A Fault-Tolerant Architecture for an Automatic Vision-Guided Vehicle

MANSUR R. KABUKA, MEMBER, IEEE, SURJADI HARJADI STUDENT MEMBER, IEEE, AND AKMAL YOUNIS

Abstract—**A fault-tolerant architecture for an automatic navigation system is presented. The system employs a mixed type of architecture in which the speed advantages of both pipelined and parallel architectures are exploited to achieve real-time navigation. The fault-tolerant architecture is presented using two reconfiguration strategies. To evaluate the proposed architecture, its reliability, availability, and safety are investigated using Markov models. In addition, the feasibility of implementing the proposed architecture is studied.**

I. Introduction

THE study of automatic guided vehicles (AGV) has received a great deal of attention in the past decade due in part to the increasingly complex modern transportation problem and to the vast area of related applications that involve monotonous and tedious tasks or hazardous environments.

In fact, AGV's can be considered one of the key factors in flexible manufacturing systems. These systems are used to maximize the throughput by attempting to equalize the workload among their various components. This can be achieved by premeditated task planning and devising of efficient navigation methods for the autonomous transporters. These methods should be flexible, inexpensive, and easily modifiable. Initial efforts in obtaining these methods for robot navigation included the use of buried wires [1] and painted lines [2], [3]. The buried-wire method is still the most popular among the Japanese manufacturers of mobile robots [4].

Research has concentrated lately on navigation with little or no *a priori* knowledge of the surrounding environment. Since the knowledge about the environment is minimal, the system must depend on its sensing mechanisms for navigation. Most of the current research has relied upon visual sensors [5]–[13], ultrasonic rangefinders [14], tactile sensors [15]–[18], and laser rangefinders [19]–[21]. These systems are usually designed with the aim of being completely autonomous. However, most mobile robots are still inept in this regard, the limitations defined finally by their sensing capabilities. Realistic implementation of completely autonomous robots capable of navigating any unknown environment has yet to be achieved, although great advancements have been made. On the contrary, if the environment is controllable—a likely situation in indoor industrial settings—the situation is simplified and practical solutions could be implemented.

In a controlled environment, the AGV usually navigates by correlating the information previously stored about the environment with the information it gathers along its way. One way of providing this information is in the use of marks and patterns that can be discretely placed within the environment. The AGV determines its position relative to these marks and subsequently locates itself in the environment. Some of the marks already developed include laser-detected corner cubes [19] and retroreflective "spot marks" [22].

Artificial intelligence also has been used for purposes of path planning and navigation [23], [24]. If the environment is assumed to be known completely, algorithmic methods can be employed to plan the path as a one-time off-line operation [25]–[27]. Although this can prove to be fast for path-planning execution, it does not account for the possibility of the path being blocked due to temporary reasons. In such a situation, the AGV's should have the ability to revise, during navigation, their previously planned paths so as to obtain optimal or suboptimal solutions [28]–[30].

To increase the efficiency and minimize the chance of accidents, an AGV system must be designed with a maximum regard towards reliability, availability, and safety. In this paper, a fault-tolerant architecture is proposed for the implementation of the AGV presented in [29] and [30] to attain these properties. No matter how well designed the system is, its circuit components may fail at a crucial moment. This can prove to be hazardous not only to the AGV but also to its environment. In a less serious condition, the AGV can lose its way and wander until rescued, with the consequence of wasting both time and efficiency. Moreover, the interruption of material flow caused by malfunction can seriously disrupt the whole system's operation. Under more serious circumstances, the AGV can collide with another AGV, causing massive destruction that could have been avoided with the use of a fault-tolerant AGV. On the other hand, a fault-tolerant AGV will continue its work uninterrupted and thus inflict a positive influence on overall productivity.

Manuscript received February 24, 1988; revised August 8, 1989.

The authors are with the Department of Electrical and Computer Engineering, University of Miami, P. O. Box 248294, Coral Gables, FL 33124.

IEEE Log Number 8932019.

0018-9472/90/0300-0380$01.00

The navigation system, presented in [29] and [30], is briefly described in Section II. The fault-tolerant architecture of the system is presented in Section III. An evaluation of the proposed architecture is investigated in Section IV, in which reliability, availability, and safety are discussed using Markov models. In Section V, a feasibility study of the proposed architecture is conducted to illustrate that it could be implemented to perform the required real-time navigation of the AGV. Also, this feasibility study could be considered, in a wider context, useful for implementing any application in which the transformation of a path network map into an interconnected graph is needed.

II. AGV System Description

The AGV goes through three different phases to achieve real-time navigation.

a) *Path network learning:* During this phase a scaled map of the path network is presented to the system via a camera. The system automatically extracts information, such as location and number of incident edges of a node, node adjacency, edge path direction trace, and edge length, using image processing techniques. Information, such as width and height of an edge, whether an edge is directed or not, and the maximum load an edge can support, is input to the system interactively and stored in the path network database. The learning process is not entirely carried out in an interactive mode due to the fact that it would have been time consuming and prone to human errors.

b) *Automatic path scheduling:* In this phase, the information in the path network database is used to generate a path that could be traced by the AGV to navigate from the source node to the destination note. The requested navigation could be specified to the system in one of three possible alternatives:

   1) initial, destination nodes;
   2) initial, destination, and part of the path crossing nodes;
   3) initial, destination, and all the path crossing nodes.

c) *Real-time navigation:* During real-time navigation, the AGV extracts information about the environment and confirms it with the previously planned path. For the purpose of guidance, a finite-state machine is designed to control the navigation of the AGV. The AGV identifies nodes by decoding a bar code attached to each node and detects its current position by monitoring a painted line on the floor. If an unexpected node is reached, or a path is found blocked by an obstacle, a double heuristic search technique is used to generate a new path, to guide the system back to the required destination.

For the AGV to perform the previously mentioned tasks, it utilizes a multiple instruction multiple data stream (MIMD) architecture in which three parallel paths are identified.

1) *Node / edge pipelined unit (NPU):* This path consists of four image-processing pipelined modules used for segmentation, smoothing, thinning, and node/edge extraction. The NPU first transforms the input image of the path network into a graph of interconnected segments whose widths are one pixel in each direction. This graph is processed to extract information about network nodes and their interconnecting edges. The extracted information is stored in order to be used for automatic path planning.
2) *Bar-code pipelined unit (BPU):* When the AGV navigates along a path, it decodes the bar codes of the nodes it encounters using the BPU. The decoded information is compared with the stored information about the path in order for the AGV to determine if it is on the right path.
3) *Obstacle avoidance unit (OAU):* This unit is used for detecting unexpected obstacles using an ultrasonic rangefinder. For the occasion that an obstacle is detected, by observing both a discontinuity in the painted line and a shorter range than the expected distance to the next node, the authors developed an algorithm in which the AGV attempts to maneuver around it, if possible, or else backtracks to the last reached node and a new path is replanned. Ultrasonic obstacle avoidance has been chosen because it is inexpensive, easy to implement, and gives the required range information. The range data obtained is considered of acceptable resolution because it is used for detection, not for recognition.

III. AGV Fault-Tolerant Architecture

Since the AGV consists of three parallel paths, each of which uses a pipelined architecture as shown in Fig. 1, it is difficult to apply fault-tolerance design techniques to the system as a whole. Hence a modular approach is used in which each of the three parallel paths is designed independently.

In this paper, the issue of fault tolerance is addressed for the NPU, due to its inherent complexity as compared to the BPU and OAU, but the concept could similarly be applied to these units. Two architectures are presented for achieving fault tolerance in the NPU. Although both architectures are able to recover from two faults only, their concept of operation could easily be extended to include any number of faults m.

In the first architecture, shown in Fig. 2, a direct replacement strategy is employed to reconfigure the system after fault detection and location. In this strategy, one of the spares is downloaded with the state and program of the faulty processor in order to replace it when normal operations are resumed after recovery. While

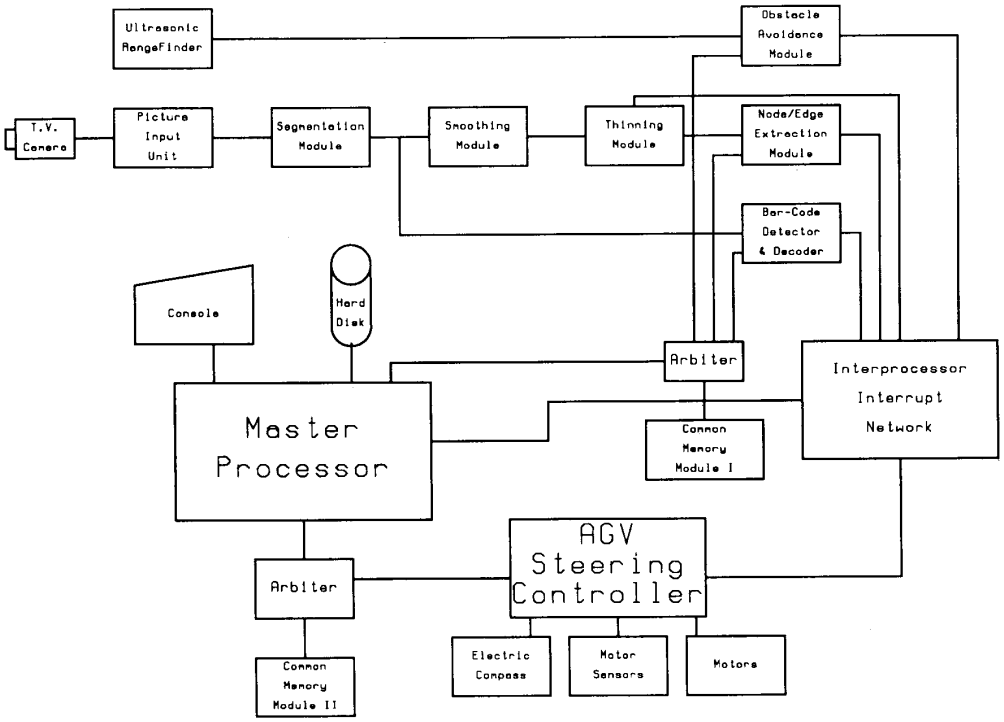

Fig. 1. AGV system architecture.

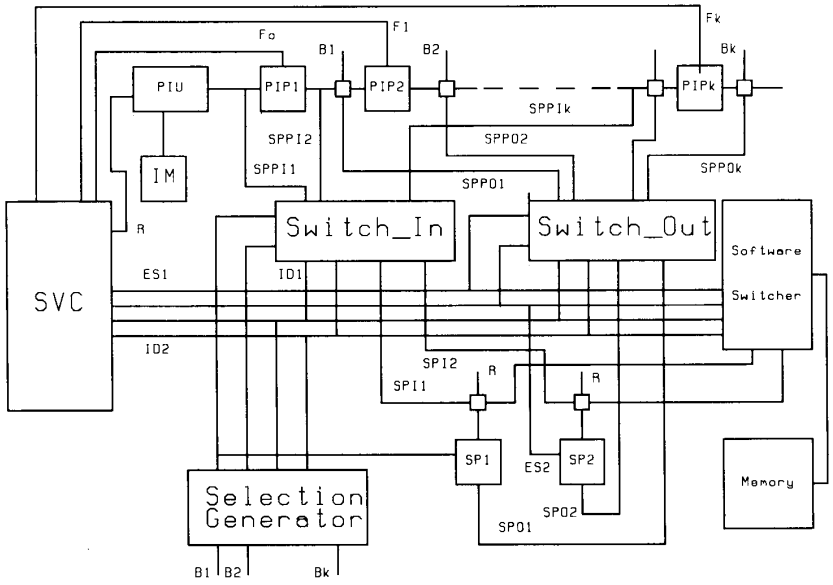

Fig. 2. Direct replacement architecture.

in the second architecture, shown in Fig. 3, a rippling replacement strategy is employed in which a faulty processor is replaced by its successor in the pipeline, and that successor is in turn replaced by its successor, and so on, until the last processor in the pipeline is replaced by one of the spare processors. Hence the reconfiguration process results in a shift-of-functions operation that starts at the faulty processor and ends up at one of the spare processors.

In both architectures, each module of the NPU is assumed to consist of a number of similar processors connected in a pipeline, with the total number of processors in all four modules equal to k. Both architectures employ the concept of standby sparing by having two

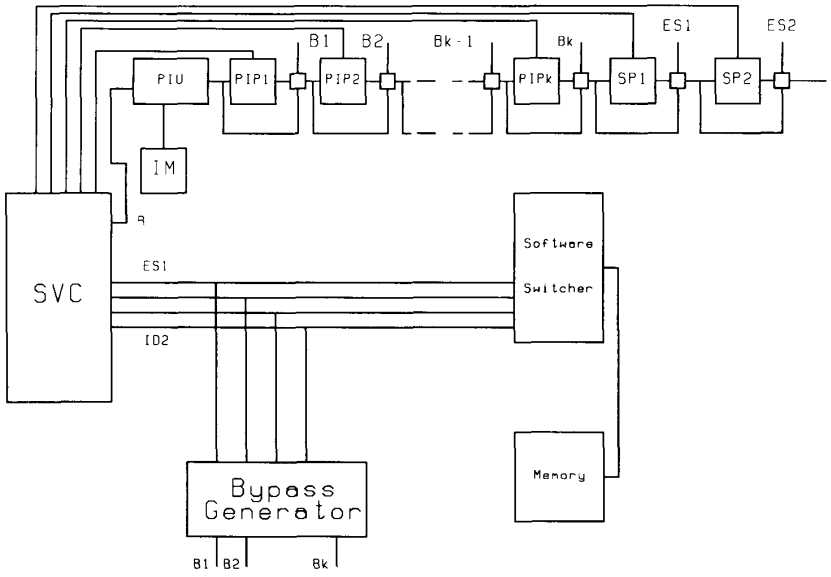

Fig. 3. Rippling replacement architecture.

spare processors ready to be substituted for faulty processors, once one or two faults have been detected and located. They differ mainly in the reconfiguration strategy, by which a faulty processor is replaced by one of the spare processors.

Moreover, to achieve their respective strategies, they use a combined hardware/software switching mechanism, which is initiated when a fault is detected. The hardware portion is responsible for establishing proper physical links between the spare processors and the nonfaulty processors, while the software portion is responsible for downloading the states and programs needed for each architecture so as to get the spare processor functioning in the pipeline.

A. Direct Replacement Architecture

The direct replacement architecture proposed for the NPU is shown in Fig. 2, where the following basic units are identified.

a) *State verification controller (SVC):* This unit is responsible for scanning and testing the pipelined processors to globally detect any faults. On the other hand, faults are detected locally within each processor running its own self-diagnostics; when a fault is detected, it alters an internal register containing its order (state) in the pipeline. When one or two faults are detected by the SVC, it generates the codes of the faulty processors on its output lines (ID_1, ID_2) and activates the corresponding enable lines (ES_1, ES_2). These signals are used in turn by the other units in the system to initiate the switching mechanism. Each of the *ID* codes has s bits, where s is given by $\lceil \log_2 k \rceil$, so that each pipelined processor is given a unique identifying code.

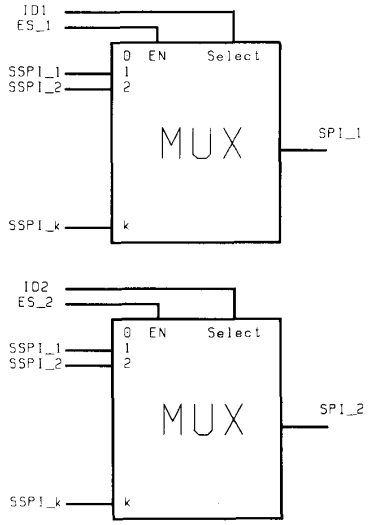

Fig. 4. Switch_{in} basic design.

b) *$Switch_{in}$:* This logic control unit establishes proper links from the outputs of the pipelined nonfaulty processors preceding the faulty ones, to the inputs of the spare processors (SPI_1, SPI_2). The basic logic design of this unit is shown in Fig. 4.

c) *$Switch_{out}$:* This logic control unit establishes proper links from the outputs of the spare processors (SPO_1, SPO_2) back to inputs of the nonfaulty pipelined processors succeeding the faulty ones. The basic logic design of this unit is shown in Fig. 5. In this design, when one of the demultiplexer (DEMUX's) is disabled, all its outputs are equal to 0. Also, the shown OR gate symbols represent banks of OR gates.

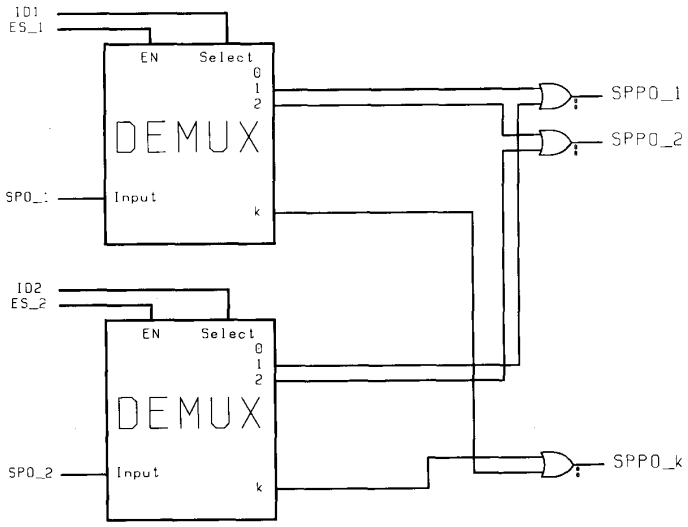

Fig. 5. Switch$_{out}$ basic design.

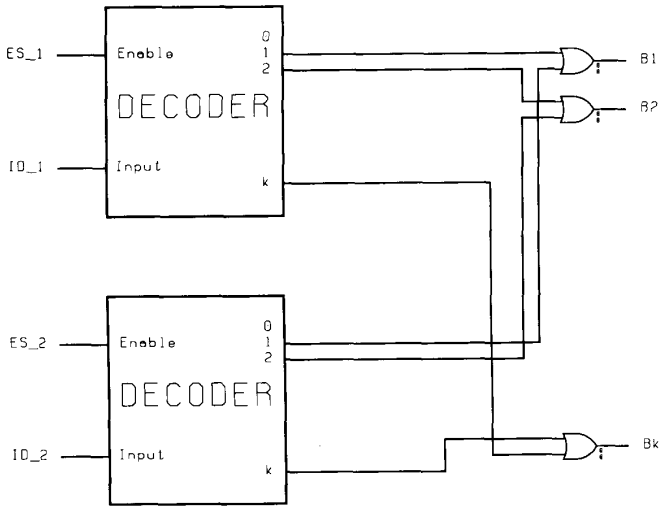

Fig. 6. Selection (bypass) generator basic design.

d) *Selection generator:* This combinational logic unit is used to generate selection signals (B_i, where $i = 0,1,\ldots,k$), which are used in turn by the steering logic at the input of each pipelined processor to reroute the tokens intended for the faulty processor to the spare processor. The steering logic is simply a MUX that is set to appropriately control the flow of information through the pipeline before and after the reconfiguration process. The basic design of this unit is shown in Fig. 6.

e) *Spare processors:* These processors are identical to the processors used in the pipeline. Whenever one or two faults are detected, they can be used to replace the faulty processors.

f) *Pipeline input unit (PIU):* This unit controls the inputs to the pipeline, depending on the state of the system. In normal operation, it supplies image information to the pipelined processors, while during recovery from one or two faults, it prohibits the input from the image memory (IM) to the system.

g) *Software switcher:* This unit is responsible for downloading spare processors with the states and programs of faulty processors in order to take over their role in the pipeline after recovery. It mainly consists of a general-purpose processor and an attached memory. The processor is interrupted by the recovery signal (R) issued by SVC once recovery from a fault is required. The interrupt service routine (ISR) performs the necessary downloading. The attached memory stores a duplicate of the programs of all pipeline processors. Also, it contains a look-up table for storing the starting and ending addresses of each

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.