

The first step (a) of the above method is suggestively represented in fig. 3.30, where superimposed are depicted both distributions (sensor probability - normal, and computed cell occupancy grid - shadowed). It can be seen that the six cells have the following probability occupancy grid: .048--.192-.36-.22-.11-.07. The described method is computationally extensive:

- given the range R_0 , the sensor position on the map and the angle alpha, we have to compute all the intersections points of the range direction with both the horizontal and vertical grid lines, in order to determine all the cells involved;
- although the integral sign reduces to basic rectangles array computation, for each individual cell we have to compute the corresponding integral from the probability

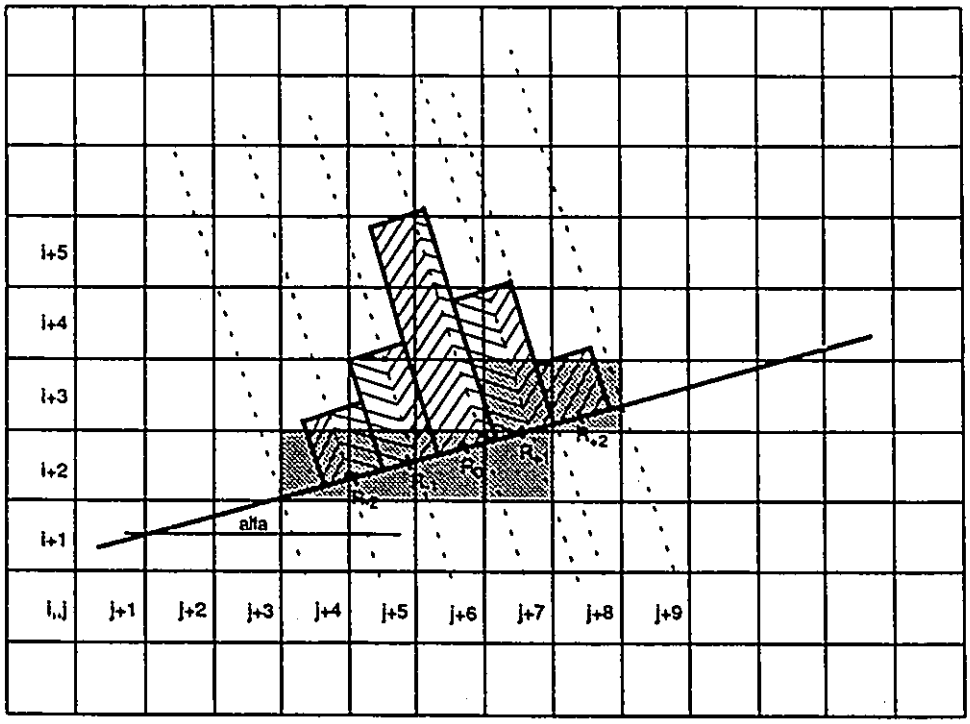


Figure 3.29. Occupancy grid method. Exhaustive computation

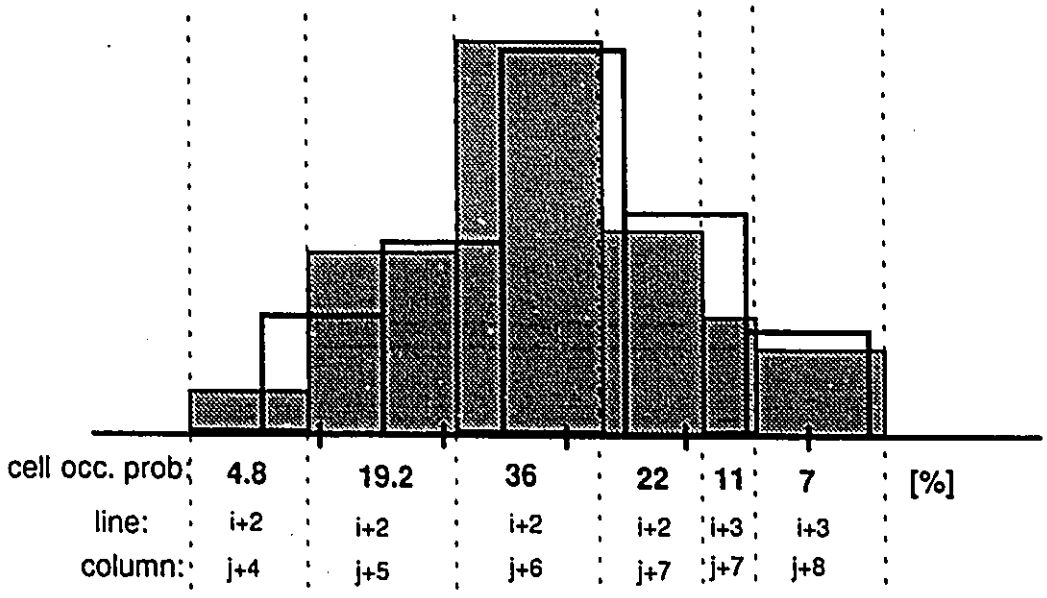


Figure 3.30. Occupancy grid method results

distribution, usually two different shapes (rectangles) being involved, but sometimes even three;

- normalize the result to one;

In order to minimize the computations required only for the first step (a) I proposed a sort of simplified method that was implemented and tested in this work. In the following I will refer to the fig. 3.31 that contains the same hypothesis as the previous described method. The slanting lines become unimportant because the employed method makes no more use of them. Instead we compute the centers of the sensor probability grid points: R_{-2}, R_{-1}, R_0, R_1, R_2. The cell to which the center belongs will be assigned the *whole* corresponding probability occupancy. In the case of a cell containing two centers, for instance (i+2,j+5), both *whole* probabilities are summed. The result is presented in fig. 3.32 and it can be seen that only five cells are involved (the four ones with probability different than zero are represented shadowed in fig. 3.31); the missing one is the cell (i+3,j+7) that is undefined because no sensory grid center is assigned to it. The other five cells are (i+2, j+4), (i+2,j+5), (i+2,j+6), (i+2,j+7) and (i+3,j+8) with the following occupancy probability: 0-.32-.35-.25-.8. The proposed method against the previous one has the following advantages:

- is computationally far less intensive, therefore being much faster, suitable for real time applications;
- instead of computing all the intersection points between the sensor direction and the row and column border lines, we compute only the position of the sensor probability distribution grid center points;
- there is no need for any integral to be computed, the occupancy probability is assigned or at most results as a sum of two terms;

Disadvantages:

- the method introduces errors that has to be addressed in the context of the whole process:
  - sometimes the first cell would have assigned a probability 0 instead the computed one, as in the above example, 0 instead 0.048. The error is lesser as lower the first grid probability is in the sensor probabilistic model; the maximum value for the

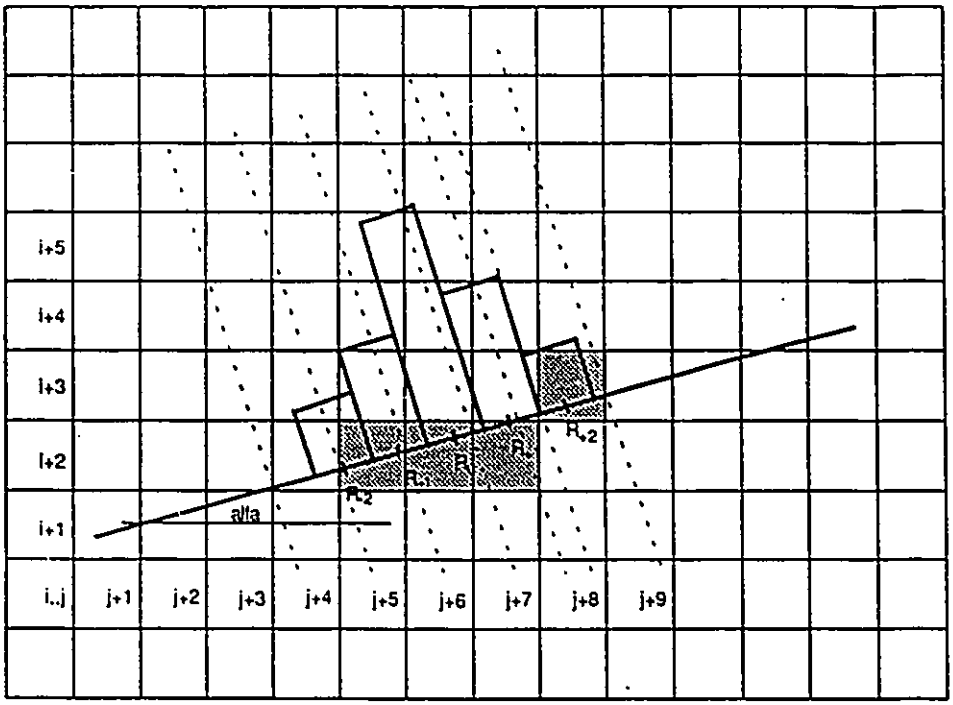


Figure 3.31. Occupancy grid method. Proposed computation

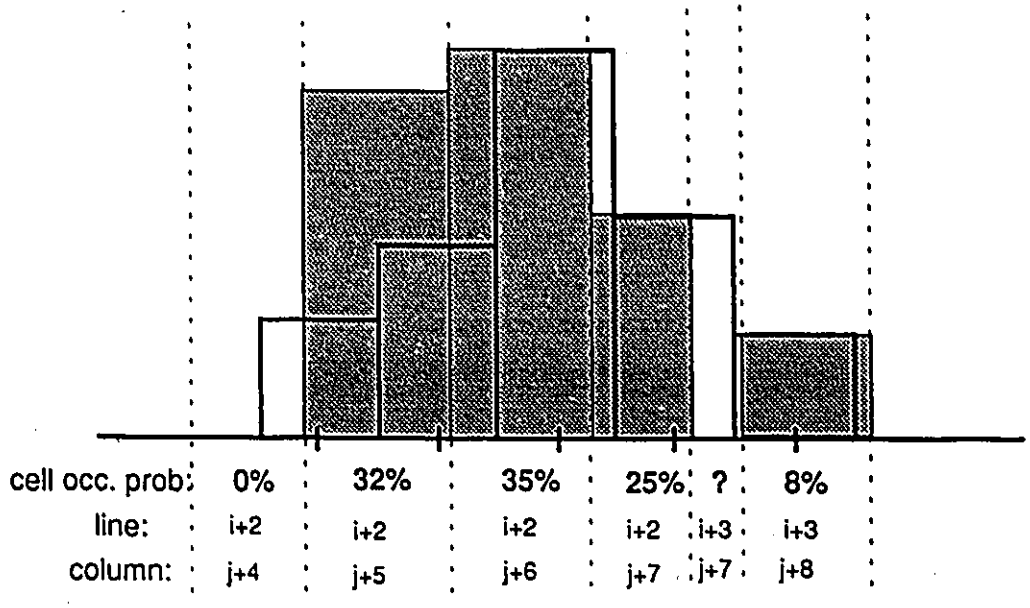
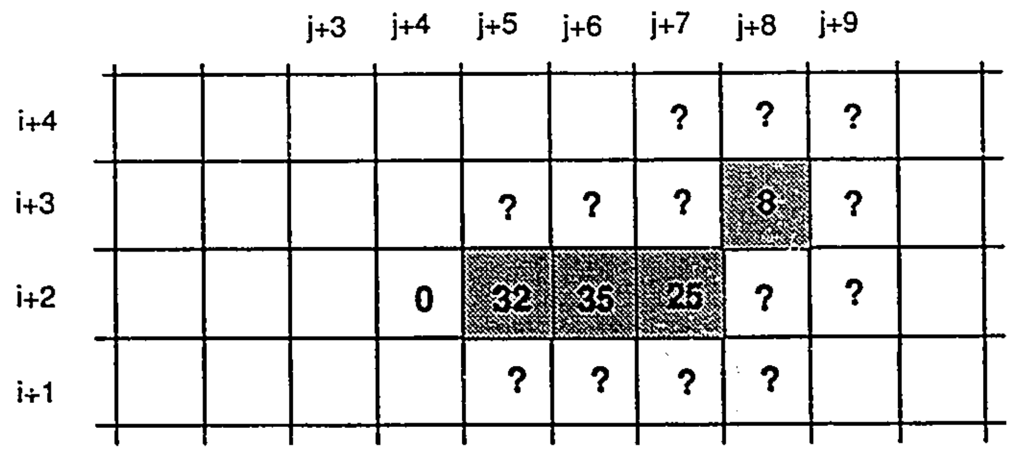


Figure 3.32. The above proposed method's results

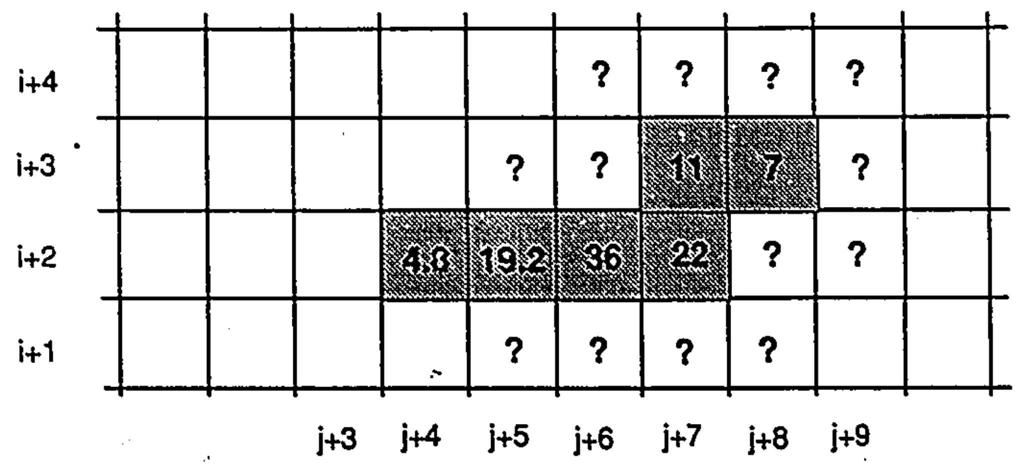
error is half of the first grid probability (in the previous example 0.1 : 2 = 0.05), because otherwise the center would already belong to that cell. This error is not very significant because in our sensor's model the first cell has usually a small probability (the distribution is "flat"): 5.88% for 3 m, 0.15% for 4 m and 4.11% for 4.8 m. The only exception is for the close range with the first cell having 48.74% probability of being occupied and in which case the probability distribution is very narrow (standard deviation is small). But in this case the trade-off is that being involved only two cells, the computational extent of the first, accurate method is not so big and it can be applied.

- sometimes one grid would contain two centers in which case instead of an accurate computation we relay on a rough estimation by adding the two probabilities. The error is smaller as flatter the sensor probability distribution is (example cell (i+2,j+5)).
- sometimes one cell would not contain any center in which case it "skips" from our computation (example cell (i+3,j+7)). The error would be once again maximum half of the corresponding grid assigned probability, in the presented example 0.25 : 2 = 0.125, otherwise a center would belong to that cell.
- always the occupancy probability grid is a "course", assigned one, instead of the computed "finer" one, but with errors always smaller than 50% of one cell probability. Of course that closer this happens to be to the mean value, therefore higher the errors: in our sensors' model the higher cells probabilities are: 43.25% for 3 m, 36.30% for 4 m, and 24.23% for 4.8 m, therefore the theoretical limit for the maximum error would be 21.625%.

The comparation of the two methods is presented in fig. 3.33. It has to be mentioned however that the values represent only the new occupancy probability grid and the Bayesian estimation process hasn't been yet involved (steps b and c). This means that these values have to be multiplied with the priors and the result normalized to one. Because usually the priors have been determined from different views, the computation errors tend to "average" and all the process can be viewed as a random additive noise. Furthermore, the method is usually integrated

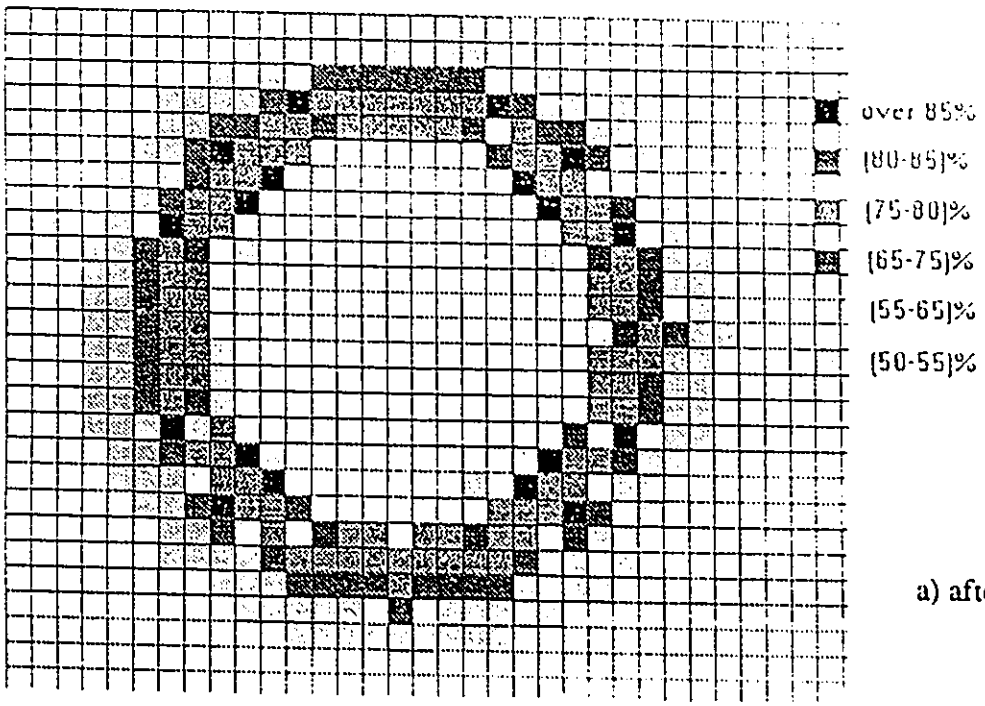


a) proposed one

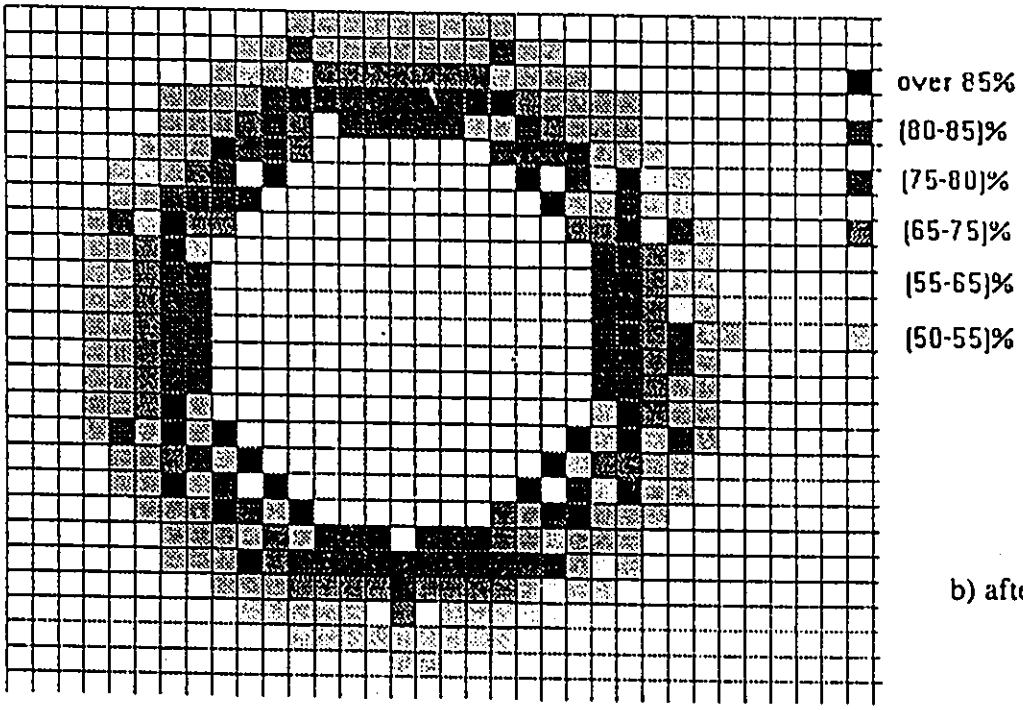


b) exhaustive one

Figure 3.33. Comparation of the two computational methods



a) after two turns



b) after 10 turns

Figure 3.34. Occupancy grid map of a hollow cylindrical obstacle measured with a rotary IR sensor (simulation)

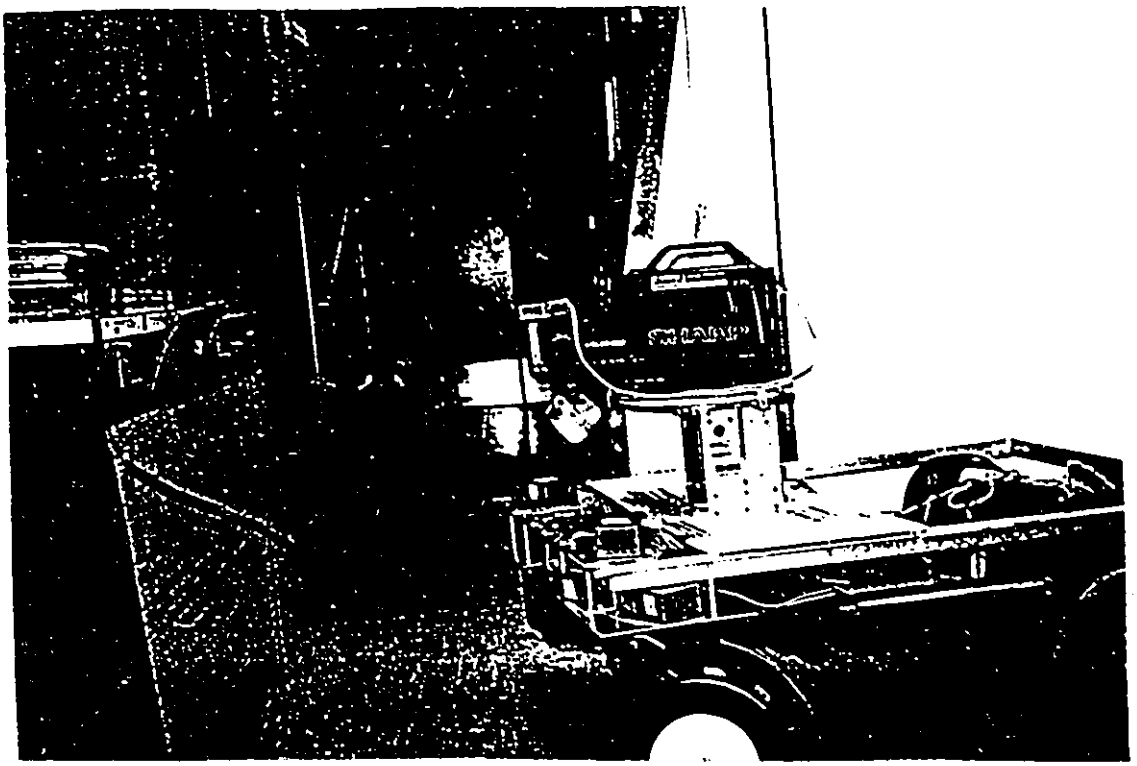
with other sensing features (vision for instance) and using correlation methods corrections can be performed. This approach was implemented in a simulation experiment described in the following and in a real life test setup described in the following paragraph. In Appendix C is presented the software algorithm for the complete Bayesian estimation method and the source code of the involved subroutines.

For the simulation purposes I imagined a IR sensor with a symmetric probability distribution on 9 grids: 0.01-0.04-0.1-0.2-0.3-0.2-0.1-0.04-0.01. The sensor would rotate with a constant speed and measure a hollow cylindrical obstacle which center coincide with the sensor position. The cylinder radius is 1 m and the samples are taken each five degrees. The prior probability distribution for all cells is 0.5, therefore no prior information is known. Fig. 3.34 .a presents the occupancy probability grid after 2 turns, while fig. 3.34.b presents the updated map after 10 turns. It can be seen that although the described method errors were emphasized due to the fact that the "point of view" remained unchanged, the result is accurate enough, behind the darker cells with higher occupancy probability the other cells occupancy probability tends smoothly with the distance toward 0.5, because the state of these cells become unknown, they being "masked" by the obstacle surface (see 3.3).

3.4.6. Experimental Scene Set-Up. Results

In figure 3.35 is presented a room layout occupancy grid map. The grid size is 0.1 m. The mobile robot explores the scene and local views are fused into a global map of the environment. This fusion is done using dead reckoning information (from the vehicle's steering and wheel encoders) to registrate the sensor view maps. Fig. 3.36 shows the resulting global probability occupancy grid for the room. This occupancy grid has been segmented by a 60% threshold decision rule yielding the binary image shown in Fig. 3.37.

In this experiment, the mobile robot navigation control is based on the shared control concept in which the human teleoperator using (virtual reality) visual feedback is enclosed in the high level and low band-width control loop. A "teleoperator-aided sensor fusion" feature has also



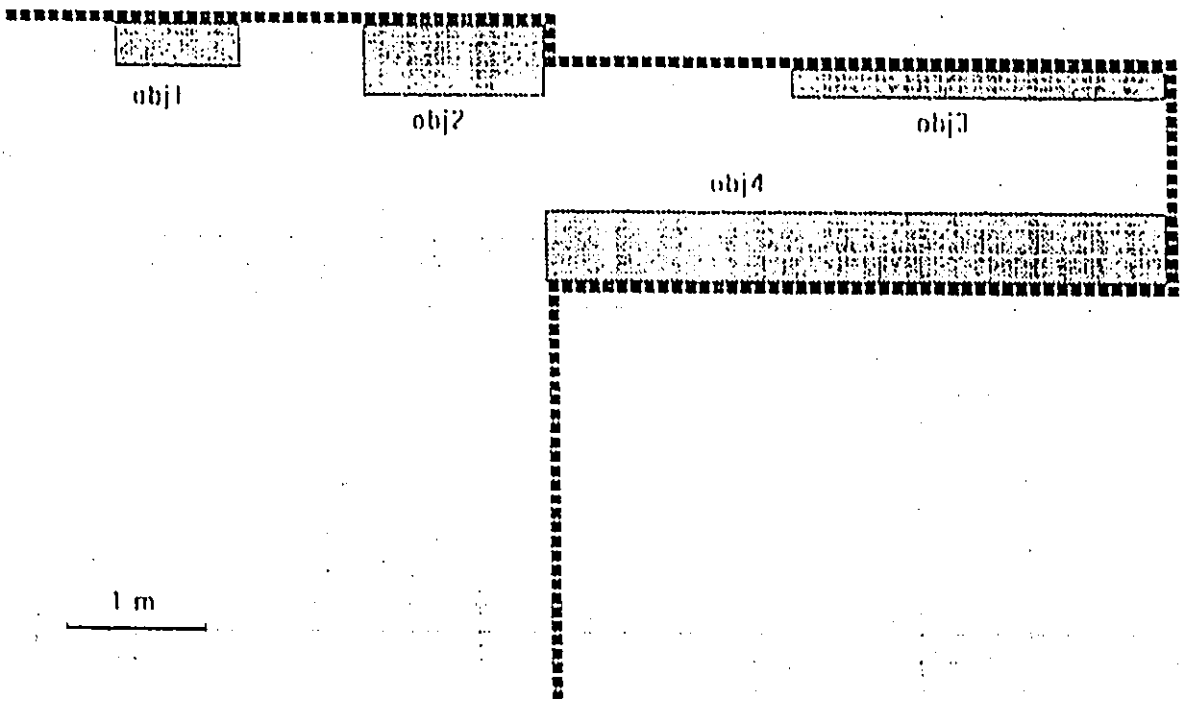


Figure 3.35. Actual photo and layout of the scene set-up experiment

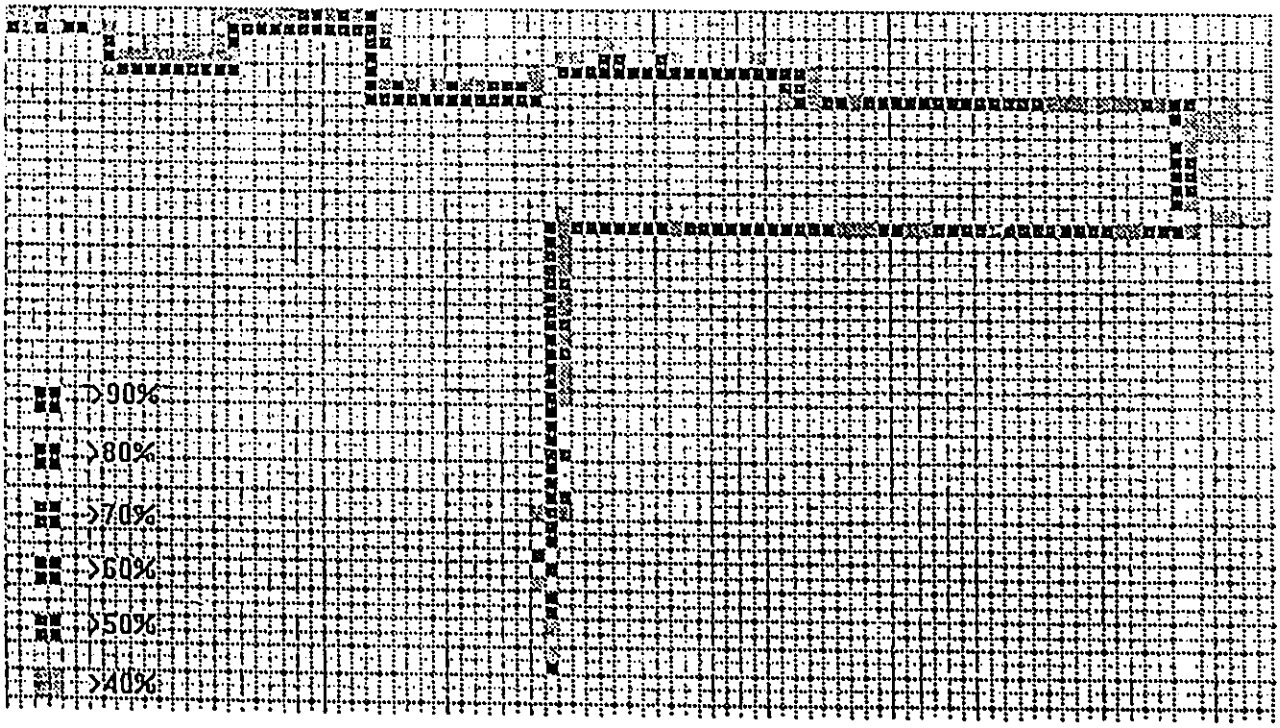


Figure 3.36. Obtained probability occupancy grid global map of the scene

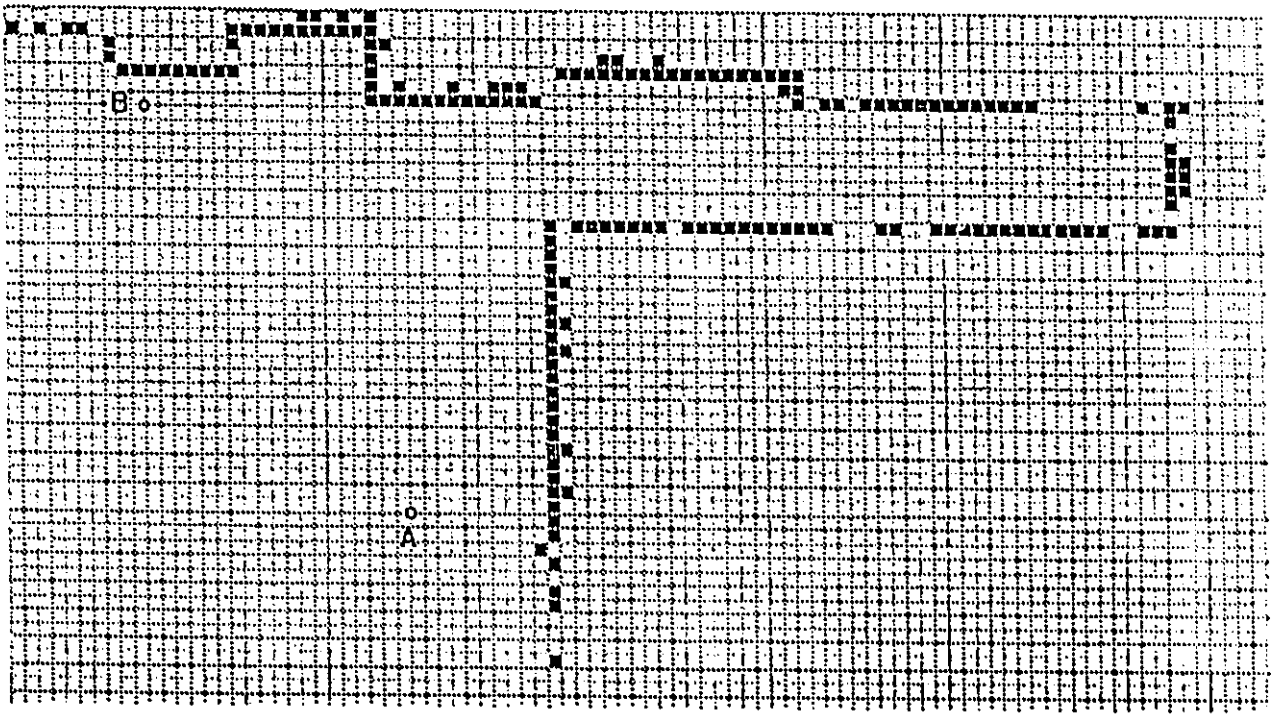


Figure 3.37. Binary image of the scene obtained by thesholding the gray image of Fig. 3.36

been incorporated to investigate the use of the (superior) human image analysis skills to interactively generate world models for mobile robot navigation. In this concept the teleoperator "manually" integrates raster images received from the onboard auto-focus video camera with a IR range finder. The range information which accompanies each image and the robot's position sensors provide a certain degree of POSE (Position, Orientation, and Scale Estimation) indexing for these images. This information helps the teleoperator to combine local snapshots in a composite image of the environment as seen from a given vantage-point of the robot. This raster image fusion process is illustrated in fig. 3.38 showing four snapshots which were integrated to produce a composite image of the environment from the vantage-point called "A". Similarly, fig. 3.39 shows two snapshots which were integrated to produce a composite image of the environment from the vantage-point called "B".

Following a raster-image / graphic-map principle, the teleoperator uses computer graphics functions provided by the interactive visual display unit to generate a wire-frame sketch of the visualized objects by drawing lines over the object features of interest apparent in the composite raster image. Fig. 3.40 shows the resulting wire-frame sketches obtained from the two composite images from fig. 3.38 and fig. 3.39.

The teleoperator then identifies common vertices in the wire-frame models. This identification provides a *de facto* connection of all vertices thought to belong to the same object. It results into a 2-D partial wire-frame model of that object. The lengths of all vertex-limited segments are automatically calculated by the computer graphic environment from their relative positions in the 2-D image and the POSE parameters of that image.

Based on the teleoperator interpretation of the composite raster images of an object seen from different vantage-points, the teleoperator assigns (fig. 3.41) the ground based vertices of the wire-frame models to specific locations on the displayed 2-D occupancy grid of the environment. This assignment / mapping gives a 3-D position at least for the wire-frame model vertices laying on the ground. The 3-D position of the above-ground vertices is then inferred, as

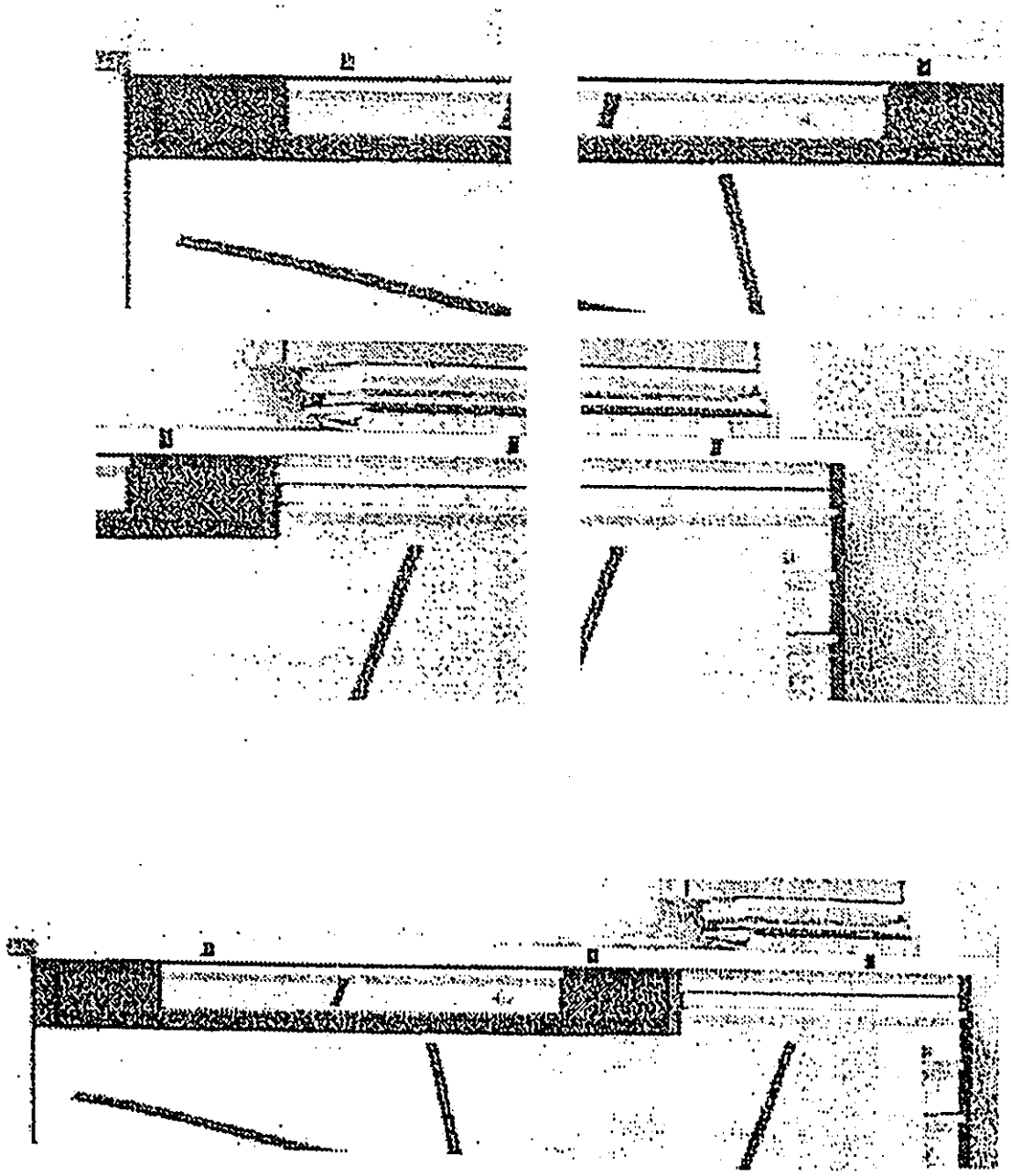


Figure 3.38. Partially overlapping snapshots taken from the vantage point "A" (see Fig.3.37) and the composite image obtained by their "teleoperator-aided" fusion

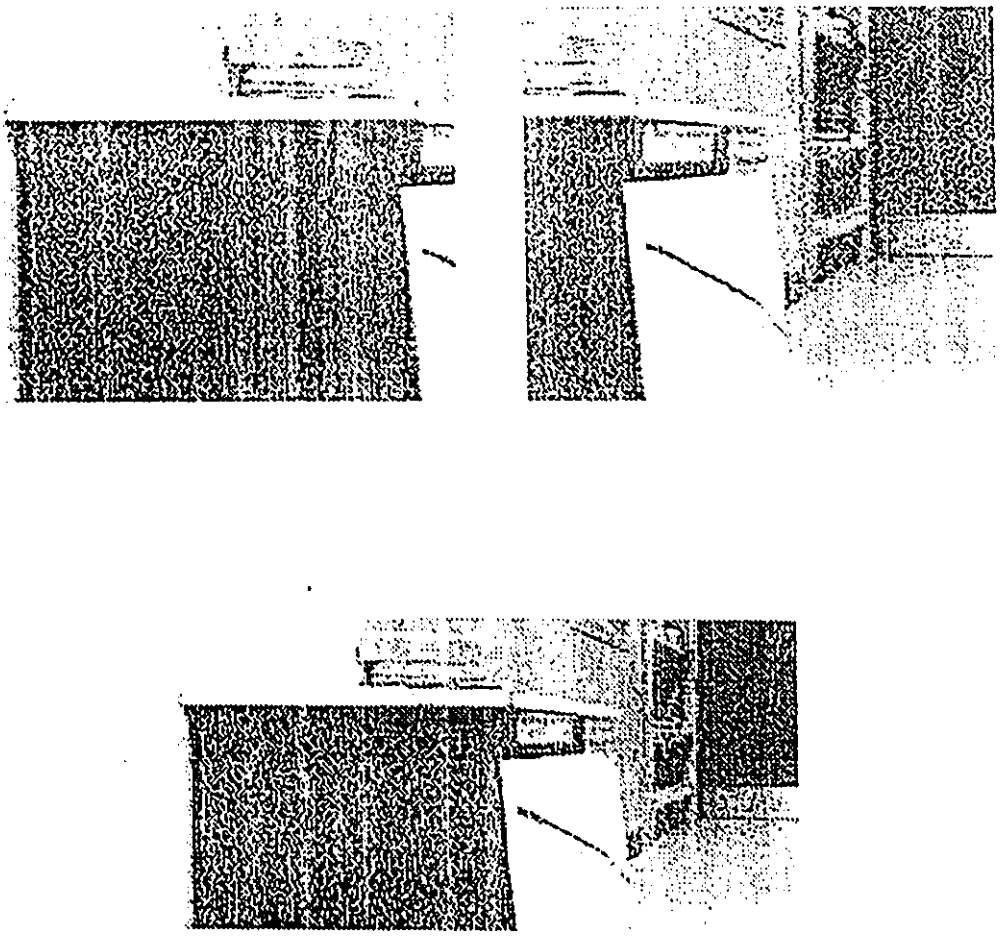


Figure 3.39. Partially overlapping snapshots taken from the vantage point "B" (see Fig.3.37) and the composite image obtained by their "teleoperator-aided" fusion

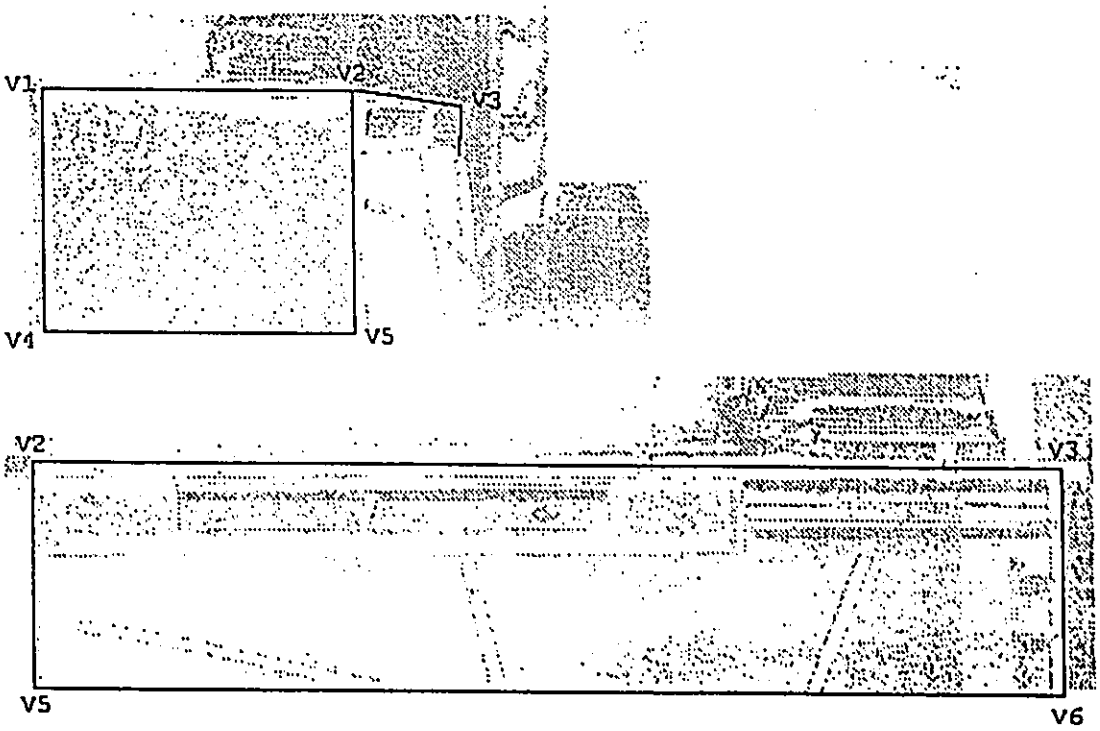


Figure 3.40. Teleoperator generated wire frame sketch of objects deemed of interest in Fig.3.38 and respectively in Fig. 3.39

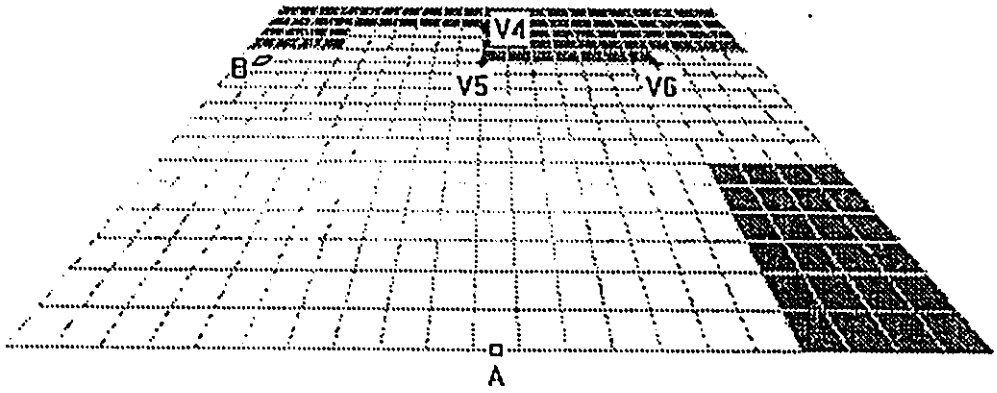


Figure 3.41. The teleoperator assigns wire-frame model vertices to specific locations on the 2-D occupancy grid

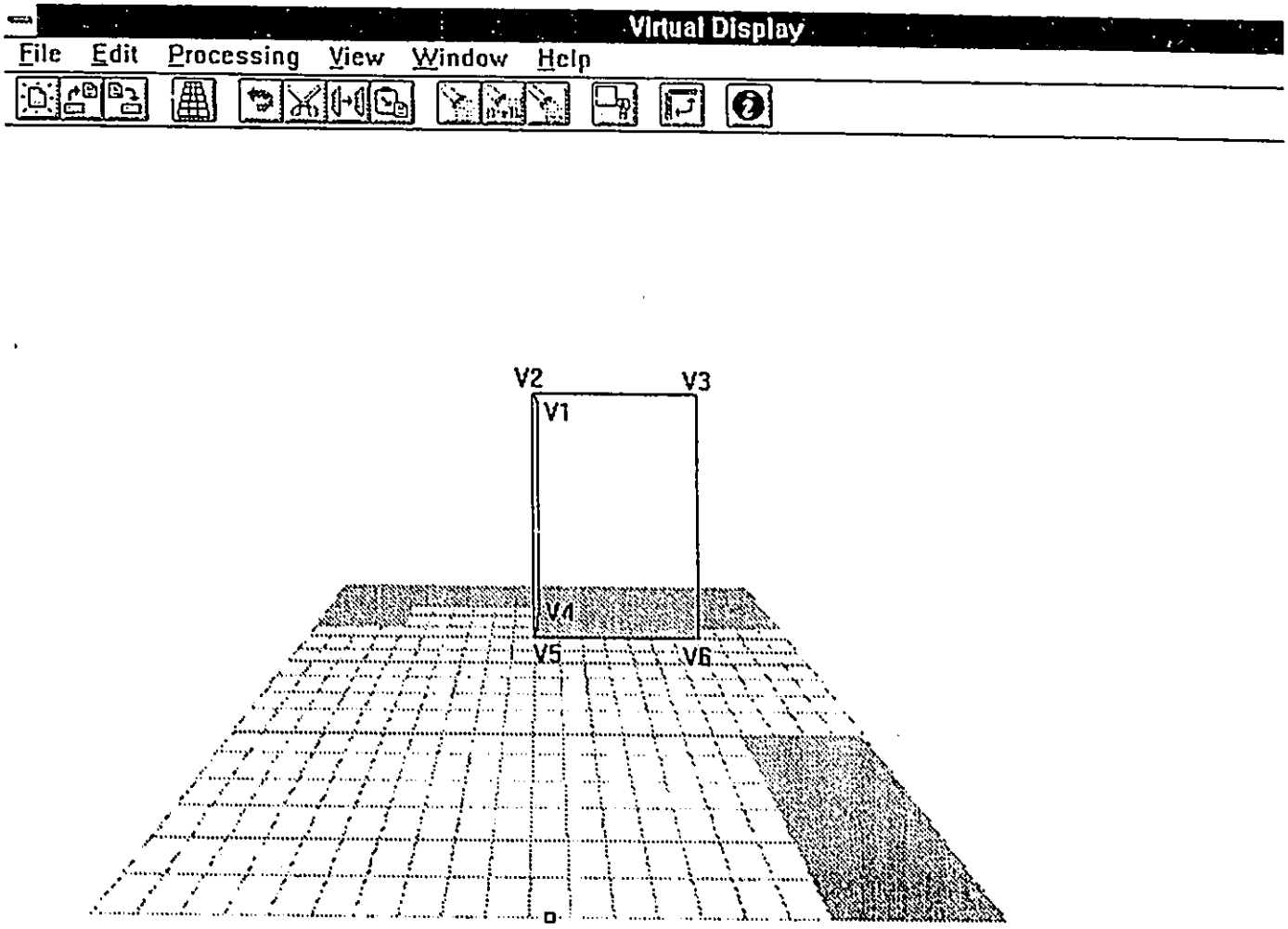


Figure 3.42. Resulting 3-D vertex positions for the wire-frame model previously placed on the occupancy grid

illustrated in fig. 3.42., by the teleoperator from the specific shape of the polygons seen from different point of views.

The resulting 3-D wire-frame model is ready to be used as virtual reality feedback for the telecontrol of the mobile robot. It can also be sent to the on-board computer to be used for autonomous navigation.

The interactive virtual reality display allows the teleoperator to choose between the purely geometric 3-D model of an object and the raster image of that object, as shown in fig. 3.43. The teleoperator has also the possibility to inquire about the position of any object on the displayed occupancy grid, as illustrated in fig. 3.44. The teleoperator can change these parameters, which result in an instant modification of the world model.

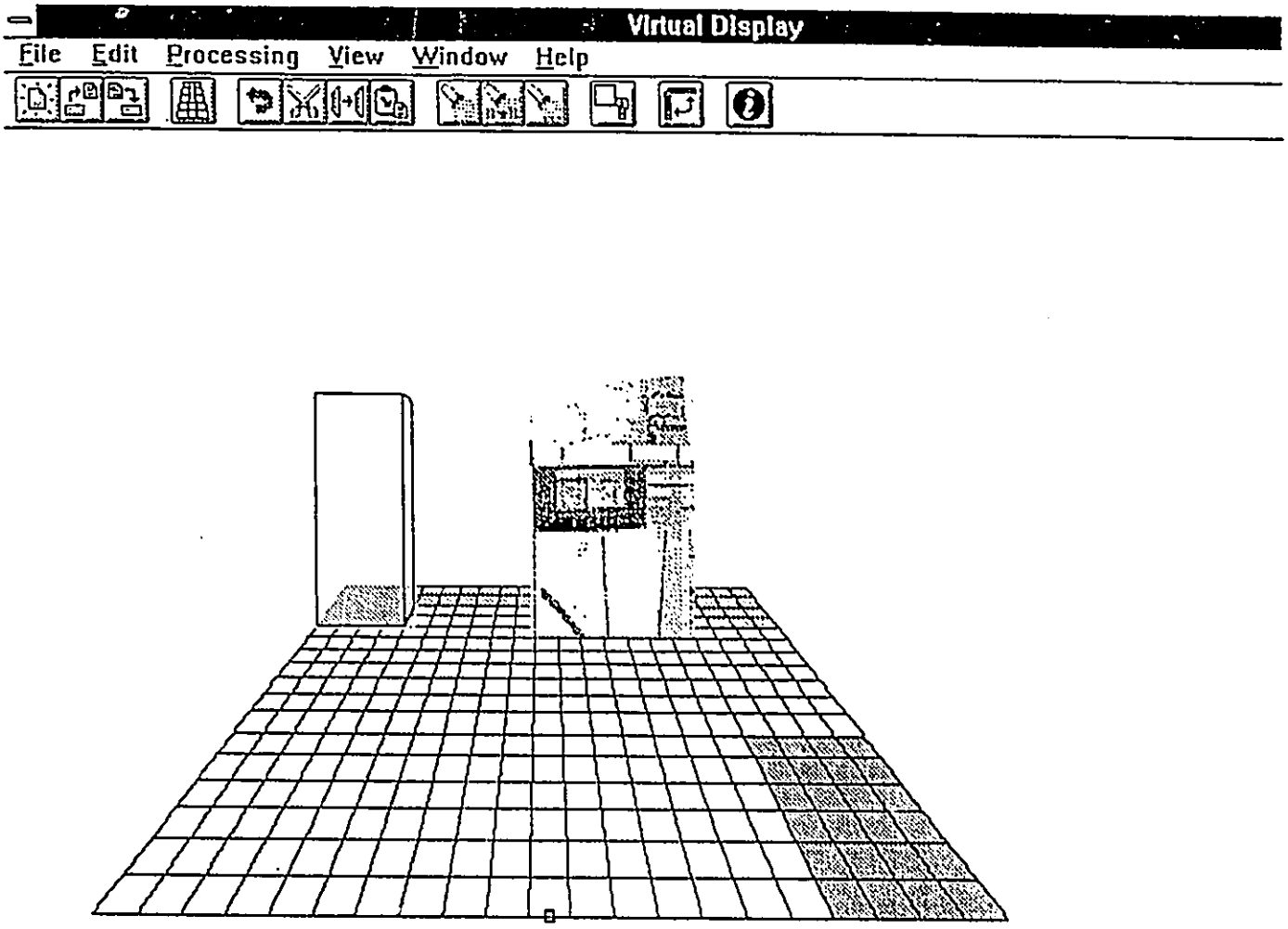


Figure 3.43. Pseudo 3-D display of the explored room containing wire-frame models and raster images of objects

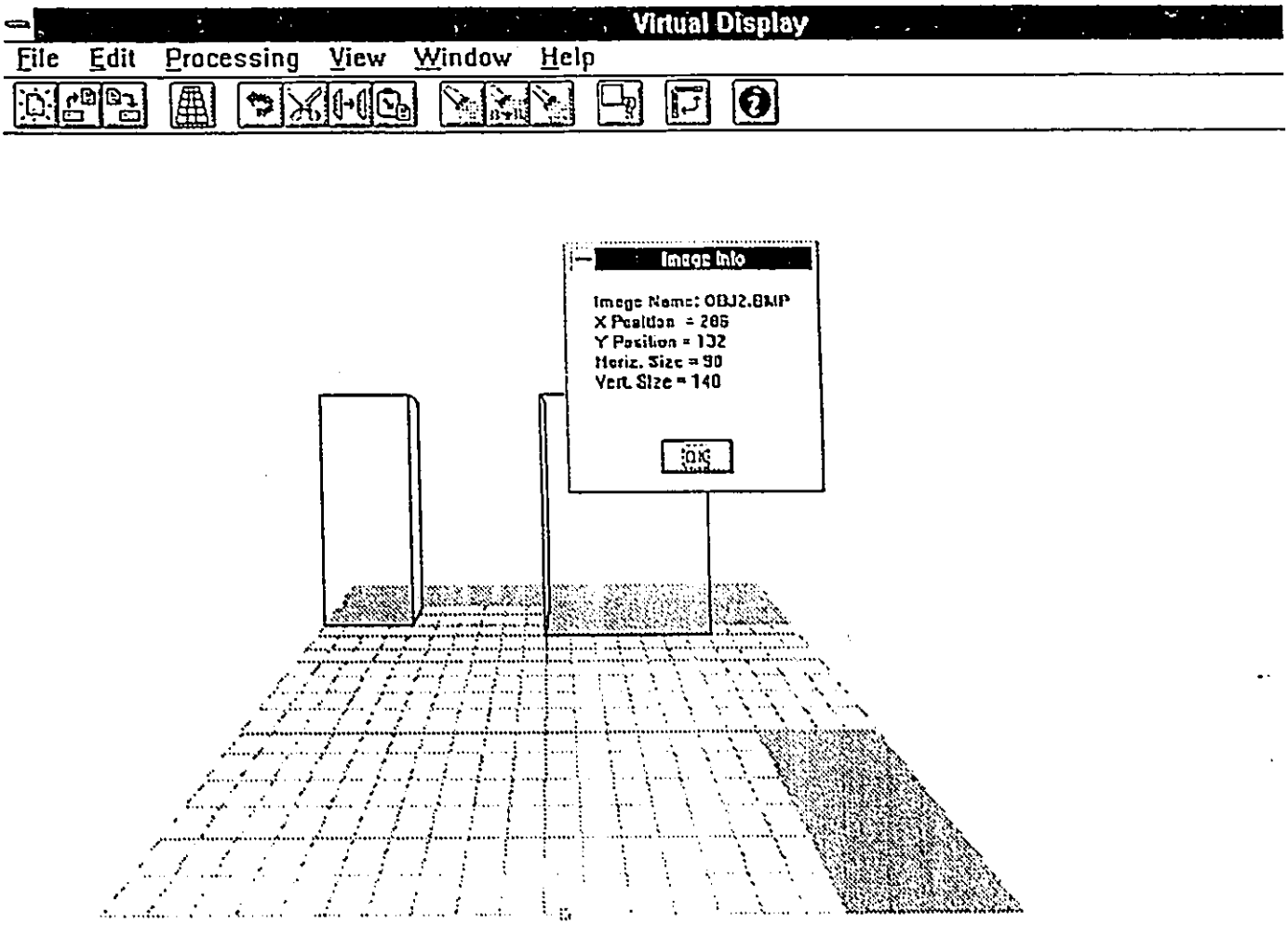


Figure 3.44. The computer graphics software environment provides on request position information about any object represented in the 3-D world model and allows the teleoperator to move objects in the scene by modifying their position parameters in the object information window

Part 4

CONCLUSION

This thesis has been presented an example of solving the data fusion problem in the case of infrared sensors. The occupancy grid method was used, method that provides a general framework for sensor integration. The method is based on the Bayesian statistical analysis, and therefore requires a statistical sensor model. In the experimental part, an example of teleoperator-aided sensor fusion for video images with IR sensory information has been presented, that produces a 3-D model of the objects related to the global occupancy grid map. This is an interactive virtual reality application limited to teleoperator controlled navigation function. The personal contributions to this work and presented in the thesis are:

- the design and testing of the electronic and software interface with the mobile robot for navigation purposes;
- the IR sensor calibration and modeling;
- the Bayesian statistical analysis method implementation for mobile robot mapping and navigation;
- a proposed method for computing the occupancy grid based on the obstacle surface probability range distribution;
- a proposed method for the occupancy grid map updating that allows real-time data aquisition and processing;
- a GUI presentation for mobile robot mapping and navigation;

Further research and experiments should be directed in this areas:

- an improved experimental mobile robot platform, that would allow finer moves and increased position recovery (a new platform is being currently developed);

- other range sensor integration (e.g. rotary) that would lead to the increase of information redundancy with benefits in error reduction;
- a 2-D correlation of the sensor maps containing views of the same landmarks that would reduce the registration errors inherent of dead reckoning position estimation;
- other task applications: autonomous navigation, object manipulation

BIBLIOGRAPHY

[1] E.Petriu, G. Costache, N. Trif, M. Colven, *C. Gal,* "Development of a Multi-Sensor Data Fusion and Interpretation System", Contract #9F011-3-0947, **May 1993-March 1994**

[2] S.K. Yeung, W.S. McMath, E. Petriu, *C. Gal,* L.Korba, S. Elgazzar, I.C. Dancea, "Multi-Sensor System for Mobile Robot Navigation", Proc. IEEE & SAE Intelligent Vehicles '93 Symposium, pp.455-460, Tokyo, **July 14-16, 1993**

[3] E. Petriu, *C. Gal,* N. Trif, V. Trif, "Development of a multisensor fusion system for space application", Contract #9F011-2-0923, **June 1992 - March 1993**

[4] L. Korba, S. Elgazzar, T. Welch, "Active Infrared Sensors for Robotics", Proc.IMTC/93, IEEE Instrum. Meas. Technol. Conf., Irvine, CA, **1993**

[5] J. Forsberg, U. Larsson, P. Ahman, A. Wernersson, "Navigation in Cluttered Rooms Using a Range Measuring Laser and the Hough Transform", AS-3, pp 248-257, Pittsburgh, **1993**

[6] L. Korba, "Command Summary for the Infrared Sensor Controller", NRC document, Nov. **1992**

[7] *** National Instruments: "Lab-PC. User Manual", **Sept.1992**

[8] A. Zelinsky, A Mobile Robot Exploration Algorithm", IEEE Transactions on Robotics and Automation, Vol.8, No.6, pp 707-717, **December 1992**

[9] Y. Fujii, D.K. Wehe, T.E. Weymouth, "Sub-step Infrared Range Estimation for a Mobile Robot", Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp 2041-2046, Raleigh, N.C., **July 7-10, 1992**

[10] J. Evans, B. Krishnamurthy, B. Barrows, T. Skewis, V Lumeski, "Handling Real-World Motion Planning: A Hospital Transport Robot", IEEE Control Systems, pp 15-19, **February 1992**

[11] G.Hager, M.Mintz, "Computational Methods for Task-directed Sensor Data Fusion and Sensor Planing", The International Journal of Robotics Research, Vol.10, No.4, pp 285-312, **August 1991**

[12] J.J. Leonard, H.F.Durrant-White, "Simultaneous Map Building and Localization for an Autonomous Mobile Robot", IEEE/RSJ International Workshop on Intelligent Robots and Systems IROS '91, pp 1442-1447, Osaka, Japan, **November 3-5, 1991**

[13] P. Moyrtalier, R. Chatila, "Incrementeal Free-Space Modelling from Uncertain Data by an Autonomous Mobile Robot", IEEE/RSJ International Workshop on Intelligent Robots and Systems IROS '91, pp 1052-1058, Osaka, Japan, **November 3-5, 1991**

[14] E. Drakopoulos, C.C. Lee, "Optimum Multisensor Fusion of Correlated Local Decisions", IEEE Transactions on Aerospace and Electronic Systems, Vol.27, No.4, **July 1991**

[15] E.B. Hall, A.E. Wessel, G.L. Wise, "Some Aspects of Fusion in Estimation Theory", IEEE Transactions on Information Theory, Vol.37, No.2, **March 1991**

[16] G. D. Hager, "Task-Directed Sensor Fusion and Planning. A Computational Approach", Kluwer Academic Publishers, **1990**

[17] *** Motorola Semiconductor: AN1078 and MC33033/D Technical Sheet, **1990**

[18] A. Elfes, "Using Occupancy Grids for Mobile Robot Perception and Navigation", Computer, pp 46-57, **June 1989**

[19] R.C. Luo, M.G. Kay, "Multisensor Integration and Fusion in Intelligent Systems", IEEE Transaction on Systems, Man, Cybernetics, Vol.19, No.5, pp 901-931, **September/October 1989**

[20] J. Borenstein, Y. Koren, "Real-Time Obstacle Avoidance for Fast Mobile Robots", IEEE Transactions on Systems, Man and Cybernetics, pp 1179-1187, Vol.19, No.5, **September/October 1989**

[21] H.F. Durrant-White, "Sensor Models and Multisensor Integration", The International Journal of Robotics Research, Vol.7, No.6, pp. 97-113, **1988**

[22] J.M. Richardson, K.A. Marsh, "Fusion of Multisensor Data", The International Journal of Robotics Research, Vol.7, No.6, pp. 78-96, **1988**

[23] R.C. Luo, M. Lin, R.S. Sherp, "Dynamic Multi-Sensor Data Fusion System for Intelligent Robots", IEEE Journal of Robotics and Automation, vol.RA-4, no.4, pp. 386-396, **1988**

[24] T. Henderson, E. Wetz, C. Hansen, A. Mitiche, "Multisensor Knowledge Systems: Interpreting 3-D Structure", International Journal of Robotics Research, vol. 7, no. 6, pp.114-137, **1988**

[25] J.L. Crowley, "Coordination of Action and Perception in a Surveillance Robot", IEEE Expert, pp 32-43, **Winter, 1987**

[26] R.C.Luo, M.G. Kay, "Multisensor Integration and Fusion: Issues and Approaches", Proc. SPIE, Vol. 931, Sensor Fusion, C.W. Weaver, Ed. Orlando, pp.83-90, Orlando, FL, **May 1987**

[27] A. Elfes, "Sonar-Based Real-World Mapping and Navigation", IEEE Journal of Robotics and Automation, pp 249-265, Vol. RA-3, No.3, **June 1987**

[28] H.F.Durrant-Whyte, "Consistent Integration and Propagation of Disparate Sensor Observations", The International Journal of Robotics Research, Vol.6, No.3, pp 3-24, **Fall 1987**

[29] J.S. Albus, R. Lumia, H.G. McCain, "NASA / NBS Standard Reference Model for Telerobot Control System Architecture (NASREM)", NASA Document SS-GSFC-0027, **6/18/87**

[30] H.P. Moravec, "Sensor Fusion in Certainty Grids for Mobile Robots", 1987 Annual Research Review of the Carnegie-Mellon Robotics Institute, pp. 34-46, **1987**

[31] T.L. Huntsberger, S.N. Jayaramamurthy, "A Framework for Multi-Sensor Fusion in the Presence of Uncertainty", Proc. Workshop Spatial Reasoning and Multi-Sensor Fusion, St. Charles, IL, pp.345-350, **October 1987**

[32] W.T.Miller, "Sensor-Based Control of Robotic Manipulator Using a General Learning Algorithm", IEEE Journal of Robotics and Automation, vol RA-3, no.2, pp.157-165, **1987**

[33] S.Chen, "Multi-Sensor Fusion and Navigation of Mobile Robots", International Journal Intelligent Systems, vol.2, no.2, pp. 227-251, **1987**

[34] A.R. de Saint Vincent, "A 3-D Perception System for the Mobile Robot HILARE", in Proc. IEEE Int. Conf. Robotics and Automat., pp.1105-1111, April, **1986**

[35] S.Y. Harmon, G.L.Bianchini, B.E.Pinz, "Sensor Data Fusion Through a Distributed Blackboard", Proc. IEEE International Conference on Robotics and Automation, San Francisco, CA, pp. 1449-1454, **1986**

[36] D.T. Lawton, T.S. Levitt, C. McConnell, J. Glicksman, "Terrain Models for Autonomous Land Vehicle", Proc. IEEE International Conference on Robotics and Automation, pp. 2043-2051, San Francisco, CA, **April 1986**

[37] S.J. Kamat, "Value Function Structure for Multiple Sensor Integration", Proc. SPIE, vol.579, Intelligent Robots and Computer Vision, pp. 432-435, Cambridge, MA, **September 1985**

[38] J. O. Berger, "Statistical Decision Theory and Bayesian Analysis", ed 2, New-York: Springer-Verlag, **1985**

[39] T. Kohonen, "Self Organization and Associative Memory", Berlin, Germany, Springer-Verlag, **1984**

[40] T.Henderson, E. Shilcrat, "Logical Sensor Systems", Journal of Robotics Systems", vol.1, no. 2, pp. 169-193, **1984**

[41] T. Henderson, W.S. Fai, C. Hansen, "MKS: A Multi-Sensor Kernel System", ", IEEE Transaction on Systems, Man, Cybernetics, Vol.SMC-14, No.5, pp 784-791, **1984**

[42] P.S. Maybeck, "Stochastic Models, Estimation, and Controls", New York: Academic, vol1 and 2, **1979 and 1982**

[43] A.K. Bejczy, "Sensors, Controls, and Man-Machine Interface for Advanced Teleoperation", Science, Vol. 208, no. 4450, pp. 1327-1335, **1980**

[44] G. Shafer, "A Mathematical Theory of Evidence", Princeton, NJ: Univ. Press, **1976**

APPENDIX A

This program **shows how to compute** the probability occupancy distribution for any kind of probability distribution and **proves** that the proposed method can lead a repeated gaussian distribution to the behaviour of an ideal sensor.

Let's assume an obstacle at the range x=2.5m. A distinction has to be made between the *obstacle surface range probability distribution* and the *cell occupancy probability distribution*.

1) *Ideal sensor*

In the case of an ideal sensor, the obstacle surface probability distribution will lead to all cells having probability zero, except the one at x=2.5m with probability one. But because beyond the obstacle's surface the occupancy state is unknown (probability 0.5), the cell occupancy probability has a different distribution:

-0 for x<2.5; 1 for x=2.5; 0.5 for x.2.5

Example:

$j := 0..25$ j, *index*

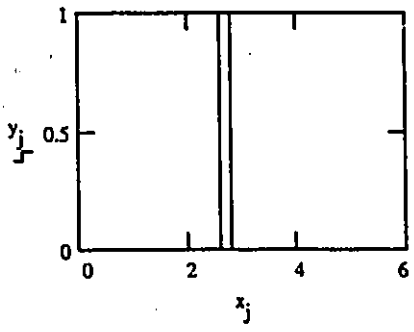
step := .2 step, *grid size*

$x_j := j \cdot step$ *cuantized distance*

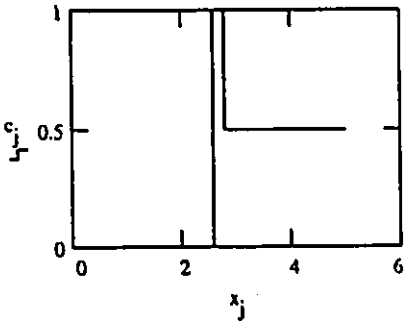
$y_j := if(j \neq 13, 0, 1)$ *definition for the ideal sensor obstacle surface probability distribution*

$c_j := if(j \leq 12, 0, .5)$ *definition for ideal sensor occupancy grid distribution*

$c_{13} := 1$



obstacle surface probability distribution



grid occupancy probability distribution

IDEAL SENSOR

2) Gaussian distribution

The probability distribution for a given gaussian distribution can be computed using the cummulative normal distribution, **cnorm(x, av, disp)**:

$av := 2.5$ *average*

$disp := .15$ *dispersion*

$i := 0..50$ *quantization index*

$s := .1$ *increment step <m>*

$x_i := i \cdot s$ *discrete points*

$x_{i+1} := (i+1) \cdot s$

$$d1_i := cnorm\left(\frac{x_{i+1} - av}{disp}\right) - cnorm\left(\frac{x_i - av}{disp}\right)$$ *probability distribution for surface range*

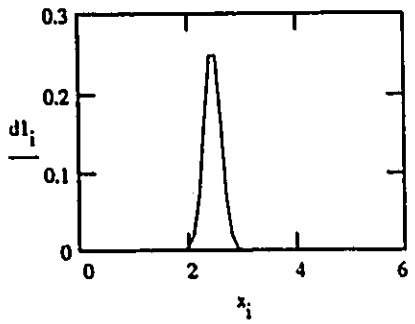
$$t1 := \sum_i d1_i \qquad t1 = 1$$ *converge to 1*

To compute the occupancy probability distribution, the proposed method would add for the cell i ' the sum of all previous cells probability multiplied with 0.5 (unknown state).

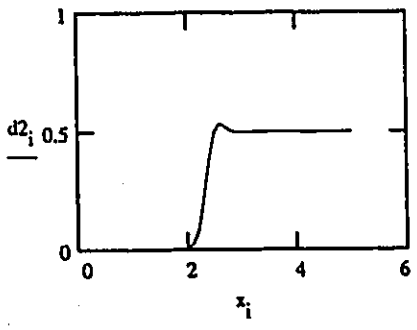
In other words, if a cell 'k' has an occupancy probability 'p', all the cells beyond it would have a probability of being occupied 0.5 times 'p'. (The ideal sensor is nothing but the case p=1, k='range'). Therefore:

$$d2_i := .5 \cdot d1_i + \sum_{k=0}^{i} .5 \cdot d1_k$$ *grid occupancy probability distribution*

$$t2 := \sum_i d2_i \qquad t2 = 13.75$$ *>1, has no meaning*



obstacle surface probability distribution



grid occupancy probability distribution

GAUSSIAN DISTRIBUTION

3) *Repeated gaussian distribution measurement converge to the ideal sensor behaviour*

$t := 0..4$ — *the number of times the measurement was repeated*

$p_{i,0} := .5$ — *the initial distribution (t=0) having all cells (i) 'unknown' (prob. 0.5)*

Bayes' formula decomposed in 3 steps:

$p_{i,t+1} := p_{i,t} \cdot dl_i$ — $p_{i,t}$ *- prior distribution*

$a_t := \sum_i p_{i,t} \cdot dl_i$ — $p_{i,t+1}$ *- posterior distribution*

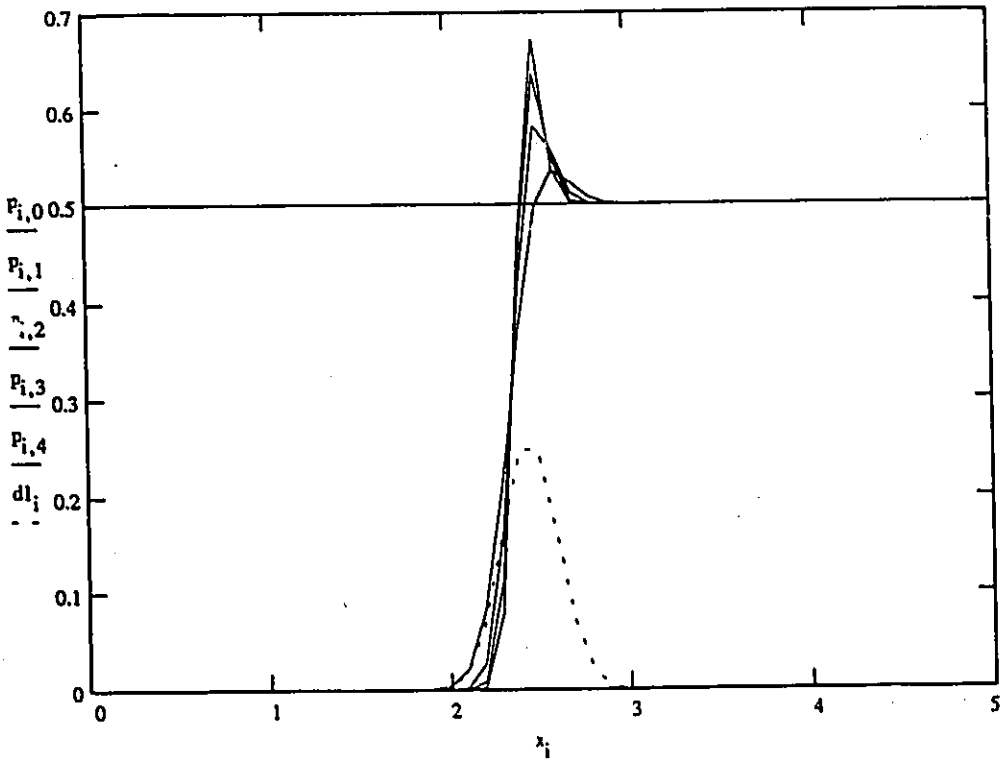
$$p_{i,t+1} := \frac{p_{i,t+1}}{a_t}$$

$c_{t+1} := \sum_i p_{i,t+1}$ *(only for verification in all cases the limit should be 1:* c_{t+1} *)*

c_{t+1}
1
1
1
1
1

The proposed method for occupancy grid computation :

$$r_{i,t+1} := \sum_{k=0}^{i} p_{k,t+1} \cdot .5 \qquad p_{i,t+1} := .5 \cdot p_{i,t+1} + r_{i,t+1}$$



Different graphics can be plotted for different dispersions or using another resolution (range of 'i' and 's', but such as i*s=5<m>, eg: 50*0.1).

APPENDIX B

```
/******************************** ********************************************/
                              LAB_PC.C
/****************************************************************************/
// contains the initialization routines for the National Instruments card

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include "nidaq.h"                                  /* function definitions */
#include "nidaqerr.h"                               /* error code definitions */

/********************** FUNCTIONS DEFINED IN THIS FILE ********************/
int find_board(void);
void set_board(int board);
/****************************************************************************/

/****************************************************************************/
                              int find_board(void)
/* retrieves the LAB-PC board ***********************************************/

{
int board, dum, brd_code;

for (board=1; board<9; board++)
        {
        Get_DA_Brds_Info(board, &brd_code, &dum, &dum, &dum, &dum, &dum, &dum, &dum);
        if (brd_code == 9) return board;
        }
printf("Sorry, cannot find the LAB-PC board instaled!\n");
printf("Press any key to exit the program:");
getch();
exit(1);
return -1;
}

/****************************************************************************/
                              void set_board(int board)
/* configures the input/outputs channels ************************************/

{
int i=0, errNum;

while(i<2)        /*initialization and config loop for both analog output channels*/
        {
        AO_VWrite(board,i,-4.99);
        errNum = AO_Config(board,i,1,0,0);
        if(errNum != 0)
                {
                printf("\nCannot set unipolar output mode for channel %d\n",i);
                printf("Press any key to exit the program:");
```

```
                getch();
                exit(2);
                }
        i++;
        }

errNum = DIG_Prt_Config(board,0,0,1); /* config DIG PORT to output */
if(errNum != 0)
        {
        printf("\nCannot set digital port to output mode\n",i);
        printf("Press any key to exit the program:");
        getch();
        exit(3);
        }

errNum = AI_Configure(board,-1,1,0,1,0);
if(errNum != 0)
        {
        printf("\nCannot set unipolar analog inputs\n",i);
        printf("Press any key to exit the program:");
        getch();
        exit(4);
        }
printf("Succesful initialization!");
}
```

```
/*************************************************** ******************************/
                                DRIVING.C
/*************************************************** ******************************/
// module that contains the software interface with the mobile robot for driving purpose

#include <stdio.h>
#include <nidaq.h>
#include <conio.h>
#include <graphics.h>
#include <stdlib.h>
#include <math.h>

#define STEAR  3.0              /* steering speed <V>, max. 5V */
#define DELTA  0.6              /* speed increment <V> (acceleration) */

/**************** FUNCTIONS DEFINED IN THIS FILE ************************/

void init_steering(void);
void speed_incr(void);
void speed_decr(void);
void steer_left(void);
void steer_right(void);
void init_all(void);
void reset_all(void);

/*****************************************************************************/

extern int board;

int Position = 320;                     /* global var. for speed drawing */
float   Max = 5.0,                      /* limit positions <V> for steering */
        Min = 3.1;
float  Angle = 90;                      /* global var. for steering position */
double  Speed = 0,                      /* global variable for speed */
        Voltage;                        /* global var. for value read by LAB-PC
                                           from steering feedback interface */

/*****************************************************************************/
                                        void init_steering(void)
/* read & display the initial position for the steering wheel ****************/

{
AI_VRead(board,0,1,&Voltage);
Angle = (Voltage - Min) * 60 /(Max - Min) + 55;
setcolor(EGA_RED/*BLACK*/);
setlinestyle(SOLID_LINE,0,THICK_WIDTH);
arc(320,340,Angle,Angle+10,50);
}
```

```
/*************************************************************************/
                                void speed_incr(void)
/* increments speed with value of DELTA and displays the new Position ***************/

{
double volt;

        /*      gotoxy(1,1);
                printf("SPEED =    "); only for testing */
setcolor(EGA_LIGHTGREEN);
setlinestyle(SOLID_LINE,1,THICK_WIDTH);
line(Position,340,Position,349);

if(Speed >= 0)
        {
        Speed = max(2.5, Speed + DELTA);
        if(Speed >= 4.7) Speed = 4.7;
        Position = 320 + (Speed - 2.3) * 60;
        DIG_Out_Port(board,0,2);
        }
        else
                {
                if(Speed >= -2.5)
                        {
                        Speed = 0;
                        Position = 320;
                        }
                        else
                                {
                                Speed = Speed + DELTA;
                                Position = 320 + (Speed + 2.3) * 60;
                                }
                }
 volt = fabs(Speed);
 AO_VWrite(board,0,volt);
        /*      gotoxy(1,1);
                printf("SPEED = %2.1f",Speed); */
 setcolor(EGA_RED);
 line(Position,340,Position,349);
 }

 /*************************************************************************/
                                void speed_decr(void)
 /* decrement Speed with value of DELTA and displays the new Position ***************/

 {
 double volt;

                /*      gotoxy(1,1);
                        printf("SPEED =    ");    */
 setcolor(EGA_LIGHTGREEN);
 setlinestyle(SOLID_LINE,1,THICK_WIDTH);

 line(Position,340,Position,349);
```

```
if(Speed <= 0)
        {
        Speed = min(-2.5, Speed - DELTA);
        if(Speed <= -4.7) Speed = -4.7;
        Position = 320 + (Speed + 2.3) * 60;
        DIG_Out_Port(board,0,0);
        }
        else
                {
                if(Speed <= 2.5)
                        {
                        Speed = 0;
                        Position = 320;
                        }
                        else
                                {
                                Speed = Speed - DELTA;
                                Position = 320 + (Speed - 2.3) * 60;
                                }
                }
volt = fabs(Speed);
AO_VWrite(board,0,volt);

/*      gotoxy(1,1);
        printf("SPEED = %2.1f",Speed);  only for testing  */
setcolor(EGA_RED);
line(Position,340,Position,349);
}

/*************************************************************************/
                                void stear_left(void)
/* start/stop stearing movement to the left ***********************************/

{
int byte;
char check;

setlinestyle(SOLID_LINE, 1, THICK_WIDTH);
if(Speed < 0) byte = 1;
        else
                byte = 3;

DIG_Out_Port(board,0,byte);
AO_VWrite(board,1,STEAR);
do
        {
        AI_VRead(board,0,1,&Voltage);
        setcolor(EGA_LIGHTGREEN);
        arc(320,340,Angle,Angle+10,50);
        Angle = (Voltage - Min) * 60 / (Max - Min) + 55;
        setcolor(EGA_RED);
        arc(320,340,Angle,Angle+10,50);
        if(kbhit())
                {
```

```
                check = getch();
                if(check != '\x0')
                        {
                        AO_VWrite(board,1,0);
                        return;
                        }
                }
        }
while(Voltage <= Max);
AO_VWrite(board,1,0);
return;
}

/*************************************************************************/
                                void stear_right(void)
/* start/stop stearing movement to the right ***************************************/

{
int byte;
char check;

setlinestyle(SOLID_LINE, 1, THICK_WIDTH);
if(Speed < 0) byte = 0;
        else
                byte = 2;

DIG_Out_Port(board,0,byte);
AO_VWrite(board,1,STEAR);
do
        {
        AI_VRead(board,0,1,&Voltage);
        setcolor(EGA_LIGHTGREEN);
        arc(320,340,Angle,Angle+10,50);
        Angle = (Voltage - Min) * 60 / (Max - Min) + 55;
        setcolor(EGA_RED);
        arc(320,340,Angle,Angle+10,50);
        if(kbhit())
                {
                check = getch();
                if(check != '\x0')
                        {
                        AO_VWrite(board,1,0);
                        return;
                        }
                }
        }
while(Voltage >= Min);
AO_VWrite(board,1,0);
return;
}
```

```
/*****************************************************************************/
                                    void reset_all(void)
/* reset Speed and exit program **********************************************/

{
AO_VWrite(board,0,0);
AO_VWrite(board,1,0);
exit(0);
}

/*****************************************************************************/
                                    void init_all(void)
/* reset/display Speed *******************************************************/

{
AO_VWrite(board,0,0);
setlinestyle(SOLID_LINE,1,THICK_WIDTH);
setcolor(EGA_LIGHTGREEN);
line(Position, 340, Position, 349);
Speed = 0;
Position = 320;
setcolor(EGA_RED);
line(Position, 340, Position, 349);
}
```

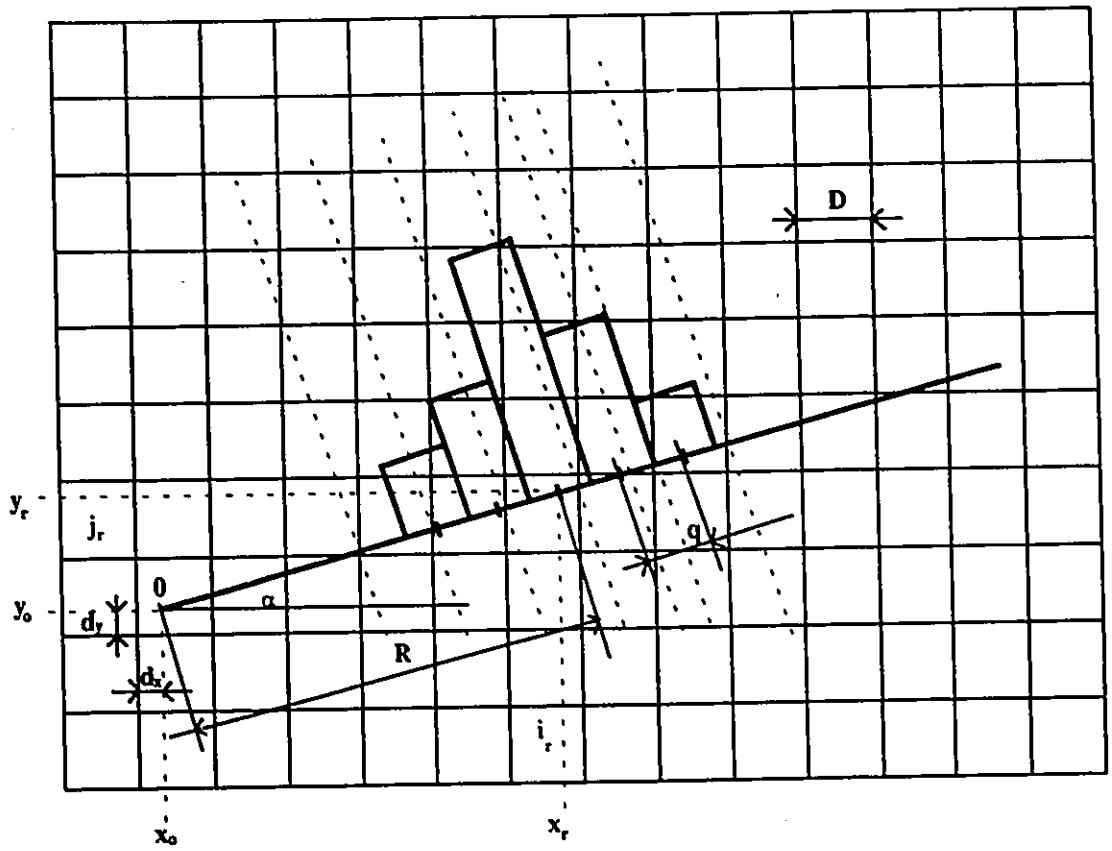
```
/*****************************************************************************/
                                SCREEN.C
/*****************************************************************************/
// module that presents a GUI for the local views

#include <stdio.h>
#include <graphics.h>
#include <conio.h>

#define VEH_LENGHT_MM  510
#define VEH_WIDTH_MM   720

/******************** FUNCTIONS DEFINED IN THIS FILE ************************/

int convert(int maxRangeM);
void sensor_position(int mmPerPixel);
void put_sensor(void);
int message(void);
void del(int num);
void display(int num);
void fillgrid(int x, int y);

/*****************************************************************************/

/* GLOBAL variable holding the pixel positions of the 8 sensors */
extern int sensorPixel[16];

/* GLOBAL variables holding the directions of the 8 sensors */
extern float direction[8] = {55, 75, 110, 60, 120, 70, 105, 125};

/* GLOBAL variables holding the cos & sin directions for the 8 sensors */
extern float cosAngle[8] = { 0.61566, -0.1908, -0.45399, 0.5299,
                             -0.5446, 0.43837, 0.17364, -0.6427 };
extern float sinAngle[8] = { 0.7880, 0.9816, 0.8910, 0.8480,
                              0.83867, 0.89879, 0.9848, 0.766 };

int mmPerPixel;          /* conversion factor */
extern Average[8];       /* the average range value for 8 readings */
int Grid;
int pixRange;

/*****************************************************************************/
                          int convert(int maxRangeM)
/* returns the conversion factor for the given range *************************/

{
int vehLenght,
        vehLeft,
        vehRight,
        gdriver = EGA,
        gmode = EGAHI,
        i;
int poly[8] = {0,0,0,300,639,300,639,0};
```

```
mmPerPixel = (1000 * maxRangeM + VEH_LENGHT_MM) / 300;
vehLenght = 300 - (VEH_LENGHT_MM / mmPerPixel);
vehLeft = 320 - (( VEH_WIDTH_MM / 2 ) / mmPerPixel );
vehRight = 320 + (( VEH_WIDTH_MM / 2 ) / mmPerPixel );
Grid = 100 / mmPerPixel - 1;

setlinestyle(SOLID_LINE, 1, NORM_WIDTH);
setcolor(EGA_LIGHTCYAN);
setfillstyle(SOLID_FILL, LIGHTGRAY);
fillpoly(4,poly);

/* added portion for the grids */
/*
setcolor(CYAN);
for(i=0; i<300; i++)
        {
        if(i<vehLenght) line(0,i,639,i);
        else
                {
                line(0,i,vehLeft,i);
                line(vehRight,i,639,i);
                }
        i = i + Grid;
        }
for(i = 0; i<639; i++)
        {
        if(i<vehLeft) line(i,0,i,300);
        else
                if(i>= vehLeft && i < vehRight) iine(i,0,i,vehLenght);
                else line(i,0,i,300);
        i = i+Grid;
        }
*/
line(vehLeft,vehLenght,vehLeft,300);            /* draw the vehicle */
line(vehRight,vehLenght,vehRight,300);
line(vehLeft,vehLenght,vehRight,vehLenght);

window(1,22,80,23);                     /* set text window to display the maxRangeM */
gotoxy(52,1);
printf(" MAX. RANGE IN FRONT: %i m ",maxRangeM);
setlinestyle(SOLID_LINE, 1, THICK_WIDTH);
return mmPerPixel;
}


/*************************************************************************/
                        void sensor_position(int mmPerPixel)
/* computes the positions of the 8 sensors ***************************************/

{
int vehLeft, i;
int sensorMM[16] =      {       700,    40,             /* sensor #     0 */
                                700,    490,            /*              1 */
```

```
                    650,    490,
                    600,    490,
                    130,    490,
                    80,     490,
                    30,     490,
                    30,     40              /*          7  */
                } ;

vehLeft = 320 - (( VEH_WIDTH_MM / 2 ) / mmPerPixel);

for(i=0; i<=14; ++i)
        {
        sensorPixel[i] = vehLeft + (sensorMM[i] / mmPerPixel);
        ++i;
        }
for(i=1; i<=15; ++i)
        {
        sensorPixel[i] = 300 - (sensorMM[i] / mmPerPixel);
        ++i;
        }
}

/*************************************************************************/
                        void put_sensor(void)
/* draws the positions for the 8 sensors *********************************/

{
int i;

for(i=0; i<=14; ++i)
        {
        putpixel( sensorPixel[i], sensorPixel[i+1], EGA_WHITE );
        ++i;
        }
}

/*************************************************************************/
                        int message(void)
/* outputs text on the screen ********************************************/

{
int maxRangeM;
gotoxy(52,1);
printf("                ");
gotoxy(52,1);
printf("INPUT NEW MAX. RANGE <m>:");
scanf("%i",&maxRangeM);
return maxRangeM;
}
```

```
/*************************************************************************/
                                    void del(int num)
/* delete previous range display for sensor num ***************************************/

{
int xmax, ymax, pix_range;

setcolor(EGA_LIGHTGRAY);
setlinestyle(SOLID_LINE,1,NORM_WIDTH);

pix_range = Average[num-1]/mmPerPixel;
xmax = sensorPixel[2*num-2] + (int)(pix_range * cosAngle[num-1]);
ymax = sensorPixel[2*num-1] - (int)(pix_range * sinAngle[num-1]);
line( sensorPixel[2*num-2], sensorPixel[2*num-1], xmax, ymax);

/*
setlinestyle(SOLID_LINE,1,NORM_WIDTH);
setfillstyle(SOLID_FILL,LIGHTGRAY);
setcolor(CYAN);
fillgrid(xmax, ymax);
*/
}

/*************************************************************************/
                                    void display(int num)
/* display new range for sensor num ***************************************/

{
int x, y, xmax, ymax, i, pix_range;

setcolor(EGA_RED);
setlinestyle(SOLID_LINE,1,NORM_WIDTH);
pix_range = Average[num-1]/mmPerPixel;
xmax = sensorPixel[2*num-2] + (int)(pix_range * cosAngle[num-1]);
ymax = sensorPixel[2*num-1] - (int)(pix_range * sinAngle[num-1]);
line( sensorPixel[2*num-2], sensorPixel[2*num-1], xmax, ymax);

/*
setfillstyle( SOLID_FILL, WHITE);
setcolor(CYAN);
for( i=Grid; i<pix_range; i++)
        {
        x = sensorPixel[2*num-2] + (int)(i * cosAngle[num-1]);
        y = sensorPixel[2*num-1] + (int)(i * sinAngle[num-1]);
        fillgrid(x,y);
        i = i+Grid/4;
*/
}
```

APPENDIX C



- Sensor location : (x_o, y_o), located inside its cell (i_o, j_0) with the offset (d_x, d_y);
- Range measurement: R;
- Sensor angle: α;
- Obstacle location: (x_r, y_r), located in the cell (i_r, j_r);
- Step increments: q;
- Grid size: D;

The following relations apply:

$$\begin{cases} x_r = x_0 + R\cos\alpha \\ y_r = y_0 + R\sin\alpha \end{cases}$$

$$\begin{cases} i_r = \text{int}(\dfrac{d_x + R\cos\alpha}{D}) \\ j_r = \text{int}(\dfrac{d_x + R\cos\alpha}{D}) \end{cases} \quad (x_r, y_r) \in Cell(i_r, i_r)$$

The iterational computation is as following :

$$\begin{cases} i_{r-q} = \text{int}(\dfrac{d_x + R\cos\alpha - q\cos\alpha}{D}) \\ j_{r-q} = \text{int}(\dfrac{d_x + R\cos\alpha - q\sin\alpha}{D}) \end{cases} \quad (x_{r-q}, y_{r-q}) \in Cell(i_{r-q}, i_{r-q})$$

.....

$$\begin{cases} i_{r\pm kq} = \text{int}(\dfrac{d_x + R\cos\alpha \pm kq\cos\alpha}{D}) \\ j_{r\pm kq} = \text{int}(\dfrac{d_x + R\sin\alpha \pm kq\sin\alpha}{D}) \end{cases} \quad (x_{r\pm kq}, y_{r\pm kq}) \in Cell(i_{r\pm kq}, i_{r\pm kq})$$

k, half the number of cells on which the p .d.f. is expressed (int(5/2) = 2, in this ex.)

Rewriting the precedent equation:

$$\begin{cases} i_m = \text{int}(\dfrac{d_x + m\cos\alpha}{D}) \\ j_m = \text{int}(\dfrac{d_x + m\sin\alpha}{D}) \end{cases}$$

$m = \overline{q, r+kq}$, i_m, j_m are the cell index relative to the sensor origin

The absolute index for the involved cells is given by :

$$\begin{cases} i_M = i_m + i_0 \\ j_M = j_m + j_M \end{cases}$$

With this we obtain a collection of points having an estimated probability P_{xy} :

$M(X_{max}, Y_{max}, P_{xy})$

and a corresponding collection of cells with the prior probability taken from the global map , P_{ij}:

$C(I, J, P_{ij})$

```
/*****************************************************************/

                        sensor.cxx

/*****************************************************************/

// this module computes the sensor location data and the occupancy probability grid
#include <iostream.h>
#include <stdio.h>
#include <math.h>
#include <graphics.h>
#include "sensor3.h"
#include <conio.h>

extern CalibSet CS;
extern float D1[11], D2[7], D3[5], D4[3];        //arays holding the sensor model
extern float huge Grid[250][200];                // global map size
extern int huge Obstacle[250][200];              //obstacle map
extern float GridSize;
extern void filltrace(int, int);                 //fills in the cell(int,int)

//------------------------------ func defined in CalibSet class: ------------------

CalibSet::CalibSet(void) {
        distStep = .1;          // value of increment step (q)
        ptoc[0] = &D1[0];       // PointersToCalib set
        ptoc[1] = &D2[0];
        ptoc[2] = &D3[0];
        ptoc[3] = &D4[0];
        }

CalibSet::~CalibSet(void) {
        }

//------------------------------ func defined in Sensor class: --------------------

Sensor::Sensor() {
        }

Sensor::~Sensor() {
        }

int Sensor::setValues(float angle, float posX, float posY, float range) {
// sets the sensor absolute position, computes all necessary data
        int i, x, y;

        stepX = CS.distStep/GridSize*cos(angle);
        stepY = CS.distStep/GridSize*sin(angle);

        absIndexX = (int)(posX/GridSize);
        absIndexY = (int)(posY/GridSize);

        offsetX = posX - absIndexX*GridSize;
```

```
        offsetY = posY - absIndexY*GridSize;

        for(i=0; i<50; i++) {
                x = (int)(offsetX + i*stepX) + absIndexX;
                y = (int)(offsetY - i*stepY) + absIndexY;
                if(x < 0 || y < 0 || x > 250 || y > 200) {          // preventive only
                        range = (i-1)*0.1;
                        break;
                        }
                }
// depinding on range, we pick up the respective sensor model:
        if(range > 4) {
                maxGrids = 11;
                ptoCS = CS.ptoc[0];
                }

        else if(range > 3) {
                maxGrids = 7;
                ptoCS = CS.ptoc[1];
                }

        else if(range > 1.5) {
                maxGrids = 5;
                ptoCS = CS.ptoc[2];
                }

        else if(range > .1) {
                maxGrids = 3;
                ptoCS = CS.ptoc[3];
                }

        maxSteps = (int)(range/CS.distStep) + (maxGrids-1)/2;
        freeCells = maxSteps - maxGrids;
// debug only:
//      cout << endl << "maxSteps: " << maxSteps;
//      cout << "\tabsX: " << absIndexX << "\tabsY: " << absIndexY;
//      cout << "\toffX: " << offsetX << "\toffY: " << offsetY << endl;
        return 0;
        }

void Sensor::assignFreeCells(void) {
        int i, x, y;

        for(i=1; i<freeCells; i++) {
                x = (int)(offsetX + i*stepX) + absIndexX;
                y = (int)(offsetY - i*stepY) + absIndexY;
                Grid[x][y] = 0;
                if( x>250 || y>200)
                        cout << x << " " << y << "; ";
                setfillstyle(SOLID_FILL, WHITE);
                filltrace(x, y);
                }
        }
```

```
void Sensor::setPriorDistr(void) {
// picks-up from the global map the priors for the involvrd cells
        int i, nrBayesGrids;

        for(i=1; i<maxGrids; i++) {
                IndexX[i] = (int)(offsetX + (i+freeCells)*stepX) + absIndexX;
                IndexY[i] = (int)(offsetY - (i+freeCells)*stepY) + absIndexY;

                if(IndexX[i] < 250 && IndexY[i] <200) {
                        Prob[i] = Grid[IndexX[i]][IndexY[i]];
                        }
                else {
//                      cout << IndexX[i] << "," << IndexY[i] << ";";
                        Prob[i] = .5;
                        }
                }
        }

 void Sensor::computePostDistr(void) {
        int i;
        float sum = 0.0, tot = 0.0;

        for(i=1; i<maxGrids; i++) {     // compute the numerator of Bayes' formula
                Prob[i] = Prob[i] * (*(ptoCS+i));
                sum += Prob[i];
                }
                if(sum < 10e-5) {       // avoid error for division by zero
                        sum = 10e-5;
                        }
        for(i=1; i<maxGrids; i++) {    // denominator of Bayes formula
                Prob[i] = Prob[i]/sum;      // normalize
                tot += Prob[i];
                if(i>0)
                        Prob[i] = .5*Prob[i] + .5*tot;  // occ. grid transformation
                #ifndef GRAPH
                fprintf(out,"\nProb[%d] = %5.4f",i,Prob[i]);
                #endif
                }

         Grid[IndexX[0]][IndexY[0]] = Prob[0];
                #ifndef GRAPH
                printf("%5.4f\t",Grid[IndexX[0]][IndexY[0]]);
                #endif
                if(Grid[IndexX[0]][IndexY[0]] >= .99)
                        setfillstyle(SOLID_FILL, BLACK);
                else if(Grid[IndexX[0]][IndexY[0]] >= .95)
                        setfillstyle(SOLID_FILL, BLUE);
                else if(Grid[IndexX[0]][IndexY[0]] >= .8)
                        setfillstyle(SOLID_FILL, GREEN);
                else if(Grid[IndexX[0]][IndexY[0]] >= .65)
                        setfillstyle(SOLID_FILL, RED);
                else if(Grid[IndexX[0]][IndexY[0]] >= .55)
                        setfillstyle(SOLID_FILL, LIGHTMAGENTA);
```

```
            else if(Grid[IndexX[0]][IndexY[0]] >= .45)
                    setfillstyle(SOLID_FILL, YELLOW);
            else if(Grid[IndexX[0]][IndexY[0]] >= 0)
                    setfillstyle(SOLID_FILL, WHITE);

            filltrace(IndexX[0],IndexY[0]);

    for(i=1; i<maxGrids; i++) {
            if(IndexX[i] == IndexX[i-1] && IndexY[i] == IndexY[i-1]) {
                    Grid[IndexX[i]][IndexY[i]] += Prob[i];
                    }
            else
                    Grid[IndexX[i]][IndexY[i]] = Prob[i];

            if(Grid[IndexX[i]][IndexY[i]] >= .95)
                    setfillstyle(SOLID_FILL, BLACK);
            else if(Grid[IndexX[i]][IndexY[i]] >= .8)
                    setfillstyle(SOLID_FILL, BLUE);
            else if(Grid[IndexX[i]][IndexY[i]] >= .7)
                    setfillstyle(SOLID_FILL, RED/*GREEN*/);
            else if(Grid[IndexX[i]][IndexY[i]] >= .6)
                    setfillstyle(SOLID_FILL, MAGENTA/*RED*/);
            else if(Grid[IndexX[i]][IndexY[i]] >= .55)
                    setfillstyle(SOLID_FILL, LIGHTGRAY);
            else if(Grid[IndexX[i]][IndexY[i]] >= .45)
                    setfillstyle(SOLID_FILL, YELLOW);
            else if(Grid[IndexX[i]][IndexY[i]] >= 0)
                    setfillstyle(SOLID_FILL, WHITE);

            filltrace(IndexX[i],IndexY[i]);

            #ifndef GRAPH
            printf("%5.4f\t",Grid[IndexX[i]][IndexY[i]]);
            #endif
            }
    }
/*************************************************************************/
```

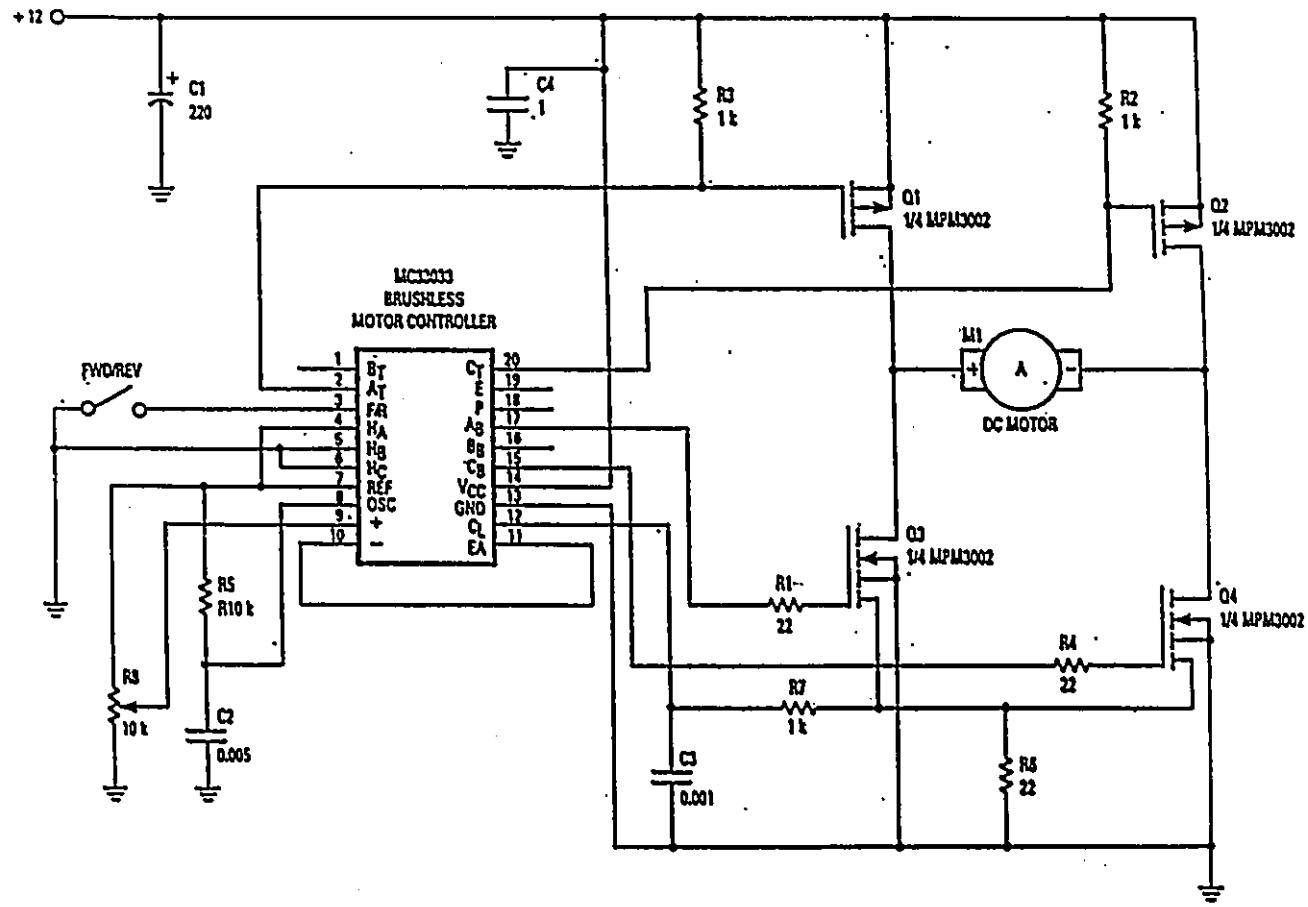
sensor.h

```
/*************************************************************/
// header file containing the declaration for CalibSet and Sensor class (objects CS. and S. in main)

#include <iostream.h>
#include <stdio.h>

#define MAX_SET 4
#define GRAPH           // for graphic display, not debug

class CalibSet {
    public:
    float *ptoc[MAX_SET];
```

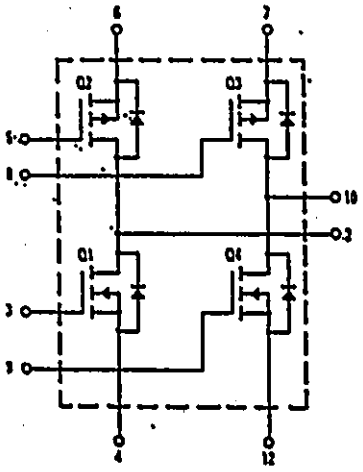
```
        int nrGrids[MAX_SET];
        float distStep;          // distribution incr. step
        CalibSet(void);
        ~CalibSet(void);
        int setCalibIndex( float range);
        };

class Sensor {
        protected:
        float stepX, stepY;
        float offsetX, offsetY;          // sensor's relative position to the grid
        float *ptoCS;
        int maxGrids, calibIndex, freeCells;
        int maxSteps, absIndexX, absIndexY;
        int IndexX[25], IndexY[25];      //keeps track of Grids involved in Bayesian computation
        float Prob[25];                  // for Bayes computation
        public:
        Sensor(void);
        ~Sensor(void);
        int setValues(float angle, float posX, float posY, range);
        void assignFreeCells(void);      // assign the free cells to the obstacle distribution
        void setPriorDistr(void);
        void computePostDistr(void);
        };
```

APPENDIX D

Motorola Circuits for Brushless DC Motors Drive (ref. from [17])



MPM3004 — Standard H-Bridge



60 Volts and 10 Amperes
P-Channel MOSFETs — $r_{ds(on)}$ = 0.28 Ohm maximum
N-Channel MOSFETs — $r_{ds(on)}$ = 0.15 Ohm maximum