NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

ISBN 0-315-53214-9

Canada

Application of Pseudorandom Binary Sequences to the Absolute Position Measurement of Automated Guided Vehicles

by

Jagdeep S. Basran

A Thesis
submitted to the School of Graduate Studies
in partial fulfillment of the requirements for the
Degree of Master of Applied Science
in
Electrical Engineering

at the
University of Ottawa
July 1989

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

Abstract

Within the modern manufacturing environment, there is a multitude of different types of equipment cooperating to manufacture today's end products. The Automated Guided Vehicles (AGVs), which are an integral part of such a factory, are the current state of the art with respect to material handling systems. These AGVs, which are frequently interacting with other manufacturing units (robots, machine tools, loading/unloading devices, etc.), would benefit from an ability to measure their absolute position along the guidepath. This position measurement capability could improve the existing AGV's ability to make routing decisions and handle the problems associated with their traffic management.

In this thesis the application of pseudorandom binary sequences to the absolute position measurement of an AGV will be explored and investigated. In particular, two new techniques that enhance the applicability of pseudorandom encoding of position, as a cost-effective upgrading feature for the industrial AGVs, will be discussed.

Acknowledgements

I wish to gratefully thank Emil for what I have learned from your natural enthusiasm towards science and your work. I will always treasure your scintillating discussions and I hope that our friendship will continue for the years to come.

I wish to thank NSERC and the University of Ottawa for the financial support that was provided to me through their scholarship programs.

I am also grateful to Jean-Serge Cardinal for his practical advice and assistance towards completing the implementation of the experimental AGV system.

A heartfelt thank you to Ira for allowing me the indulgence of sharing my thesis problems with you, and for all the pleasureable and serious conversations that we had on topics too numerous to mention.

My greatest appreciation goes to all the members of my family - Sharan, Cindy, Paul, Mom, and Dad - who each in their own way have provided me with special support and encouragement. My parents are deserving of the most gratitude for without their support I wouldn't be where I am today. In particular, I would like to thank my father for the value that he placed in education, shown through his willingness to financially support me throughout my university studies.

Table of Contents

Chapter 1 - Introduction

1.1 Background

The field of robotics is presently eliciting a widespread interest and being recognized as a formal discipline of study and research. Robotics, as a system science, has come to encompass a very broad range of subjects requiring the talents of people from several disciplines of study, such as kinematics, dynamics, planning systems, control, sensing, programming languages, and machine intelligence. An important reason for the marked growth in this field is the increase in the different applications for which robotic abilities are being considered. The pervasiveness of robotics is evidenced by the following list of potential applications:

- decommissioning nuclear power plants
- patrolling a prison
- mining precious metals on the ocean bed
- mechanical "guide dogs" for the blind
- office mail delivery system
- spatial tasks
- domestic servant
- manufacturing plant tasks
- fast food work

Robots are part of an evolving technology that has concentrated on developing machines that do work, usually done by people. But how did the idea for the tasks of robots develop to include such a diverse scope?

The word robot is derived from *robota*, the Czech word for serf or forced labor, and was introduced to the English language by the Czech playwright, Karl Capek, in his play, R.U.R. (Rossum's Universal Robots). In this play, a factory owner, called Rossum, turns out machines that resemble people but differ in that they have a wonderful capacity for work. These robots become more advanced and eventually revolt and overcome their masters. This fictional play along with the subsequent viewpoints of several authors is responsible for the views popularly held towards robots to this day, including the perception of robots as humanlike machines endowed with intelligence and individual personalities.

1.2 Robotic Technology Today

Although the word robot has come to encompass a vast number of ideas, there is no precise single definition of this word. However, a reasonable general definition of robot would be a programmable machine that, in some aspect of appearance or behavior, imitates men or women. In this context the android can be considered as a special form of robot, which most strongly agrees with the original introduction of the word *robota* to the English language. The computer controlled mechanical arms of the factory floor can be categorized as robots because they act, at least in part, like a person. Another type of robot is the telechiric device, which is a mechanical arm that works under the command of an operator who sits some considerable distance from the actual assembly. The person who operates this device is called a teleoperator.

An important point with respect to the independence of different types of robotic systems, as shown in Figure 1, is the general notion that there are two extremes of operation for these systems. In the one extreme, in order to make it possible for a robot to be completely autonomous,

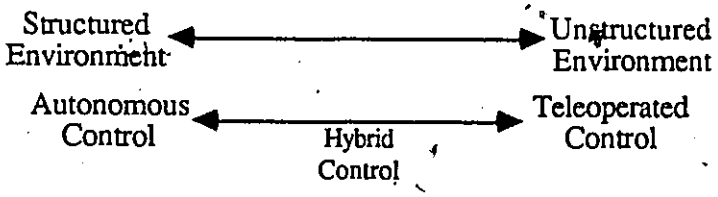


Figure 1.1 - Range of Robotic Systems

it is necessary to structure the environment. For example, the industrial robot is capable of working autonomously in a manufacturing plant because of the limitations imposed on the possible events in the system. On the other hand, teleoperation - a human in the feedback path - is required in order to cope with a completely unstructured world. It is also worth noting that between these two extremes, autonomous and teleoperated, lies the valuable hybrid approach which allows a human supervisor to intervene when necessary. The general research trend seems to be towards

the development of robotic systems capable of operating independently in more unstructured environments.

It is important to understand the meaning and implications of the words *autonomy* and *structure* with respect to robotic systems. Autonomy refers to the ability of such a system to perform actions without human intervention. The quality and the kind of actions that can be performed by the robot will determine the degree of autonomy of the system. Therefore, through an improvement in autonomy the robot should be able to complete tasks more effectively and efficiently. Two factors that can improve the self-action ability of a robotic system are the quality and quantity of input data from the environment, and the capacity to process that input data. In this context, structuring of the environment can refer to the providing of information, external to the robot and upon which the robot can base decisions. But, other structures in the environment, instead of providing input data to facilitate decisions, simply constrain robot motion to make it manageable. This distinction between different types of structure and their effects is important because in the first sense, the autonomy of the system is being improved, whereas in the latter it is being removed. As a final point, structuring to provide input data must be in a physical form readily able to be perceived by a robotic system.

Having given a brief overview of the broad field of robotics, it would be appropriate to address the specific area of interest for this thesis - the manufacturing environment. Historically, the first attempt to automate existing manufacturing facilities relied on the introduction of automatic machinery to continually repeat a specific operation. Although this method, called "hard automation", is effective for very large batches of products, it lacks flexibility and adaptability. The industrial robot, which is defined by the Robotics Industries Association as:

> a programmable multifunctional manipulator designed to move materials, parts, tools, or specialized devices through variable programmed motions for the performance of a variety of tasks

and, which was first introduced in 1959 by Unimation, has produced a different set of operating procedures for the factory. The programmable capability of the industrial robot has promoted the

development of the idea of "soft automation", which is effective for product variations and small to medium batch production.

1.3 Automated Guided Vehicles

Other machines, which have promoted the modern concepts of "soft automation", are computer numerically controlled machines and automated guided vehicles (AGVs). Of particular interest are the recent developments of the AGV as a critical part of any modern flexible manufacturing system. To begin with, a strict definition of an AGV from the Material Handling Institute is:

> a vehicle equipped with automatic guidance equipment, either electromagnetic or optical. Such a vehicle is capable of following prescribed guidepaths and may be equipped for vehicle programming and stop selection, blocking, and any other special functions required by the system.

That is to say, an AGV is a vehicle with a limited degree of autonomy in an orderly environment and it is quite distinct from an industrial robot. The difference is that an AGV follows a guidepath while a robot does not and an AGV without a robot arm cannot be considered to be a manipulator. Just as the different robotic terms can be confusing, there is a similar problem with the vehicular terms. In order to remain clear, a concise definition of the associated vehicular terms is necessary. A free-ranging AGV is defined as an AGV with a degree of autonomy such that a physical guidepath is not necessary. A mobile robot is defined as an AGV with an industrial robot mounted on it.

Having focussed the discussion from the broad field of robotics onto the specific topic of interest, AGVs within a manufacturing environment, it can be stated that the essential point of this thesis is to improve the autonomic ability of each AGV by the addition of structured information to the environment. The next chapter discusses the motivations for further pursuing this subject, an evaluation of the existing technology, and a presentation of the proposed enhancement.

CHAPTER 2 -- Justification for the Research

2.1 Motivations

Having established that the target application area is AGVs within a manufacturing environment, a further discussion with respect to the significance of these devices would be appropriate. As was stated in the previous chapter, the concept of "soft automation" is important for current manufacturing systems. More specifically, flexibility and adaptability are becoming almost absolutely necessary for the factory floor [5]. Flexibility in manufacturing allows rapid responses to be made to changing demands and circumstances (optimizing and adapting production schedules as a function of resource availability, producing variants of similar products, responding to changes in customer demands, etc.). Adaptability in manufacturing allows today's systems to make future changes at a reasonable cost (coping with short product life cycles, responding to market trends, etc.). Automated guided vehicle systems (AGVSs) are an important research and development topic as they represent a promising technology that can improve the current state of flexibility and adaptability on the factory floor.

The specific ways in which AGVSs can or do support these automation principles in the present industrial situation should be presented [5→11]. Firstly, because the mapped routes or physical guidepaths can be reconfigured, it is fairly easy to adapt to changes in product design and factory machine layouts without major further capital expenditure. Secondly, the control over the factory machine layout allows complicated networks of material flow with alternate or parallel routes between processes. This redundancy improves reliability by ensuring that individual breakdowns do not disturb the whole system. Thirdly, the programmability of the AGV is a basic versatility which makes variable product routing and asynchronous assembly possible. Fourthly, a diversity of AGVs can follow the same guidepath and perform different functions (interface with different types of automatic equipment, etc.). Fifthly, AGVSs provide adaptability by not being obstructive, i.e. taking up valuable floor space and presenting a barrier to the free movement of personnel. Sixthly, these systems can operate in a hazardous or clean-room environments.

Finally, the capability of full computer control and system integration into an overall production management computer is a notable advantage of AGVSs.

Hence, there are several obvious ways in which AGVSs currently promote the general intention of "soft automation", but as with most systems there is room for improvement. Basic improvements in AGV guidance capabilities is a topic which has received a considerable amount of interest for being able to extend the current trends of AGVSs. A close attention has been paid to improving guidance techniques since, with this approach, there are potential implications from the lowest level of AGV routing on to the highest level of system accountability.

2.2 Evaluation of Existing AGV Navigation Techniques

As its name implies, an AGV requires a navigation system which generally must be composed of two parts. One part, which is external to the AGV, refers to the guiding structures set up within the manufacturing environment. Another part, residing within each AGV, allows the AGV to determine its location in the surroundings by using the structures of the first part.

The pros and cons of seven principle guidance techniques will be discussed independently [12, 13]. These principle methods can be divided into two groups upon the basis shown below:

physical guidepath (AGV)
- mechanical
- inductive
- optical/chemical/magnetic

virtual guidepath (free ranging AGV)
- dead reckoning
- beacon system
- inertial navigation system
- direct imaging system

The virtual guidepath refers to a guidepath that must be derived from indirect indicators by the AGVs. In order to make the evaluation, it is necessary to enumerate the performance criteria to be applied to each guidance technique and the different qualitative terms to be used for rating purposes.

- *Range* - How far can an AGV be guided by the technology considered?
- *Accuracy* - By how much will the track of the AGV deviate from the position of an ideal guidepath?
- *Modification* - How much effort and cost is necessary to move a guidepath in order to adapt to changed operational requirements?
- *Reliability* - What is the probability of a malfunction of the navigation system considered in industrial use?
- *Controllability* - What are the means of traffic control intrinsic to the navigation system?
- *Vehicle equipment cost* - What is the cost of the navigation equipment needed aboard the AGV based on an estimation of complexity and maintenance requirements?
- *Stationary equipment cost* - What is the cost of the navigation equipment infrastructure needed based on an estimation of its complexity and maintenance requirements?
- *Range for the performance qualification words :*

  Poor ----- Acceptable ----- Medium ----- Good ----- Excellent

Having presented a list of the evaluation criteria and the verbal descriptive scale, a brief performance assessment of each of the above stated navigation techniques will be given.

2.2.1 Physical Guidepath AGVs

For these methods, the external structuring of the environment refers to a physical form of the guidepath, and the necessary stop/start places for servicing the manufacturing equipment.

Mechanical Guidepath

This technique uses some form of rails, directly or indirectly, to guide the vehicle. Although strong (good or excellent) in the areas of range, accuracy, reliability, and vehicle equipment cost, this method is particularly weak (poor or acceptable) in modification, controllability and stationary equipment cost. In short, the mechanical guidepath is of minor importance in modern AGV systems and it is considered here only for the sake of completeness.

Inductive Guidepath

Presently, this is the most widely used guidance technique for industrial AGVs. This method uses a network of buried cables, arranged in the form of closed loops, each with an alternating current (a.c.) signal of a different frequency. By detecting this current via electromagnetic induction, it is possible to generate an error signal which is used to steer the AGV such that it follows the guidepath. In such networks as well as steering cables, communication cables, which allow linking between each AGV and a network supervisory computer, are available.

The range of this method is good but it is limited by the drive capability of the a.c. current amplifier used. The accuracy of this method is good but it is limited by tolerances in cutting the wire-slot into the floor and by distortions of the electromagnetic field caused by metallic objects in the vicinity of the guidepath. The reliability and the controllability of this system are excellent while the vehicle equipment cost is good. The weaknesses (medium) of this approach are in modification and stationary equipment cost, i.e. it is difficult and expensive to reconfigure the mapped routes of the AGVS.

Optical/Chemical/Magnetic Guidepath

In this case, the system that is involved is similar to wire-guided AGVs except that these systems follow passive stripes on the floor. Like wire-guided systems an error signal, using proximity sensors, is generated to allow the AGV to track the guidepath.

The advantages and disadvantages of this method are considerably different from the previous one. The passive nature of the guidepath ensures that the range performance, which is excellent, is not affected by distance. The stationary equipment cost is also excellent because the cost per unit length is the lowest of all physical guidepath systems. The modification is good because it is easier to change the existing network layout for guidepaths, that are attached to the surface of the floor by painting or adhesive bonding. Since this approach has a guidepath which is vulnerable to the cleanliness of the environment, the reliability only has a medium rating. For

similar reasons the vehicle equipment cost has the same qualitative value. The biggest deficiency is with the poor controllability of this system which is caused by the passive nature of the track and the absence of an inherent communication ability.

2.2.2 Free-Ranging AGV Methods

These methods require that each AGV is capable of somehow estimating its absolute position and orientation within the environment. This coordinative information is critical to the development and following of the virtual guidepath. The structuring of the environment for these methods refers to the reference points or landmarks for absolute location identification. With these navigation techniques, the modification and stationary equipment cost will be assumed to be excellent, unless stated otherwise, due to the fact that a physical guidepath is not necessary.

Dead Reckoning

Dead reckoning is a technique where the relative position and orientation of a vehicle is tracked by using an optical encoder or resolver to measure the precise rotation of each drive wheel. With these relative values it is possible to extrapolate an AGV's movement from a known reference point without a physical guidepath.

With only the floor quality playing an important part in the reliability assessment, it is good. Since the equipment required for this approach is already available with most modern AGVs, the vehicle equipment cost is good. The problems with this method are with the range, accuracy, and controllability. The range is poor because dead reckoning is essentially an open loop system where the errors of the odometric system are caused by wheel slippage, floor unevenness, etc. The accuracy which is strongly interdependent with the range is acceptable for limited distances. The incapacity for communication is the primary reason for the inferior (poor) controllability.

Beacon Systems

Several beacons are fixed at appropriate locations in the surroundings. If the AGV knows the precise positions of these beacons and it can measure its exact distance and direction from any one beacon, then it can calculate its own location in the environment.

The range and accuracy are both dependent on the distance of the AGV from the beacon. This problem is further amplified by the fact that beacons require unobstructed line of sight for proper operation which in the end results in the range and accuracy having a medium rating. The controllability and vehicle equipment cost are similarly medium but the main drawbacks are with the acceptable opinion for the reliability and stationary equipment cost. The main reason for the reliability inadequacy is because of drift effect; whereas the expensive price of beacon transmitter/receiver systems produces the low grade for the stationary equipment cost.

Inertial Navigation Systems

This method measures translational and rotational acceleration of the AGV and then deduces its position and heading relative to a reference point by double integration of the respective accelerations. The physical setup can consist of a gyroscope which has been located on the AGV such that it is parallel to the direction of motion on the vehicle. The detected gyroscopic variations, which record the history of vehicle motion, are utilized to measure the relative position and orientation with respect to a reference location and thereby, estimate the AGV's absolute location.

Because of the incremental nature of locational determination which is prone to drift, the range suffers and is given a medium rank. Another fundamental difficulty is with the cost of acceleration detection equipment, which manifests itself in a poor vehicle equipment cost. The controllability has a poor rating because there is no means inherently available for supervisory control. Finally, the difficulty of keeping gyroscopes working effectively under vibration leads to acceptable accuracy and medium reliability.

Direct Imaging Systems

In this approach, optical or ultrasound images of the surroundings are used in conjunction with internal environmental maps and/or easily identifiable landmarks to obtain the AGV's position and orientation.

Range and stationary equipment cost have good ratings, while accuracy has an excellent rating. Controllability has an acceptable rating because of the possibility of data communication via the optical or ultrasound imaging carrier. The reliability is medium because of differences between the actual image of the surroundings and the captured stored image. Since the imaging sensors are unreliable, not sufficiently precise or too expensive, the vehicle equipment cost for these systems is also exorbitant (poor).

2.2.3 Synopsis and Further Assessment

From the previous evaluation, it is limpid that each navigation system is deficient in certain ways, and no one system possesses all desired qualities. Presently, there is a considerable effort towards developing free-ranging AGVs because they are strongly considered to be the answer for the next generation of AGV systems. Because none of the above free-ranging AGV systems is completely adequate, several current papers [15→22] have proposed a combination of the above techniques. The essential principle with these approaches is the use of a cumulative position and orientation measurement technique (inertial, dead reckoning, etc.) in conjunction with an absolute position and orientation reference system (beacon system, direct imaging, etc.). The basic idea is to occasionally reset the error-prone relative location method by the absolute location measurement of the AGV. For example, dead reckoning in combination with a beacon system or direct imaging system have been proposed and are being developed as an improved free-ranging AGV navigation technique. A fundamental limitation with these approaches is the fact that due to economic reasons only a limited number of reference points can be used.

Although free-ranging AGVs are creating a strong interest, it is also important to note four underlying points. One of the strongest motivators for this approach is the elimination of the cost of fixed guidepath installation but, more accurately, there is a cost tradeoff between eliminating the fixed guidepath and adding navigational sensors to the vehicle. Secondly, although this method provides the ability for unconstrained movement, there will still be a need to plan the AGV courses systematically so that the vehicles work together effectively in common areas for higher overall productivity. The point being made is that self-navigating AGVs may provide more flexibility than is usually needed in an manufacturing plant, where, once laid out, AGV routes are not likely to change, since the massive workstation equipment they service can't be moved and rearranged anyway. Thirdly, the problem of finding the orientation with respect to the environment is not necessary for an AGV following a physical guidepath. As a final point, there are very few industrial AGVs which have incorporated these free-ranging principles. In short, it is not absolutely necessary for free-ranging AGVs to be used within a manufacturing environment.

Having presented a general survey of the existing guidance methods, the next section presents the specific approach of this thesis to this guidance problem and the implications of this approach.

2.3 The Proposed Enhancement

From the earlier examination of the various AGV guidance techniques, it is clear that the crux of the guidance problem is in being able to determine the AGV's position and orientation, quickly and effectively. The determination of this locational knowledge, which is generally simplified by partially structuring the environment, is necessary for a greater degree of autonomous operation. It would be worthwhile to develop a simple, noninvasive, inexpensive, structuring system which could be easily implemented in a wide variety of manufacturing environments. One such proposed system is based upon the use of standard pattern to provide a unique identification code (using bar codes) that enables the system to find the absolute location of the pattern [23, 24].

Although useful this approach still suffers from the limited number of reference points problem which will be clarified in the next paragraph.

The proposed enhancement, which relieves the problem of limited number of reference points and is an upgrading feature for physical guidepath AGVs, involves a positional coding procedure based upon the properties of pseudorandom binary sequences (PRBSs). Figure 2.1 clarifies the guidepath encoding problem. As can be seen one possibility is to mark either transversally (Fig. 2.1(a)) or longitudinally (Fig. 2.1(b)) relative to the guidepath, a distinct n-bit code word for each quantization step. These approaches are inefficient because of the redundant

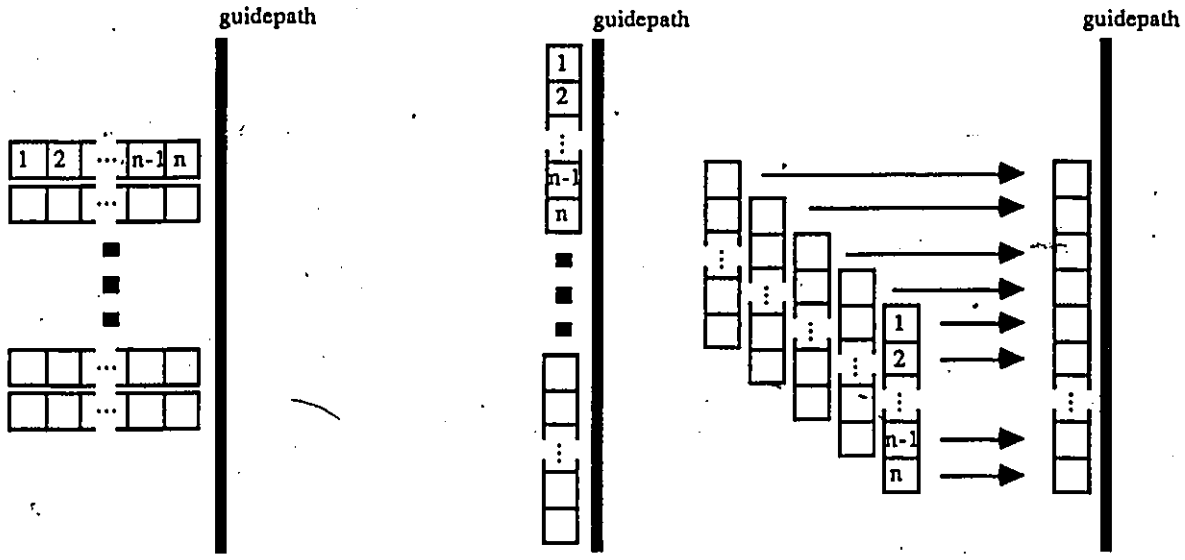


Figure 2.1 - The Simplicity and Cost Advantage of PRBSs

information stored at different locations and it becomes especially cumbersome for a larger number of reference points. On the other hand the PRBS encoding procedure (Fig. 2.1(c)), which is applied longitudinally, requires only 1 bit per quantization step regardless of the number of resolution bits required. This special property is possible because the first (n-1) bits of the current

PRBS code is identical with the last (n-1) bits of the previous PRBS code. Furthermore, this means that each window of size n scanning the PRBS code is unique and identifies a specific location in the environment. Therefore, irrespective of all other factors, the PRBS solution is a very cost-effective approach for guidepath encoding.

2.3.1 Potential Implications/Benefits of this Approach

By essentially removing the restriction on the number of absolute reference points, the absolute positional measurement along an entire guidepath becomes possible. That is, the PRBS encoding can be added as an upgrading feature to the existing guidepaths so that an AGV can measure its absolute position anywhere along the guidepath. Note that the orientation is intrinsically known from the AGV's perspective of the guidepath and code track arrangement.

Assuming for the moment that this positional information can be made available on a real-time basis then a number of benefits can be stated. Firstly, the vehicles are self-correcting in that the vehicle can leave or be taken off the guidepath and put back at any point without having to go through extensive reset procedures. The AGV may leave the guidepath in order to avoid obstacles or other AGVs, off-line work, etc. (Figure 2.2). This is a fundamental improvement in the navigational capability of each individual AGV. This navigational improvement makes the use of a bi-directional guidepath (a single guidepath where vehicles can travel in both directions) appealing [29]. In particular the problems of leaving the guidepath and then returning to it to avoid another AGV and of performing bi-directional network control are made simpler. Secondly, the network control is made easier for the case of optical/chemical/magnetic guidepath because knowing its absolute position and aided by an environmental map, the AGV can quite easily decide which fork in the road to take. Thirdly, this PRBS encoding can be useful for docking purposes because it

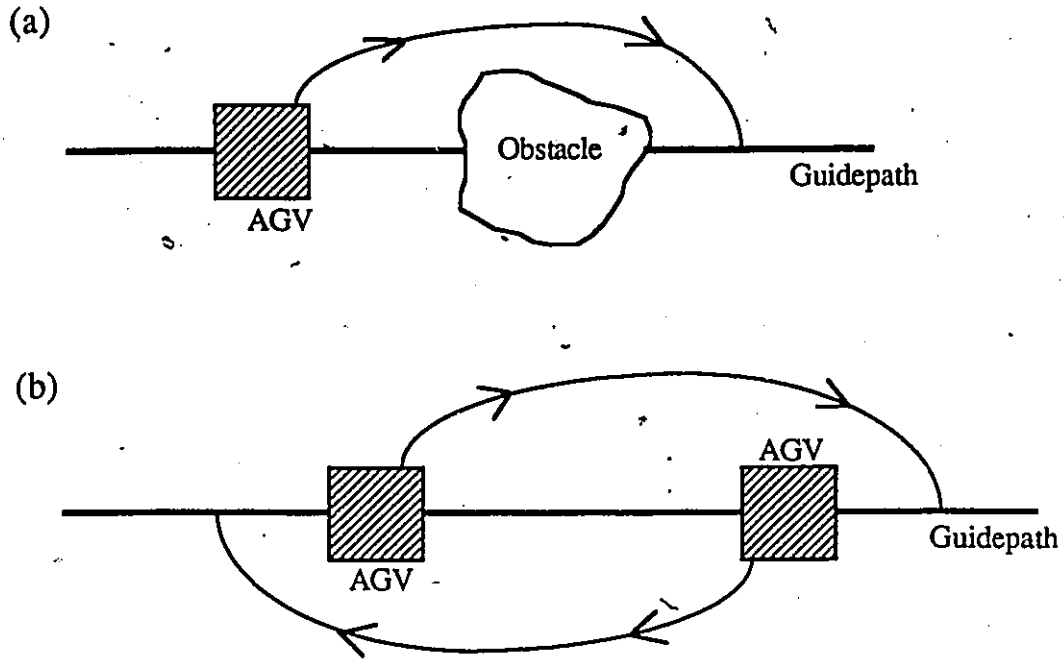


Figure 2.2 - Potential AGV Navigational Improvements

can provide for variable and accurate docking capability. This asset is particularly suitable for mobile robots (AGV with robot affixed) where a strong need for this capability exists [26, 27].

Underlying the above discussion is the effect on the traffic control of multiple AGVs in a manufacturing environment [33→35]. By providing this locational information the AGVS promotes a decentralized traffic control policy, where each AGV has the autonomy to select its own route to serve the common good of the overall manufacturing environment. The operations which each vehicle must keep track of include its own status, position, and task sequence of operation. A further point is that with this encoding method and the above information available onboard each vehicle, then the recovery from any forced down time of the central computer could be executed simply and quickly. With this decentralized approach each AGV can act as an autonomous active object within the dynamic manufacturing framework. It is also worthy of note that real-time monitoring at the system management level is possible.

Shown by Figure 2.3 is a possibile environment where each AGV could move with a greater degree of independence.

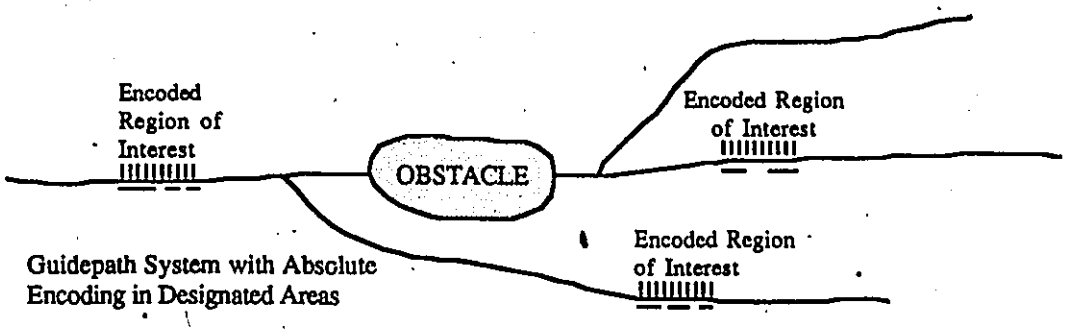


Figure 2.3 - Perceived PRBS Encoded Environment

In summary PRBS encoding by improving the navigational capabilities of each AGV can increase the flexibility of existing physical guidepath AGVs and therefore, supports the idea of distributed processing control within the manufacturing environment.

Having presented and established the value of the proposed approach, the theoretical background, the practical considerations, and the details of implementation will be the subjects of the subsequent chapters.

CHAPTER 3 - Pseudorandom Binary Sequences

3.1 General Information

Pseudorandom binary sequences (PRBSs), which have been known for a long time, have a wide range of technical applications, including:

- range-finding
- coding theory - scrambling, error correcting codes
- communications - modulation, synchronizing
- electronics - high speed counters [36→39].

These sequences have special properties that make them applicable in a multitude of fields. The AGV absolute position encoding method, that was proposed in the last chapter, is based upon one of the characteristics of PRBSs. In the context of this specific approach, the background of PRBSs along with the most important of their useful properties will be presented.

3.2 PRBS Theory

Pseudorandom binary sequences, which are also called pseudo-noise sequences, maximal-length shift register sequences (m-sequences in short), etc., are deterministic strings of binary digits (bits) that have statistical properties similar to those of true random sequences. The general idea behind the generation of any binary sequences is exemplified in Figure 3.1. As can be seen there is a shift register consisting of n stages, representing binary memory elements. At each time

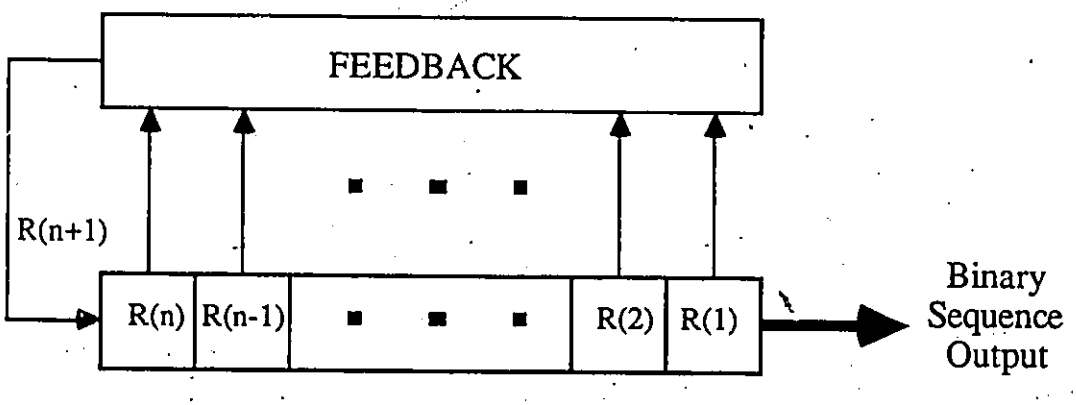


Figure 3.1 - General Binary Sequence Generator

unit, the contents of these binary memories are shifted one place to the right and the present value of the feedback, R(n+1), is placed into the leftmost memory element. The bits which are shifted out of the rightmost box represent the values of an infinite binary sequence.

A critical part of this PRBS generation is the type of feedback that will be used. The feedback is performed using modulo two addition or an exclusive or gate, defined by $0+0=1+1=0$, $0+1=1+0=1$. But it is still necessary to determine which terms of the n-bit shift register will be used in this feedback process. This problem can be restated as follows; what values does the array c need in the following equation (where all symbols refer to logical operations):

$$R(n+1) = c(n) \cdot R(n) \oplus c(n-1) \cdot R(n-1) \ldots c(2) \cdot R(2) \oplus c(1) \cdot R(1)$$

to produce a PRBS. The determination of the array c specifies the particular feedback connections to be made with the shift register.

The resolving of this matter can be achieved by exploring the area of polynomials. That is, a polynomial of degree n can be used to delineate the feedback elements required for an n-bit shift register. Consider the following particular form of a polynomial equation (where all symbols refer

$$f(x) = x^n + c_n \cdot x^{n-1} + \ldots + c_2 \cdot x^1 + c_1 \cdot x^0$$

to arithmetic operations) where the subscripted c's directly correspond to the previous array of c's and the power of the variable x refers to a location within the shift register ($x^{n-1} \leftrightarrow R(n)$). With this correspondence explicated, the important connection is that in order to construct a PRBS sequence of length $2^n - 1$, one needs a primitive polynomial of degree n. A primitive polynomial of degree n clearly designates the desired feedback shift register relationship.

Consider now the specific example of degree n=4, with the following primitive polynomial:

$$p(x) = x^4 + x + 1$$

which specifies a feedback shift register as shown in Figure 3.2. Notice the direct correspondence between the primitive polynomial and the feedback connections that are made.

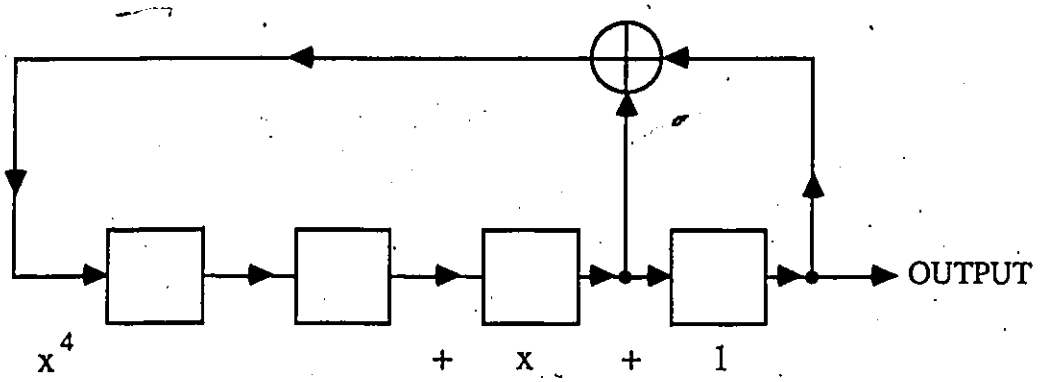


Figure 3.2 - Feedback Shift Register corresponding to $x^4 + x + 1$

In the particular case where the register contains a_{i+3}, a_{i+2}, a_{i+1}, a_i, at time i, then at time i+1 it contains

$$a_{i+4} = a_{i+1} \oplus a_i, a_{i+3}, a_{i+2}, a_{i+1}$$

as shown in Figure 3.3. This specific feedback relationship is used to generate the subsequent bits which will be shifted into the register. In summary, this feedback shift register generates an infinite sequence $a_0\ a_1\ a_2, \ldots\ a_i \ldots$ which satisfies the recurrence

$$a_{i+4} = a_{i+1} \oplus a_i, \quad i=0,1,2, \ldots$$

In order to produce the infinite sequence, it is necessary to specify the initial values for $a_0, a_1, \ldots, a_{n-1}$.

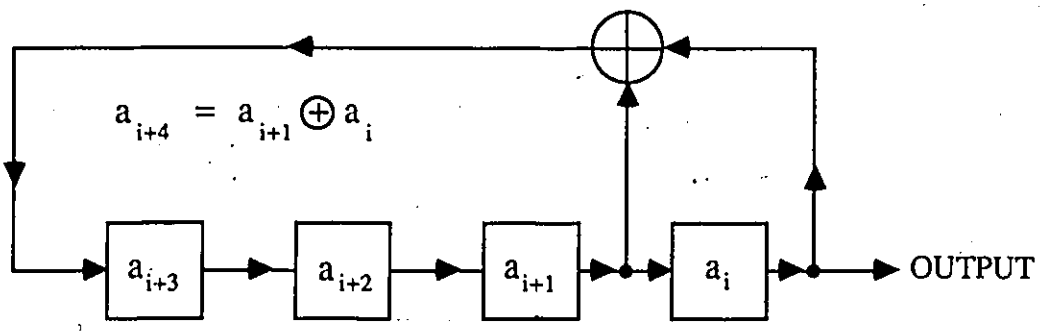


Figure 3.3 - The Specific Recurrence Relation

3.2.1 Specific Definition of a PRBS

The first step in this definition should be a further examination of this infinite sequence. Because each of the n-bits of the shift register can only be a 0 or 1, there are 2^n possible states which means that the infinite sequence $a_0\ a_1\ a_2 \ldots$ must be periodic. Moreover, with modulo two or exclusive or feedback, the zero state 00...00 cannot occur unless the infinite sequence is the trivial case of all zeros. The exclusion of the zero state results in the maximum possible period of a useful PRBS to be $2^n - 1$. For example, figure 3.4 shows the successive states and the periodic output sequence for primitive polynomial $p(x) = x^4 + x + 1$ and initial state = 1000.

STATE NO.	STATE			
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	1	0	0	1
4	1	1	0	0
5	0	1	1	0
6	1	0	1	1
7	0	1	0	1
8	1	0	1	0
9	1	1	0	1
10	1	1	1	0
11	1	1	1	1
12	0	1	1	1
13	0	0	1	1
14	0	0	0	1
15=0	1	0	0	0
16=1	0	1	0	0
	(REPEATS)			

Figure 3.4 - Successive States and Output Sequence for the previous example

Note: Although it will not be proved here, it is known [36, 39] that there exist primitive polynomials of degree n for every n. Table 1 at the end of this chapter has a list of such polynomials for $n \leq 40$.

If p(x) is a primitive polynomial of degree n, then the shift register goes through all $2^n - 1$ distinct nonzero states before repeating, and produces an output sequence $a_0\ a_1\ a_2 \ldots$ with period $2^n - 1$. A **pseudorandom binary sequence** is any segment

$$a_i\ a_{i+1} \ldots a_{i+2^n-2} \qquad (1)$$

of length $2^n - 1$. There are $2^n - 1$ different PRBSs for i=0, 1, ... $2^n - 2$. The different sequences corresponding to the previous example of degree 4 are given in Figure 3.5.

```
0 0 0 1 0 0 1 1 0 1 0 1 1 1 1
0 0 1 0 0 1 1 0 1 0 1 1 1 1 0
0 1 0 0 1 1 0 1 0 1 1 1 1 0 0
1 0 0 1 1 0 1 0 1 1 1 1 0 0 0
0 0 1 1 0 1 0 1 1 1 1 0 0 0 1
0 1 1 0 1 0 1 1 1 1 0 0 0 1 0
1 1 0 1 0 1 1 1 1 0 0 0 1 0 0
1 0 1 0 1 1 1 1 0 0 0 1 0 0 1
0 1 0 1 1 1 1 0 0 0 1 0 0 1 1
1 0 1 1 1 1 0 0 0 1 0 0 1 1 0
0 1 1 1 1 0 0 0 1 0 0 1 1 0 1
1 1 1 1 0 0 0 1 0 0 1 1 0 1 0
1 1 1 0 0 0 1 0 0 1 1 0 1 0 1
1 1 0 0 0 1 0 0 1 1 0 1 0 1 1
1 0 0 0 1 0 0 1 1 0 1 0 1 1 1
```

Figure 3.5 - The 15 ($2^4 - 1$) PRBSs for $p(x) = x^4 + x + 1$

3.2.2 Important Properties of Pseudorandom Binary Sequences

With the definition and the generative technique of a PRBS elucidated, the attributes of PRBSs can be expounded [36].

Let $p(x)$ = fixed primitive polynomial of degree n

δ_n = the set consisting of the PRBSs obtained from p(x) plus the sequence of $2^n - 1$ zeros.

Property I - The Shift Property

If $b = b_0 b_1 \ldots b_{2^n-2}$ is any PRBS in δ_n then any cyclic shift of b, say

$$b_j b_{j+1} \ldots b_{2^n-2}\ b_0 \ldots b_{j-1}$$

is also in δ_n.

Property II - The Recurrence

Suppose $p(x) = \sum^{n}_{i=0} h_i x^i$, with $h_0 = h_n = 1$, $h_i = 0$ or 1 for $0<i<n$. Now any PRBS $b \varepsilon \delta_n$ satisfies the recurrence

$$b_{i+n} = h_{n-1}\, b_{i+n-1} + h_{n-2}\, b_{i+n-2} + \ldots + h_1\, b_{i+1} + b_i$$

for i=0, 1, ... This expression is simply a generalization of the feedback equation which is obtained from the primitive polynomial and used with the feedback shift register.

Property III - The Window Property

If a window of width n is slid along a PRBS in δ_n, each of the 2^n - 1 nonzero binary n-tuples is seen exactly once. This property follows from the fact that p(x) is a primitive polynomial which is irreducible.

Property IV - Half 0's and Half 1's

Any nonzero PRBS sequence in δ_n contains 2^{n-1} 1's and 2^{n-1}-1 0's. This property is obvious from the fact that the shift register runs through all possible states except the zero state.

Property V - The Addition Property

The sum of two sequences in δ_n is another sequence in δ_n, if the sum is performed via modulo two addition.

Property VI - The Shift and Add Property

The sum of a PRBS and a cyclic shift of itself is another PRBS.

Property VII - Autocorrelation Function

The autocorrelation function of a PRBS of length 2^n - 1 is given by

$$t(0) = 1$$
$$t(i) = -1/n, \quad \text{for } 1 \le i \le 2^n - 2.$$

This is the best possible autocorrelation function of any binary sequence of length $2^n - 1$, in the sense of minimizing $\max_{0<i<n} t(i)$.

Property VIII - Runs (maximal string of consecutive identical symbols)

In an PRBS, one-half of the runs have length 1, one-quarter have length 2, one-eight have length 3, and so on, as long as the fractions give an integral number of runs.

3.3 Synopsis and Further Discussion

From the above given properties, it is possible to justify the qualification *pseudorandom* to these types of sequences. The tossing of a fair coin, $2^n - 1$ times, is an event which will produce a random sequence that can be compared with a PRBS. The fact that properties III, IV, VII, and VIII of PRBSs are expected to hold for this coin tossing experiment, justifies the word random for these sequences. But the discussed sequences are only *pseudorandom* because the properties that have been mentioned hold for every PRBS, whereas in a coin tossing experiment there would be some variation from sequence to sequence.

Of the aforementioned PRBS characteristics, the property which is vital to the position encoding technique for AGVs is the window property. With such a sequence, each n consecutive bits form a unique pattern; hence, may be used to fully identify the AGV's absolute position on the guidepath. A 5-bit PRBS code and the corresponding absolute position values are displayed in Figure 3.6.

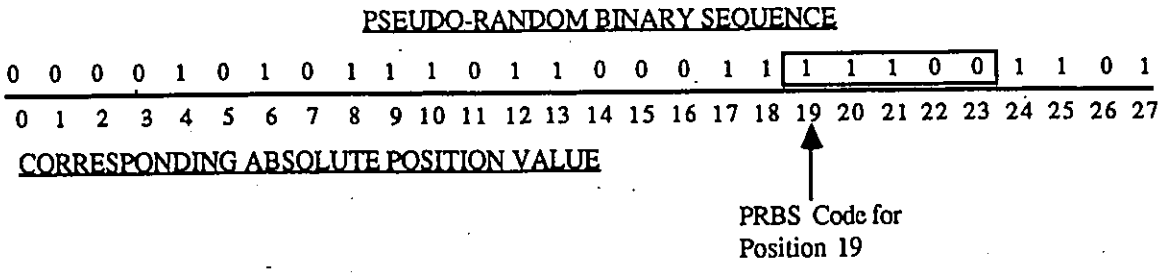


Figure 3.6 - The Window Principle to be used for AGV Position Encoding

Another salient feature of PRBS is the fact that there do exist complementary primitive polynomials. The generation of the PRBSs has been such that the resulting sequence shifts out the right end of the illustrated diagrams for a given primitive polynomial. It is also possible for a primitive polynomial to produce the same PRBS, but this time have the feedback entering at the right end while the output is being produced in the reverse order at the left end. This complementary, forward and reverse PRBS generation, is an underlying feature which will be utilized later. Table II summarizes the required modulo two forward and reverse feedback connections for shift registers with bit lengths from 4 to 10. These equations have been obtained from the complementary primitive polynomial expressions.

The practical considerations of making this absolute-type position encoding based on PRBSs possible will be the subject matter of the next chapter.

Table I - Primitive Polynomials for $n \leq 40$.

deg n	h(x)	deg n	h(x)
1	$x + 1$	21	$x^{21} + x^2 + 1$
2	$x^2 + x + 1$	22	$x^{22} + x + 1$
3	$x^3 + x + 1$	23	$x^{23} + x^5 + 1$
4	$x^4 + x + 1$	24	$x^{24} + x^4 + x^3 + x + 1$
5	$x^5 + x^2 + 1$	25	$x^{25} + x^3 + 1$
6	$x^6 + x + 1$	26	$x^{26} + x^8 + x^7 + x + 1$
7	$x^7 + x + 1$	27	$x^{27} + x^8 + x^7 + x + 1$
8	$x^8 + x^6 + x^5 + x + 1$	28	$x^{28} + x^3 + 1$
9	$x^9 + x^4 + 1$	29	$x^{29} + x^2 + 1$
10	$x^{10} + x^3 + 1$	30	$x^{30} + x^{16} + x^{15} + x + 1$
11	$x^{11} + x^2 + 1$	31	$x^{31} + x^3 + 1$
12	$x^{12} + x^7 + x^4 + x^3 + 1$	32	$x^{32} + x^{28} + x^{27} + x + 1$
13	$x^{13} + x^4 + x^3 + x + 1$	33	$x^{33} + x^{13} + 1$
14	$x^{14} + x^{12} + x^{11} + x + 1$	34	$x^{34} + x^{15} + x^{14} + x + 1$
15	$x^{15} + x + 1$	35	$x^{35} + x^2 + 1$
16	$x^{16} + x^5 + x^3 + x^2 + 1$	36	$x^{36} + x^{11} + 1$
17	$x^{17} + x^3 + 1$	37	$x^{37} + x^{12} + x^{10} + x^2 + 1$
18	$x^{18} + x^7 + 1$	38	$x^{38} + x^6 + x^5 + x + 1$
19	$x^{19} + x^6 + x^5 + x + 1$	39	$x^{39} + x^4 + 1$
20	$x^{20} + x^3 + 1$	40	$x^{40} + x^{21} + x^{19} + x^2 + 1$

Table II - Forward and Reverse Feedback Equations for PRBSs

Shift register length n	Feedback for forward p.r.b.s. $R(0) = R(n) \oplus c(n-1) \cdot R(n-1) \oplus \ldots \oplus c(1) \cdot R(1)$	Feedback for reverse p.r.b.s. $R(n+1) = R(1) \oplus b(2) \cdot R(2) \oplus \ldots \oplus b(n) \cdot R(n)$
4	$R(0) = R(4) \oplus R(1)$	$R(5) = R(1) \oplus R(2)$
5	$R(0) = R(5) \oplus R(2)$	$R(6) = R(1) \oplus R(3)$
6	$R(0) = R(6) \oplus R(1)$	$R(7) = R(1) \oplus R(2)$
7	$R(0) = R(7) \oplus R(3)$	$R(8) = R(1) \oplus R(4)$
8	$R(0) = R(8) \oplus R(4) \oplus R(3) \oplus R(2)$	$R(9) = R(1) \oplus R(3) \oplus R(4) \oplus R(5)$
9	$R(0) = R(9) \oplus R(4)$	$R(10) = R(1) \oplus R(5)$
10	$R(0) = R(10) \oplus R(3)$	$R(11) = R(1) \oplus R(4)$

CHAPTER 4 - Absolute Position Measurement Through PRBS Guidepath Encoding

4.1 Problem Introduction

With the viability of PRBS encoding for the absolute position measurement of AGVs ascertained, the practical aspects of this approach should be addressed. Figure 4.1 clearly illustrates the problem at hand. An n-bit PRBS of length $2^n - 1$, written one bit per quantization

$$\{S(j) \mid j=0, 1, \ldots, 2^n - 2\}$$

step, forms the pseudorandom code track.

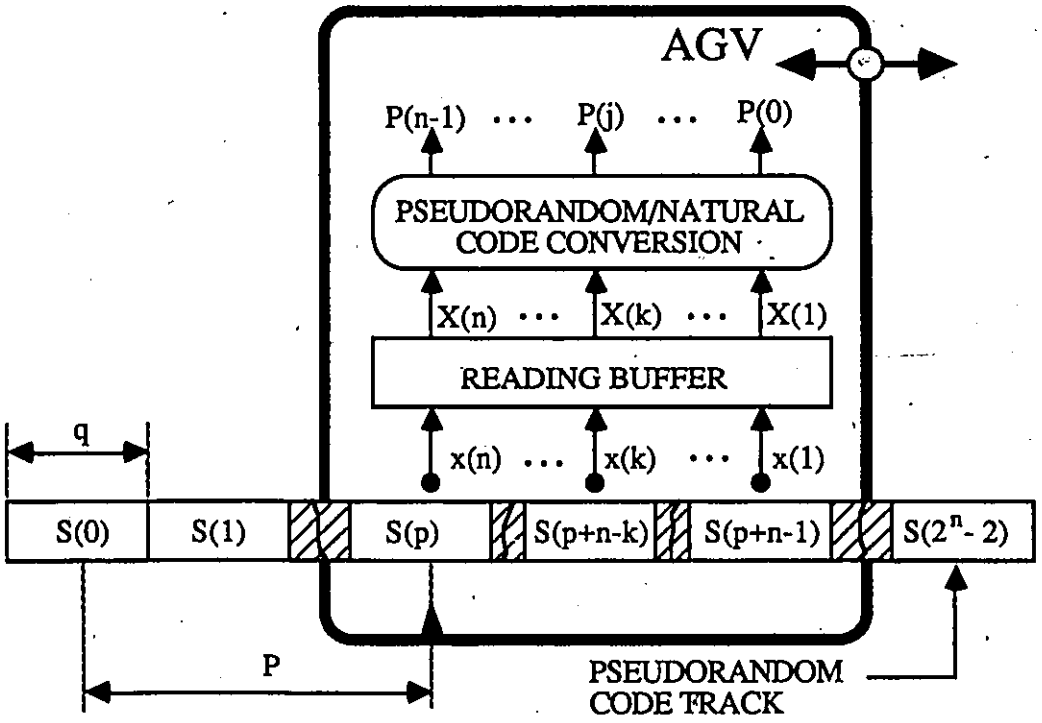


Figure 4.1 - The Principle of AGV Position Measurement using PRBS Encoding

The physical tracks, guidepath along with this pseudorandom code track, must be present external to the AGV. The first aspect of this problem is **reading the n consecutive bits**, $\{X(j) \mid j=1, 2, \ldots, n\}$, corresponding to the current AGV's location into the reading buffer. This scanning of

the coded information is dependent upon the direction of travel of the AGV. The pseudorandom n-tuple, which is available in the reading buffer, is not in a ready to use format and needs further processing. This point leads to the second practical aspect of consideration, **the code conversion of the scanned pseudorandom n-tuple**, $\{X(j) \mid j=1, 2, \ldots, n\}$, into the more convenient natural binary n-tuple form, $\{P(j) \mid j=0, 1, \ldots, n-1\}$. These two aspects of obtaining and converting the pseudorandom binary code are the topics which must be fully discussed in order to make PRBS encoding of the AGV's guidepath a practical alternative.

4.2 Code Reading

The two important factors of this stage of absolute position measurement are the generation of a load pulse to synchronize the reading of the coded information and the determination of the AGV's direction of travel on the guidepath to interpret the coded information correctly.

A straightforward way to solve the **load pulse problem** is to use an external synchronization provided by a special track laid along the PRBS code track. The first two methods to be discussed later use this approach. The third method that will be discussed uses an indirect but effective technique called a virtual synchronization track to solve the reading problem.

For the **directional problem**, a clarification of what is meant by the AGV's travelling direction is necessary. Supposing that the AGV has a front and back end and can drive itself either forward or backward, then Figure 4.2 displays all the different movement possibilities. The determination of direction refers to an AGV's awareness of whether it is going left or right, which are defined as follows:

left - the PRBS code, that is currently being measured, is progressing in the reverse feedback generation direction.

right - the PRBS code, that is currently being measured, is progressing in the forward feedback generation direction.

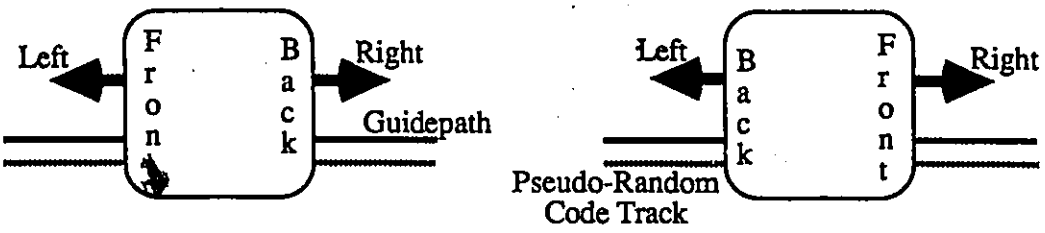


Figure 4.2 - AGV's Direction of Travel

Observe that the guidepath and code track layout do not have topological symmetry. The different approaches to determine direction generally use this asymmetric information from the reading heads and will be presented concomitantly with the synchronization methods discussed next.

4.2.1 Sampling Strips

An immediate implementation of the synchronization track, as shown in Figure 4.3, is to have sampling strips of value 1 fall in the middle of each quantization interval of the PRBS track [42]. Whenever such a strip arrives in the front of the synchronization reading head AUT, fixed opposite to the AGV's reference marker, the information delivered by the n code reading heads is considered to be stable and ready to be stored in the code reading buffer. The reading buffer information change happens on the rising edge of the AUT synchronization signal. Although it solves the reading ambiguities problem (the ambiguity facing all absolute-type position measuring techniques when a transition between two consecutive positions takes place), this solution introduces a systematic hysteresis error, $e = q_0 / 2$, where q_0 is the width of the space "0" between sampling strips. As displayed in Figure 4.3, the way in which this error affects the measured position, represented by the current contents of the reading buffer, is a function of the direction in which the AGV moves.

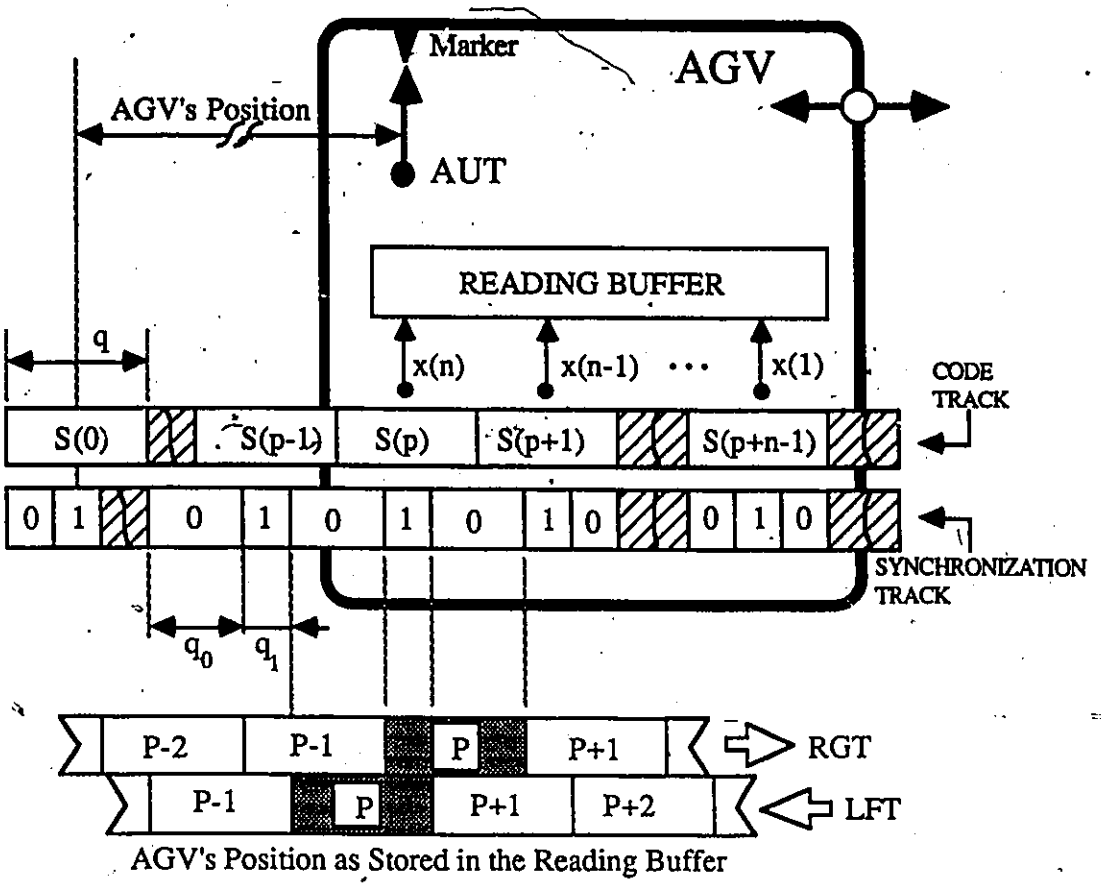


Figure 4.3 - The Sampling Strips Approach

For this reading approach, the orientation of the reading heads relative to the physical tracks combined with a priori directional information is one way to resolve the travelling direction of the AGV. For example, if it was found from the reading heads that the PRBS code track was to the left of the guidepath, then the AGV could be programmed to know that forward movement was right while reverse movement was left.

4.2.2 An Effective Physical Synchronization Track

In this case the synchronization track employs a simpler layout, consisting of a sequence of alternate "0" and "1" intervals each having the same width q as the quantization intervals of the PRBS code track [44]. Similar to the previous approach, the reading head AUT is employed to

detect the transitions between two consecutive quantization intervals by scanning the synchronization track. The information in the reading buffer is updated each time the AUT signal generates an appropriate clock signal by detecting an edge, either rising or falling. The n code reading heads, {X(j) | j=1, 2, ..., n}, are shifted by q/2 (to the left) relative to the AUT head. This ensures that the information provided by the code reading heads is always stable when AUT detects a transition between two consecutive positions. However, a new but correctable problem arises due to this non-aligned arrangement of the reading heads and the synchronization head.

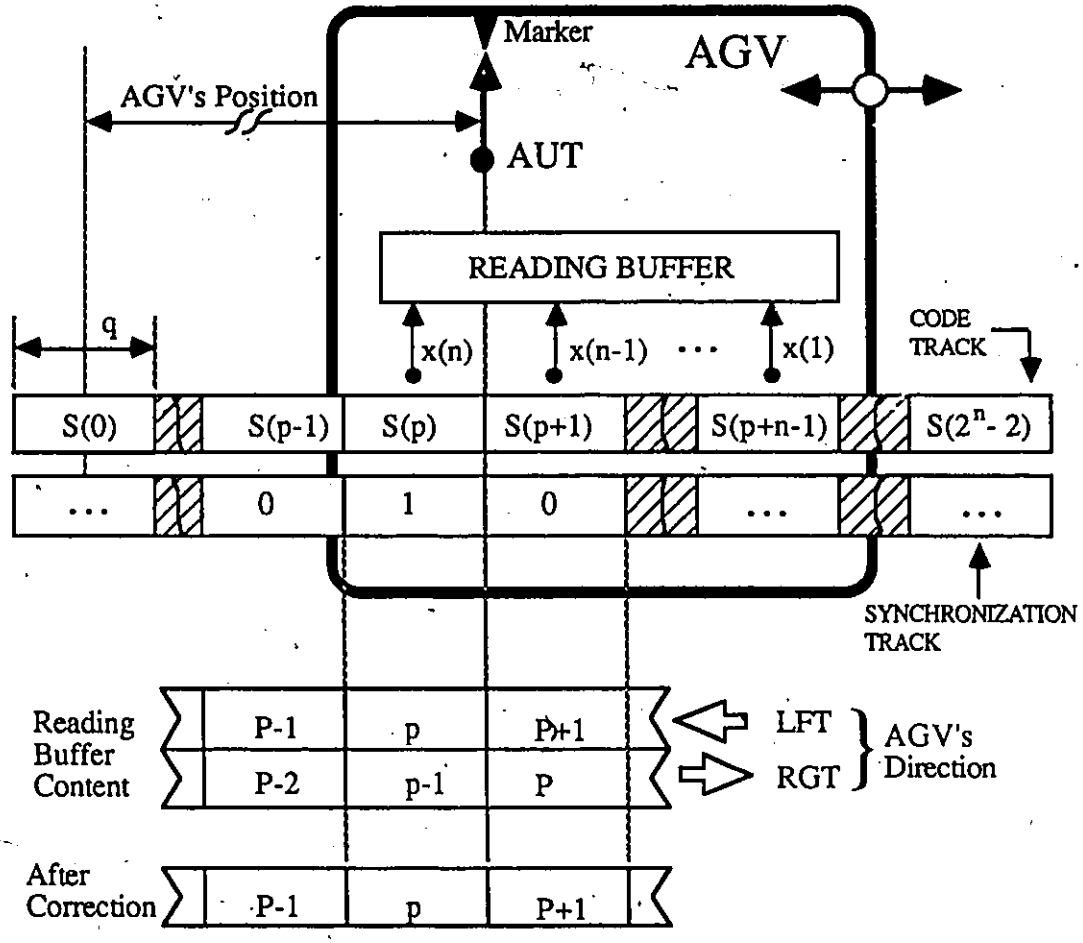


Figure 4.4 - Alternating Synchronization Track Approach

Let us consider the general case illustrated in Figure 4.4, where the AGV is at a transition point between positions p and p+1. Regardless of the AGV's moving direction, the n-tuple {x(k)

$= S(p+n-k) \mid k=n, \ldots, 1\}$, which will be stored in the reading buffer at the end of this transition, represents the pseudorandom code corresponding to the position p. If the transition occurs when the vehicle moves to the left, the new AGV's position will be p, which is correctly identified by the new content of the reading buffer. If the transition occurs during a move to the right, the new AGV's position will be position p+1 and the information from the reading buffer has to be incremented by one in order to represent the correct value of the position. It results that besides the pseudorandom information provided by the code reading heads, the position estimation procedure needs to know the direction in which the AGV travels on its guidepath. It can be presumed that the travelling directional information can be determined by the same technique that was used in the previous synchronization method.

A further improvement with this type of synchronization track is possible by the addition of a second heading read, VER placed at a distance q / 2 from AUT, to investigate the synchronization track, Figure 4.5. With such an arrangement the two heads, AUT and VER, may fulfill a supplementary function as vernier. The vernier will increase the overall measuring resolution by a factor of two, one more bit. It is worthy to note that more heads may be added to the vernier arrangement leading to a corresponding increase in the overall measuring resolution, without any change in the patterns physically marked on the pseudorandom code track. A second advantage of this two head approach is that a simple control logic can be used to determine the AGV's moving direction. The essential principle behind this directional resolving is the leading or lagging of the AUT reading head signal with respect to the VER reading head signal, i.e. if AUT is leading VER by 90° then the AGV is moving to the left.

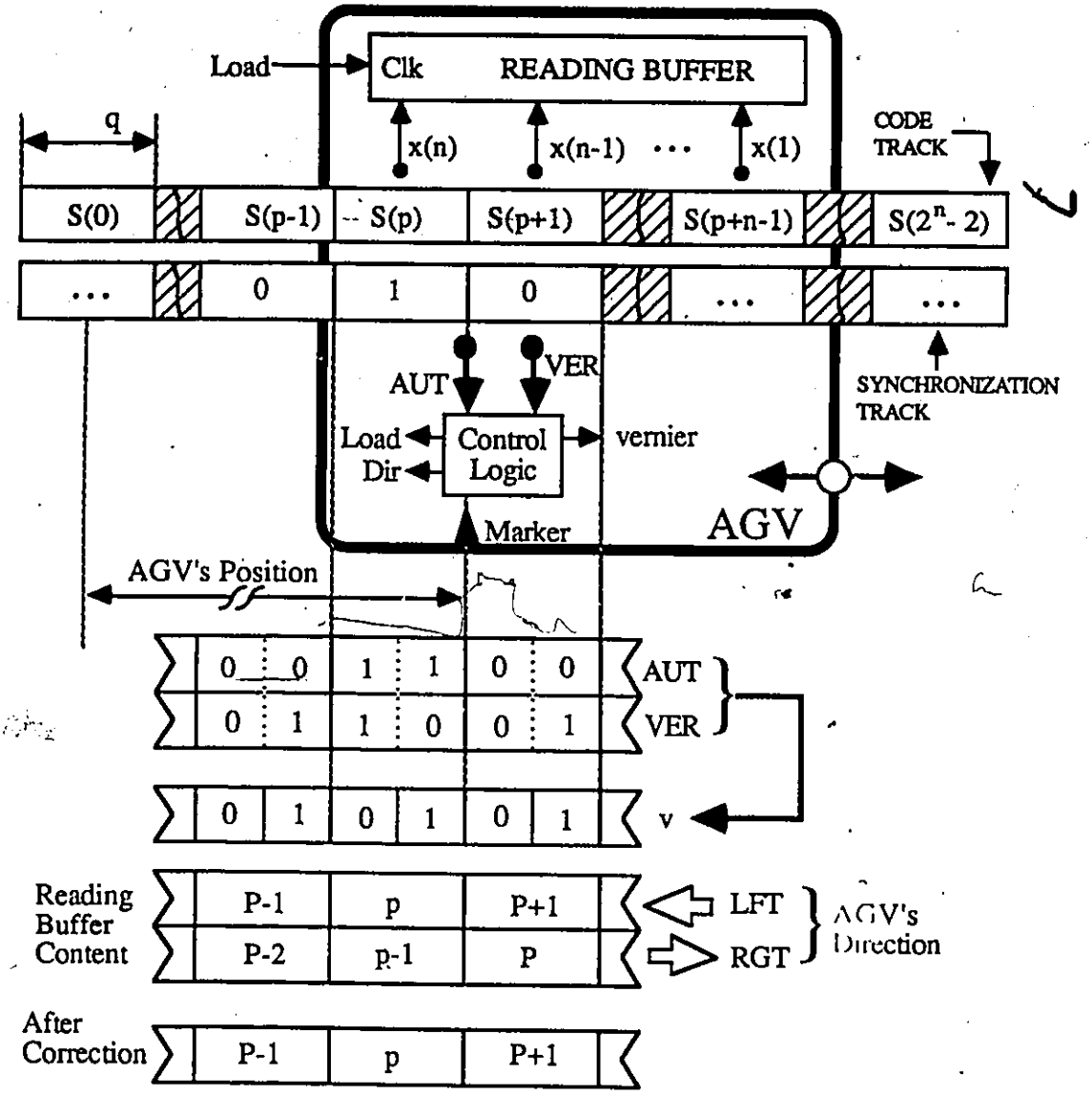


Figure 4.5 - Improved Code Scanning Approach

4.2.3 Virtual Synchronization Track

Although the two aforementioned code scanning techniques do work, they have one major drawback associated with them, the cost for the physical implementation of the additional synchronization track. In this section, a novel on-board synchronization method [46] that leads to the elimination of the off-board synchronization track will be discussed. This results in a minimum

complexity for the track configuration required to be laid on the floor - a guidepath (mandatory for all AGVS) and the one-bit wide pseudorandom code track.

The functional principle of this solution is illustrated in Figure 4.6. The technique for scanning the code track employs an incremental encoded free-running wheel for the on-board generation of the code reading synchronization signals. The free running wheel has an encoding resolution y appropriately selected as a submultiple of the encoding resolution q of the pseudorandom code track: $y = q / c$. As the AGV moves on its guidepath, the incremental encoded wheel will turn producing an output signal SYNC that may be perceived for convenience as a virtual synchronization track alongside the other two physical tracks laid on the floor: the guidepath and the PRBS code track.

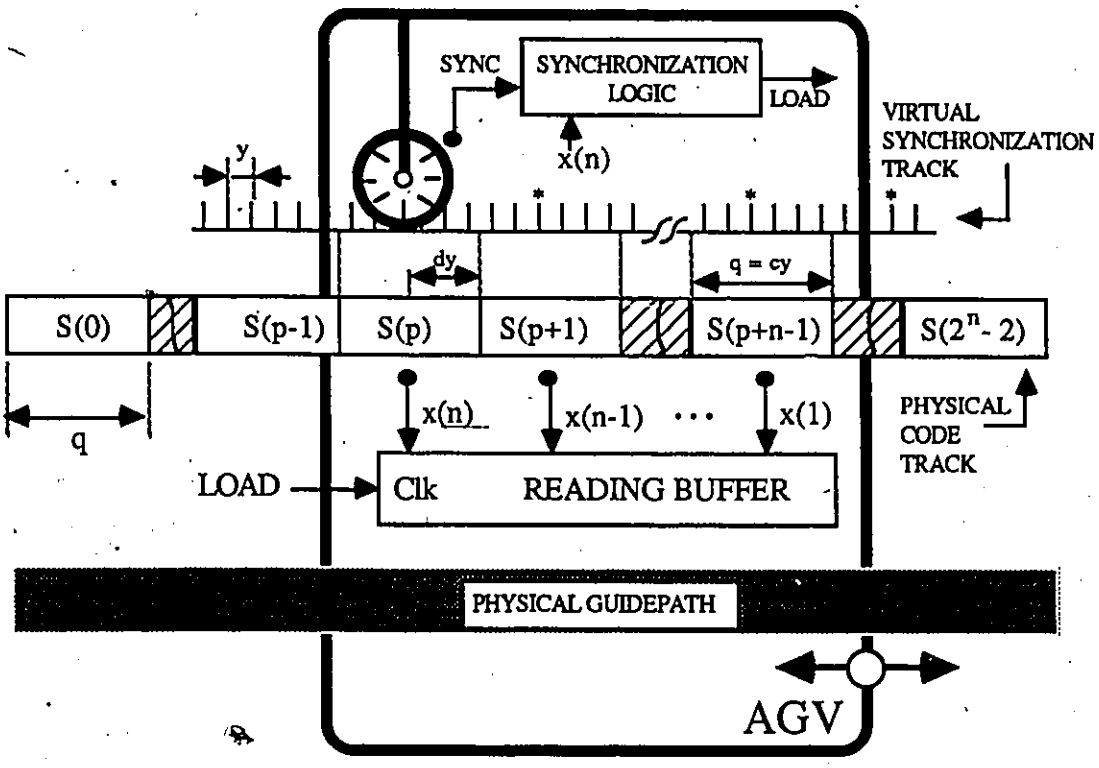


Figure 4.6 - Virtual Synchronization Track Approach

A SYNCHRONIZATION LOGIC circuit will select the specific SYNC pulse that occurs when the AGV code reading heads arrive in the middle of the quantization step. This specific pulse (marked by stars in Figure 4.6) propagates as the LOAD signal used to update the READING BUFFER containing the pseudorandom positional information. To avoid the cumulative errors inherent to the incremental encoding of the free wheel, an external initialization mechanism was introduced. This resetting of the SYNCHRONIZATION LOGIC is performed whenever the reading head x(n) of the code track detects a quantization step transition. Moreover, this resynchronization which is controlled by the physical code track may not occur at every quantization step but it will definitely occur after n consecutive steps (n is the AGV position measurement resolution). In this way the slippage error that may occur with the incremental encoded free wheel are not accumulated and their effect is limited.

Being directly driven by the AGV positional parameters, the described synchronization method is time invariant; hence, it is also speed and acceleration insensitive. As a drawback the on-board synchronization may introduce some limited slippage error. The actual value of this error may be reduced by increasing the c ratio between the code track resolution and the virtual synchronization track resolution. Finally, the previously discussed orientation of reading heads method along with a priori knowledge can readily be used to determine the direction of travel of the AGV.

4.3 Code Conversion

A translation from the pseudorandom binary code into the more convenient natural binary representation is always necessary for practical AGV applications. This code conversion process, which is the second stage of absolute position measurement, must be carefully considered to make PRBS encoding of the AGV's guidepath, a practical alternative. This perusal is necessary because this conversion process can be expensive with respect to time and/or equipment. In particular, the duration of the code conversion is critical for the absolute position measurement cycle and hence

for the real-time performance of each AGV, such as the maximum AGV speed on the guidepath. In order to attempt to reach an optimal compromise, a complete analysis of the code conversion problem will be detailed.

4.3.1 Approaches to the Code Conversion Process

The code conversion methods that can be employed vary in the extent that the translation is a serial and/or parallel process. A **strictly parallel translation** from the pseudorandom binary code into the more convenient natural binary representation can be done using a code conversion table stored in ROM as shown in Figure 4.7. This solution is equipment expensive for long PRBSs.

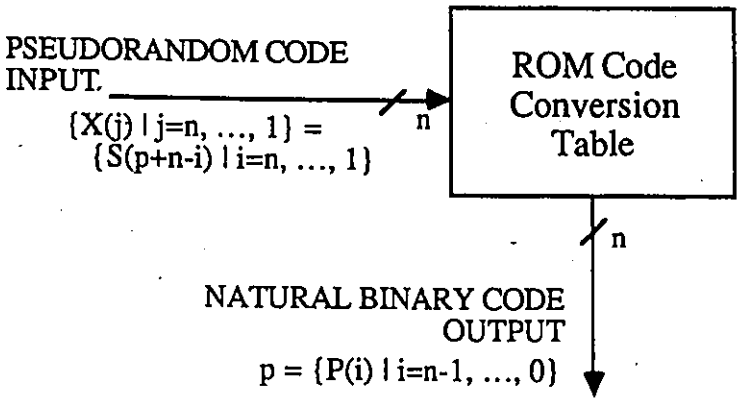


Figure 4.7 - ROM Lookup Table

A **strictly serial translation** exploits the reversibility of the PRBS generating algorithm [41]. The reverse feedback shift equations are given in Table 1. This method is based on the idea that it is possible to find the natural value of the AGV's position by simply counting the number of reverse feedback shifts that it takes for the given pseudorandom code to arrive back into the initial code for the "zero position". This code conversion algorithm is given in Figure 4.8. In this case the solution is less equipment expensive but more time expensive for long PRBSs.

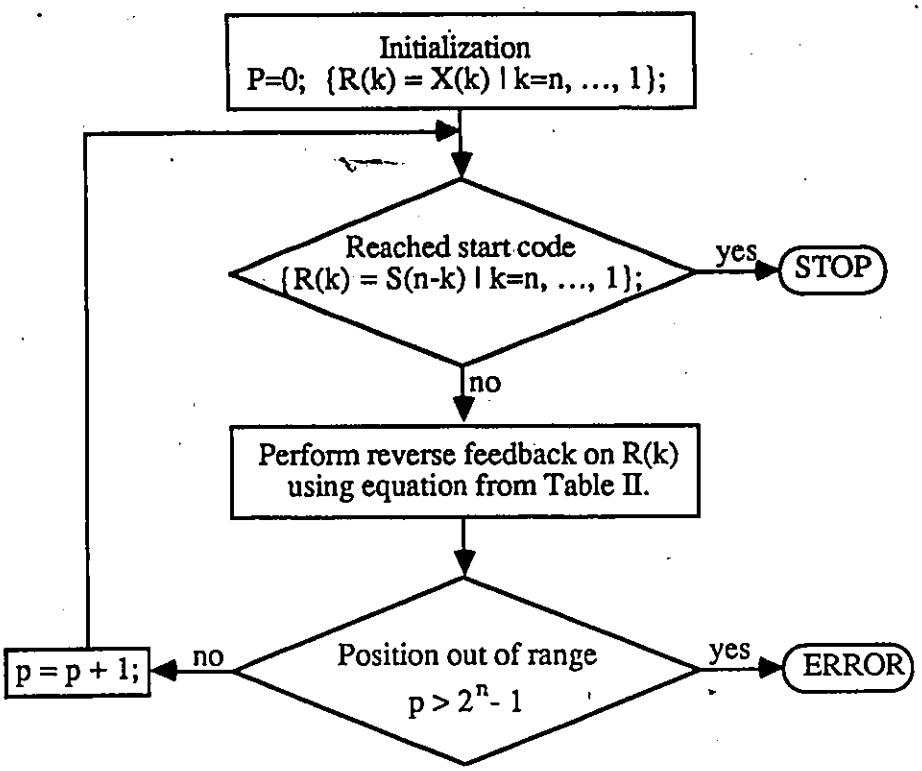


Figure 4.8 - Flowchart for the Serial Approach

A better solution to convert pseudorandom to natural binary is a **combination of the serial and parallel methods** and is exemplified in Figure 4.9 [43]. Consider a pseudorandom encoded track where certain positions (uniformly distributed with a period of t) are employed as milestones. The code conversion for a general position, $p=m \cdot t + r$, where $m \cdot t$ is the position of the nearest down the track milestone Q(m) and r represents the position relative to this milestone, will be discussed. The natural code for r is found by counting the steps required to arrive by successive backshifts from the initial code to the nearest milestone Q(m). All intermediate states of this serial shift-back operation are checked in parallel against all possible milestone pseudorandom patterns. Thus with this method the code conversion of the relative position r distance is found serially while the milestone code conversion is done in parallel. This approach to translation has equipment and time costs associated with it.

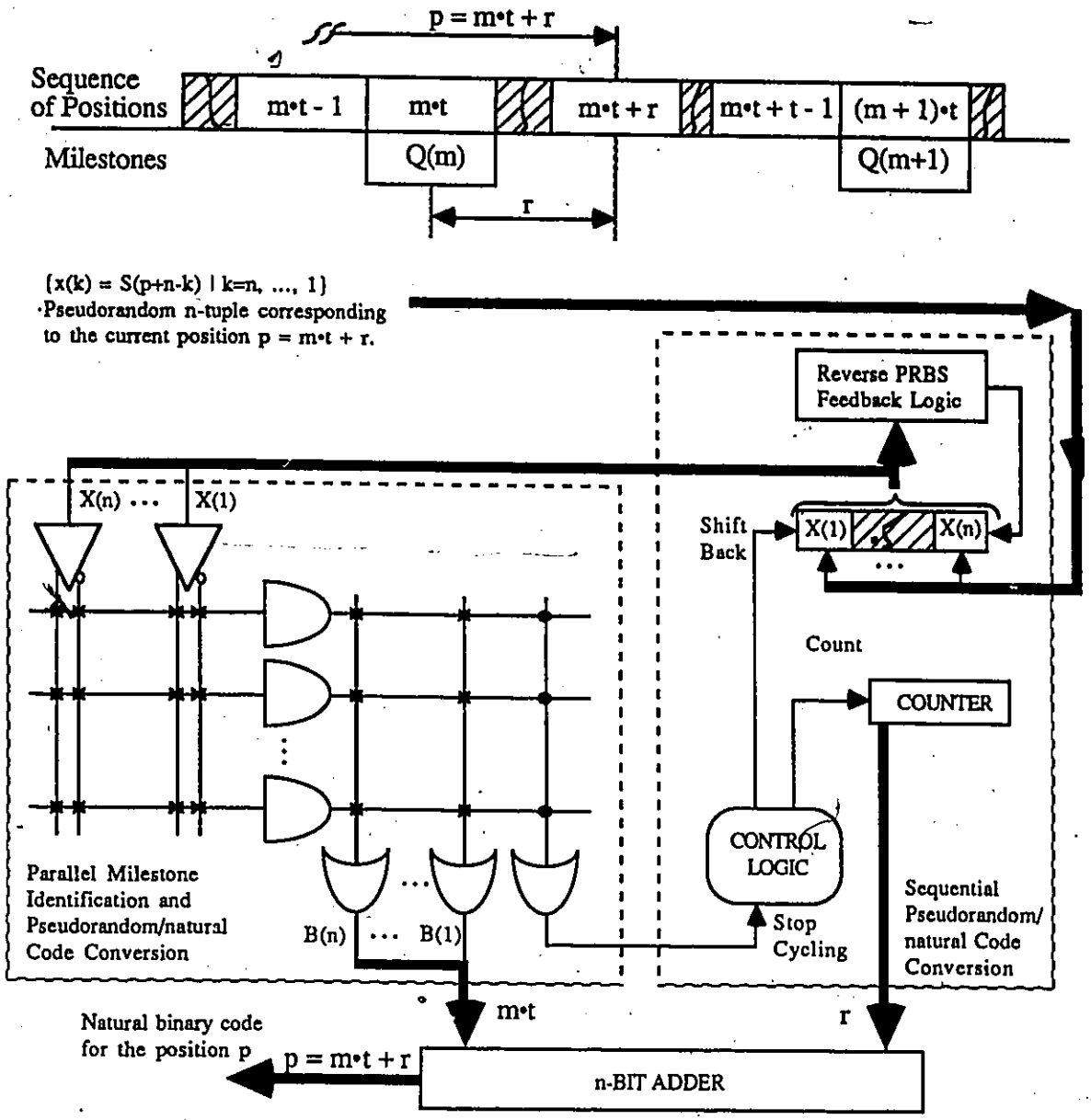


Figure 4.9 - Serial/Parallel Code Conversion

To help to clarify the different code conversion algorithms, an example with the comprehensive third algorithm will be given. Figure 4.10 illustrates the third code conversion method for a 5-bit absolute position measurement on a pseudorandom encoded track with 31 positions. This system uses four milestones which are paced with a period of t=8: Q(0), Q(1), Q(2), and Q(3), corresponding to the positions 0, 8, 16, and 24. The milestone identification

method has to implement the following four pseudorandom/natural binary code conversion rules: [00001/00000], [11101/01000], [01111/10000], [11010/11000].

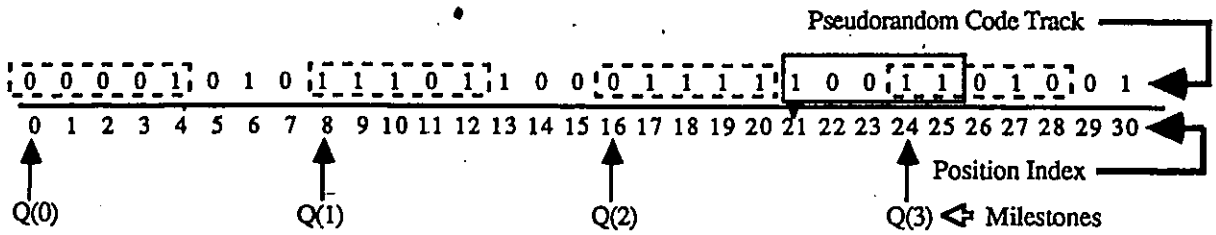


Figure 4.10 - Five Bit Absolute Position Measurement with Four Milestones

In this example, the current absolute position is p=21, for which the scanned pseudorandom 5-tuple is [10011]. According to the code conversion method, this 5-tuple is sequentially back-shifted, passing through the intermediate states 11001, 11100, 11110, 11111, until it reaches the 01111 state. This state is recognized by the parallel milestone identification mechanism which will deliver the corresponding natural binary code $m \cdot t = 10000$. When the milestone recognition occurs, the sequential code conversion procedure is stopped. Because it took five back-shifts to reach the milestone state, the final binary content of the counter is r=001101. The resulting binary code for the measured position is then $p = m \cdot t + r = 10000 + 00101 = 10101$, which correctly represents the measured position p=21. A detailed cost analysis of these code conversion approaches follows.

4.3.2 Cost Analysis of the Different Approaches

The equipment cost and the temporal cost are the two costs that constitute the total cost associated with the code conversion problem. Given below are the definitions of the variables and constants required for the cost analysis:

- n = the bit length of each pseudorandom binary code
- $2^n - 1$ = the total length of the PRBS
- t = the uniformly distributed period between successive milestones

k_1 = the equipment cost associated with each milestone
k_2 = the basal equipment cost for the serial backshift operations
k_3 = the basal temporal cost for a fully parallel solution
k_4 = the temporal cost associated with each backshift operation

Note: n is assumed constant for a given optimization

k_1, k_2, k_3, k_4 are constant for a given n and are functions of the employed technology

The equipment and temporal cost equations will be developed with these definitions as a function of the encoded distance t between successive milestones. The equations for these costs are as shown below:

equipment cost = k_1 • number of milestones + k_2

(k_2= 0 for the fully parallel solution)

equipment cost = k_1 • int $(2^n - 1 / t)$ + k_2

temporal cost = k_3 + k_4 • (t-1)

Figure 4.11 shows the temporal cost variation for the different approaches to the code conversion problem.

Now the total cost for the code conversion approach is as follows:

total cost = equipment cost + temporal cost

total cost = k_1 • int $(2^n - 1 / t)$ + k_2 + k_3 + k_4 • (t-1)

Figure 4.12 shows a graph of the equipment cost, temporal cost, and total cost as a function of the variable t, the periodic encoded distance between successive milestones. The temporal cost associated with the serial backshift matching operation linearly increases with the independent variable t while the equipment cost is inversely proportional with the independent variable t. The

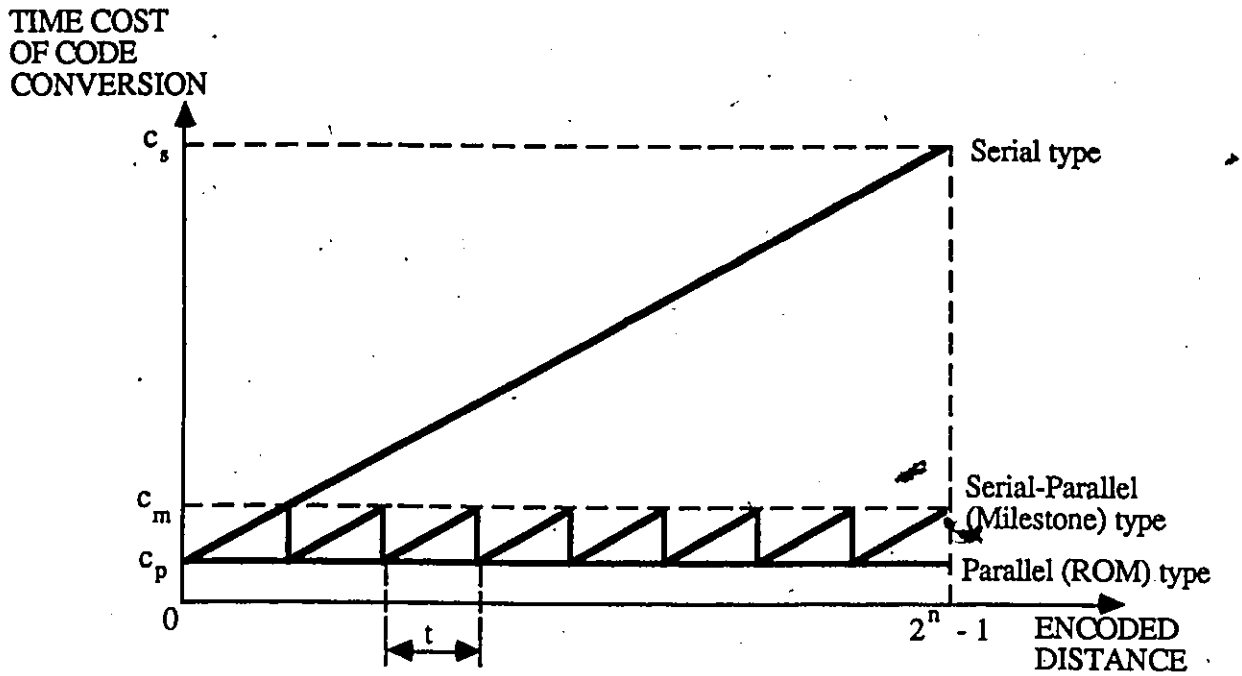


Figure 4.11 - Summary of Code Conversion Time Costs

combined total cost graph does have a minimum between 1 and 2^n - 1 for this analysis. The value of t which minimizes the total cost function is the following:

$$t_{opt} = \text{int}\left((k_1 / k_4 \cdot 2^n - 1)^{1/2}\right)$$

Depending on the relative ratios of the constants k_1 and k_4 (for a given n) the optimal distance between milestones will vary. This cost analysis provides a guide for choosing the number of milestones (i.e. the degree of parallelism) for a specific application. A number of examples of this optimization procedure are summarized in Table II. As one can expect a higher degree of parallelism is requested when the temporal cost is relatively higher than the equipment cost. A higher measuring resolution will always lead to an increase of the number of milestones in order to maintain the same code conversion speed.

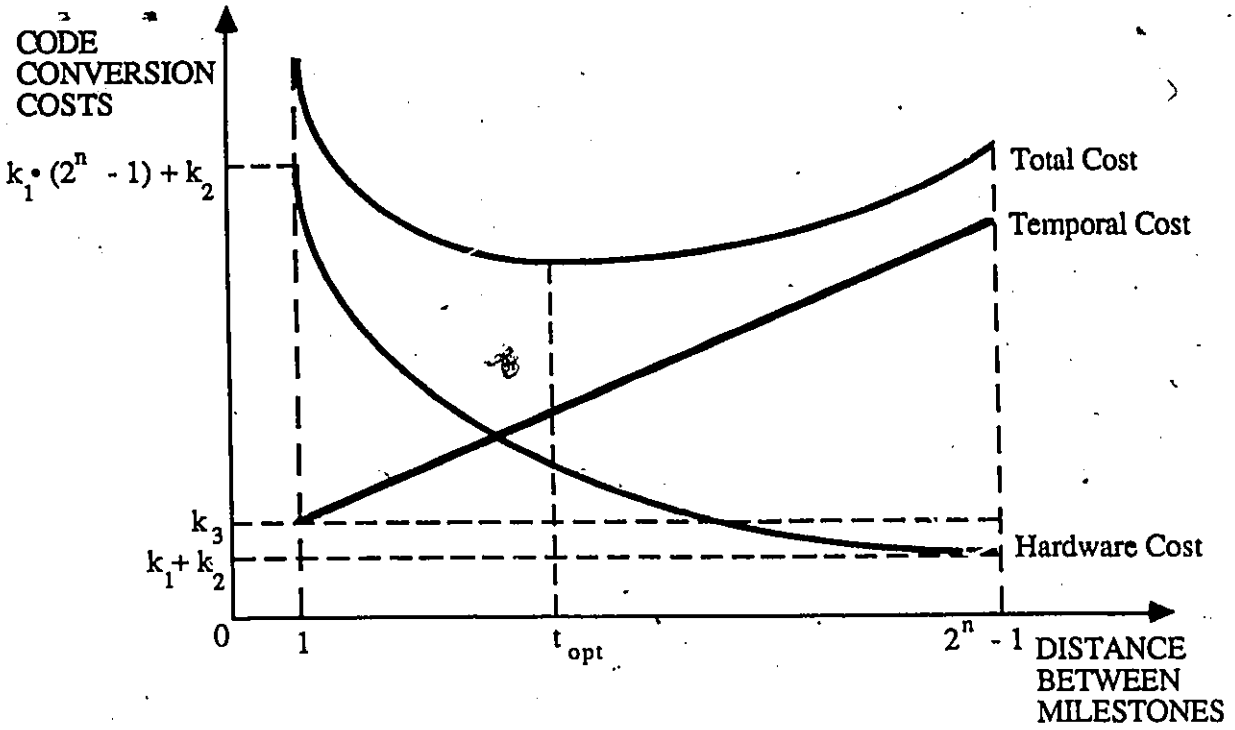


Figure 4.12 - Different Costs as a Function of the Distance between Milestones

Table III - Examples of Different Minimized Costs

REQUESTED NUMBER OF MILESTONES $(2^n - 1)/t$		Measurement Resolution		
		8 bits	16 bits	24 bits
Relative "equipment/ temporal" cost k_1/k_4	1/4	*32*	*512*	*8192*
	4/1	*8*	*128*	*2048*

4.4 Summary

Of the different discussed techniques, two of them, in particular, enhance the applicability of pseudorandom absolute position measurement as a cost-effective upgrading feature for the industrial AGVs. The on-board synchronization technique for code track scanning leads to a minimum complexity for the tracks required to be physically laid on the floor. The serial/parallel implementation of the pseudorandom/natural code conversion allows the best design tradeoff between the equipment and time costs to be made. The implementation aspects of these two methods will be discussed in the next chapter.

CHAPTER 5 - Implementation and Testing Aspects

.5.1 Preliminary Remarks

An experimental AGV, illustrated in Figure 5.1, was built to test two relevant techniques supporting the AGV absolute position measurement, the on-board synchronization and the serial/parallel code conversion. Also note that the track configuration on the floor consists only of a guidepath (mandatory for all AGVs) and a one bit wide pseudorandom code track.

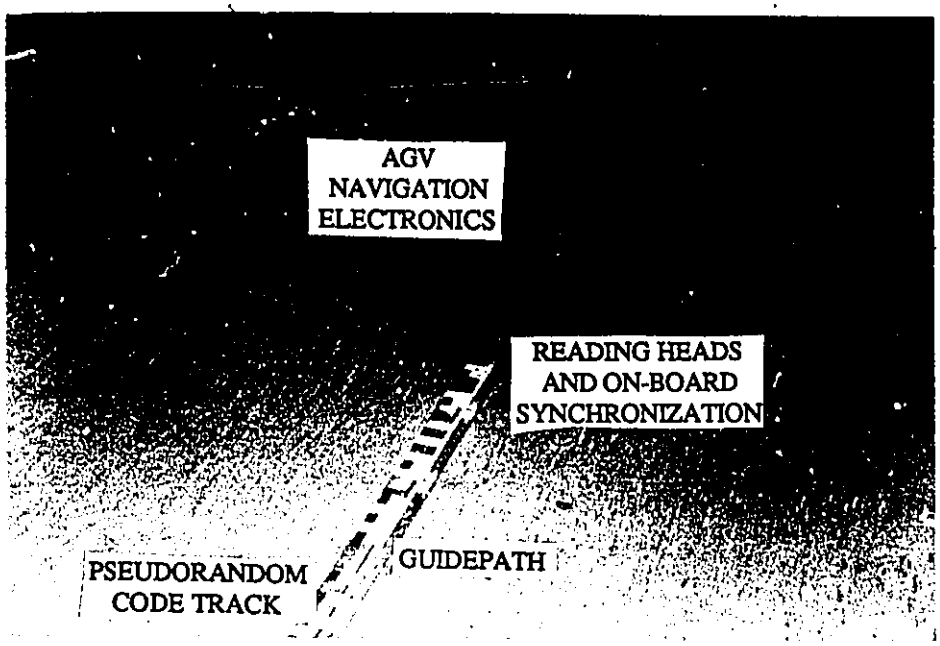


Figure 5.1 - Experimental AGV

The functional diagram of this AGV testbed is depicted in Figure 5.2. The AGV system is made up of two components, guidepath tracking and absolute position measurement, which have been implemented as independent functions. The information from the guidepath tracking heads is used by the guidepath tracking function to drive the AGV, with its two motors, in such a way as to maintain the moving AGV on the guidepath. This path tracking function was developed as a

machine language program running on an 8-bit microprocessor. A more thorough discussion of this function is beyond the aim of this thesis.

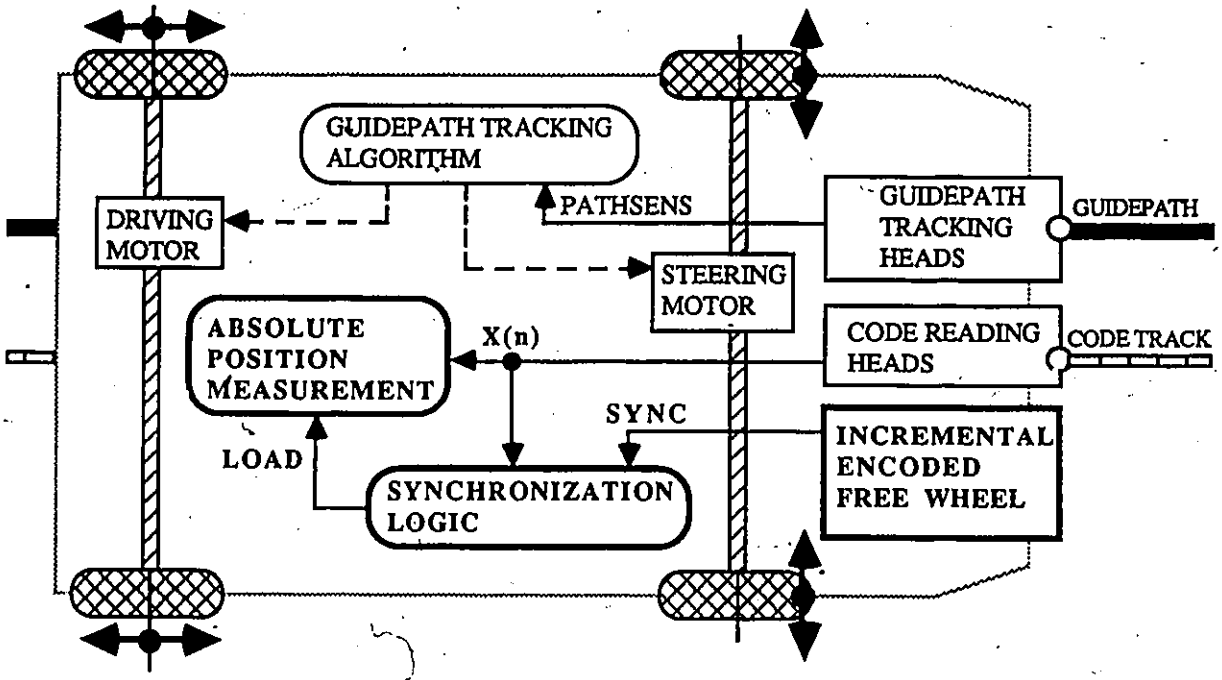


Figure 5.2 - Block Diagram of the Experimental AGV

Displayed in bold in the diagram are the three functions to be discussed in detail in this chapter. The **incremental encoded free wheel**, which is the basis of the on-board synchronization technique, produces the SYNC pulses for the synchronization logic. The **synchronization logic**, which by using the SYNC pulses and the code track transition changes, produces the LOAD pulses for the absolute position measurement component. Finally, the **absolute position measurement function**, which given the LOAD signal and the code track information, handles the following two steps:

- scan the code track in order to obtain the PRBS window corresponding to the current AGV's position.
- perform the pseudorandom/natural code conversion to deliver the measurement results in a format more appropriate for further use.

The details of implementation along with the testing of the experimental AGV system will now be presented.

5.2 Incremental Encoded Free Wheel

The technique for scanning the code track employs an incremental encoded free wheel, shown in Figure 5.3, to produce the SYNC signal. By being directly attached to the AGV, the number of synchronization signals generated will be directly proportional to distance moved; thereby, speed and acceleration insensitive too.

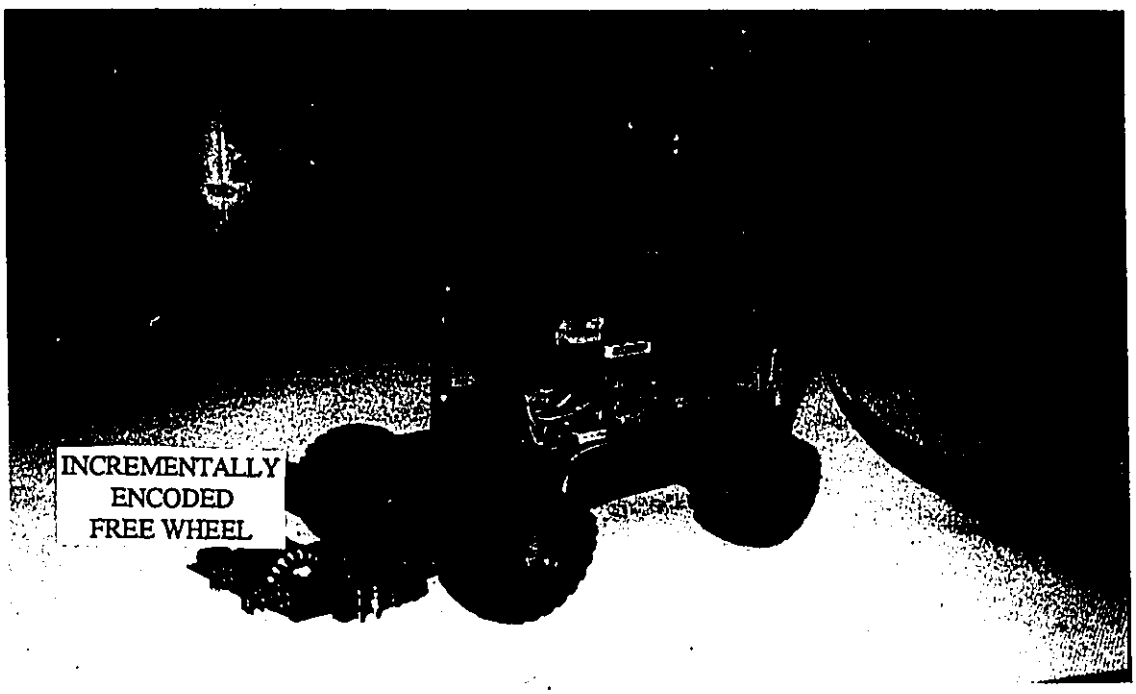
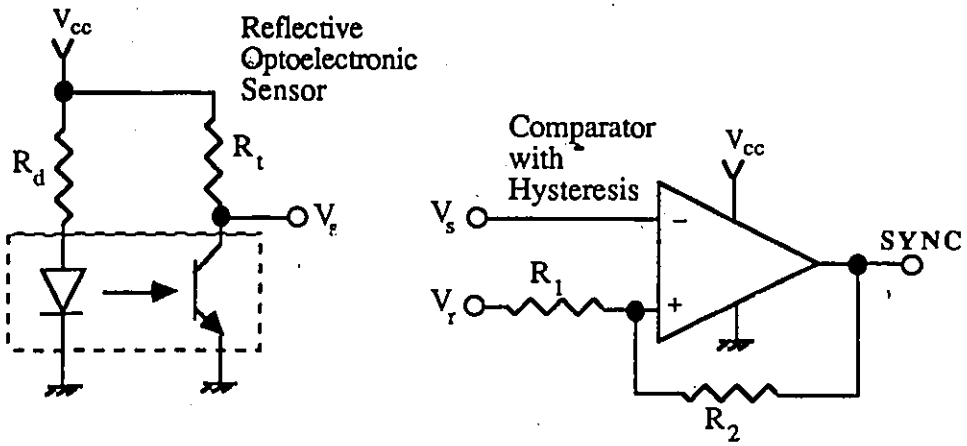


Figure 5.3 - Physical Implementation of the Free Wheel

Figure 5.4 shows the electronic circuitry used in combination with the free running wheel to produce the SYNC pulses. Located on the left is an optoelectronic module for reflective sensing applications. This assembly consists of an infrared emitting diode and an n-p-n silicon photo-transistor mounted in a plastic housing. The sensor output signal, V_s, is an analog signal

dependent on the value of the resistance R_d, the position of the sensor relative to the reflective surface, and the reflective surface itself. The reflective surface, the slots of the incremental encoded free wheel, were painted chrome silver to be the most effective in the application. The value of the resistance R_d and the position of the sensor relative to the reflective surface were determined by a heuristic process to provide the greatest dynamic range for V_s.



V_s - sensor output voltage
V_r - reference voltage
V_{th1} - threshold voltage 1
V_{th2} - threshold voltage 2

$$V_{th1} = V_r \cdot \frac{R_2}{R_1 + R_2}$$

$$V_{th2} = V_{th1} + V_{cc} \cdot \frac{R_1}{R_1 + R_2}$$

Figure 5.4 - Circuitry to produce SYNC pulses

The analog V_s signal is converted to a digital SYNC signal by connecting it to a comparator with hysteresis, shown on the right in the diagram. The operation of this portion of the circuit is as follows:

if SYNC=logical high voltage level, then V_s must rise above V_{th2} before SYNC will change to a logical low voltage level; moreover, once this change has happened the output SYNC level will not change unless V_s falls below V_{th1}.

else if SYNC=logical low voltage level, then V_s must fall below V_{th1} before SYNC will change to a logical high voltage level; moreover, once this change has happened the output SYNC level will not change until V_s rises above V_{th2}.

In other words in the range $V_{th1} \leq V_s \leq V_{th2}$, the comparator output voltage level will not change. This hysteresis ensures a clean SYNC signal without any oscillation problems.

5.3 Synchronization Logic

The purpose of the synchronization logic circuit is to select the specific SYNC pulse that occurs when the AGV code reading head(s) arrive in the middle of the quantization step and to propagate this specific pulse as the LOAD signal used to update the storage location containing the current PRBS window.

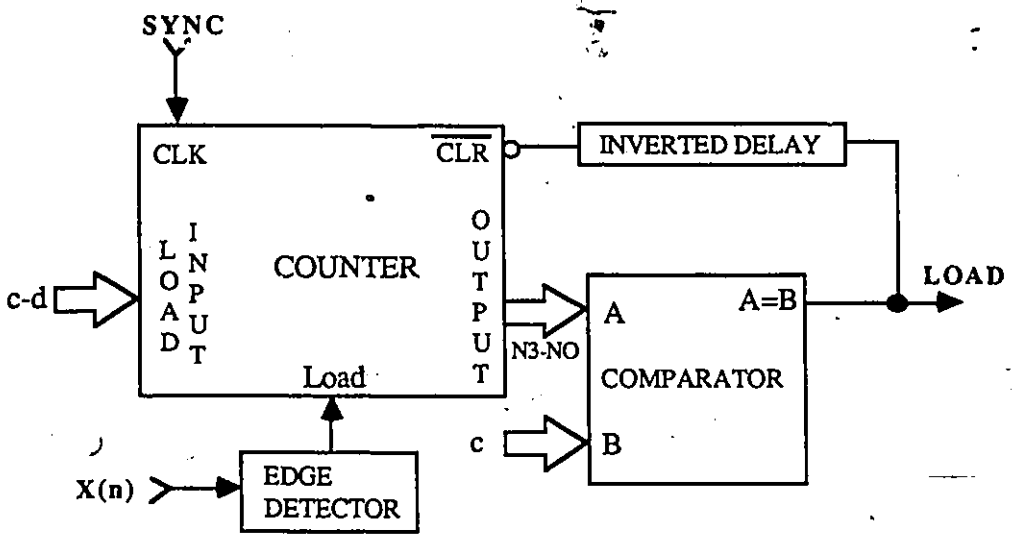


Figure 5.5 - Details of the Synchronization Logic

The synchronization logic, which is detailed in Figure 5.5, was implemented as a modulo c counter that generates a LOAD pulse after every c SYNC pulses. An external initialization mechanism, that uses code track transitions, had to be introduced to avoid the cumulative errors inherent to the incremental encoding of the free wheel. Specifically, every time a quantization step transition is detected by the code reading head, x(n), the counter is preset with the value c-d so that the LOAD pulse will occur after d SYNC pulses from every quantization step transition. Adopting a proper d=c/2 value will ensure that the information provided by the code reading head(s) is

always stable when the PRBS window is updated. This resetting of the modulo c counter, controlled by the physical code track, definitely occurs after n consecutive steps where n is the AGV position measurement resolution. In this way the slippage error that may occur with the incremental encoded free wheel are not accumulated and their effect is limited.

In the actual implementation, the synchronization logic circuit was built such that, $2 \leq c \leq 16$ and $1 \leq d \leq 8$. Due to the physical constraints of the incremental encoded free wheel and of the building of the PRBS code track, for overall system operation the particular values that were used were c=4 and d=2.

5.4 Testing of the Incremental Encoded Free Wheel and the Synchronization Logic

In this test setup (Figure 5.6) the incremental encoded free wheel was placed on a rotating platform to obtain the low frequency of system operation. The output signal to be measured from the circuitry was connected to a Hewlett•Packard 7045A X-Y Recorder.

The first signal to be scrutinized was the SYNC pulse which is illustrated in Figure 5.7. Although the pulses do vary in width, they are suitable for further use. The next signals to be examined were the binary outputs of the counter for different values of c. For example, shown in Figure 5.8, are the correct binary output signals for c=3, c=4, and c=5. The final testing of this portion of the circuitry was for proper resynchronization of the modulo c counter with the signal from the physical code track. This test was performed for the overall system values of c=4 and d=2.

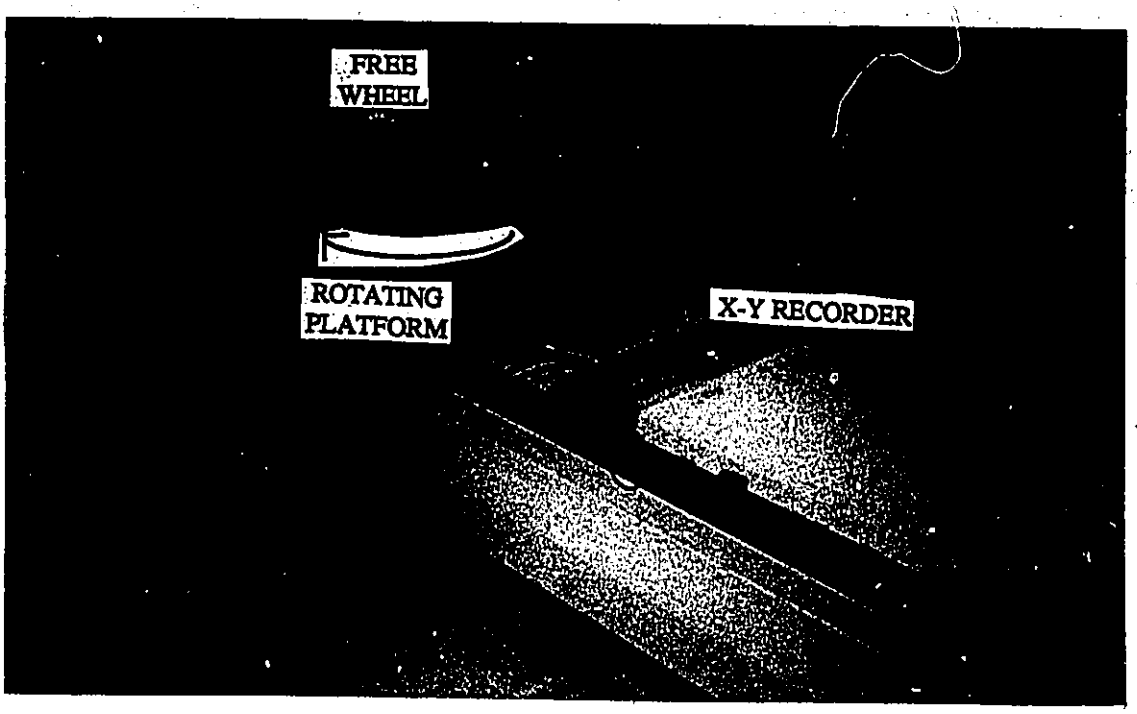


Figure 5.6 - Test Setup for First Stage

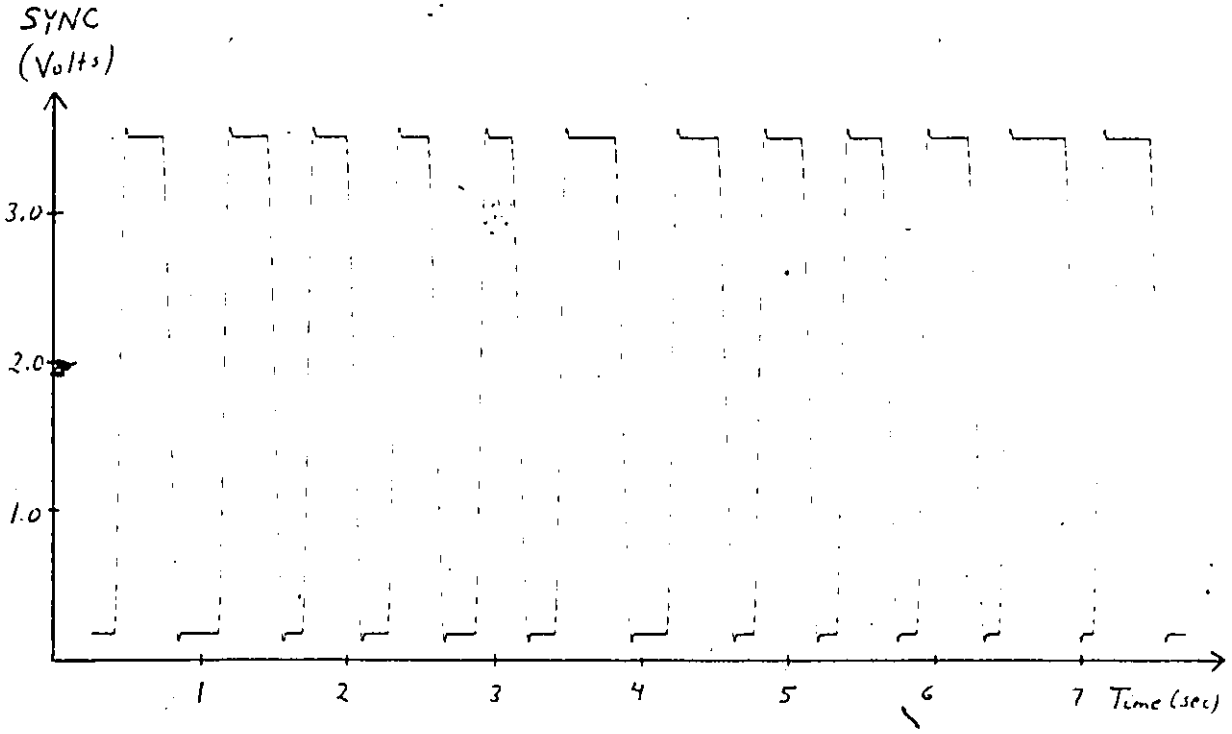


Figure 5.7 - SYNC pulses from X-Y Recorder

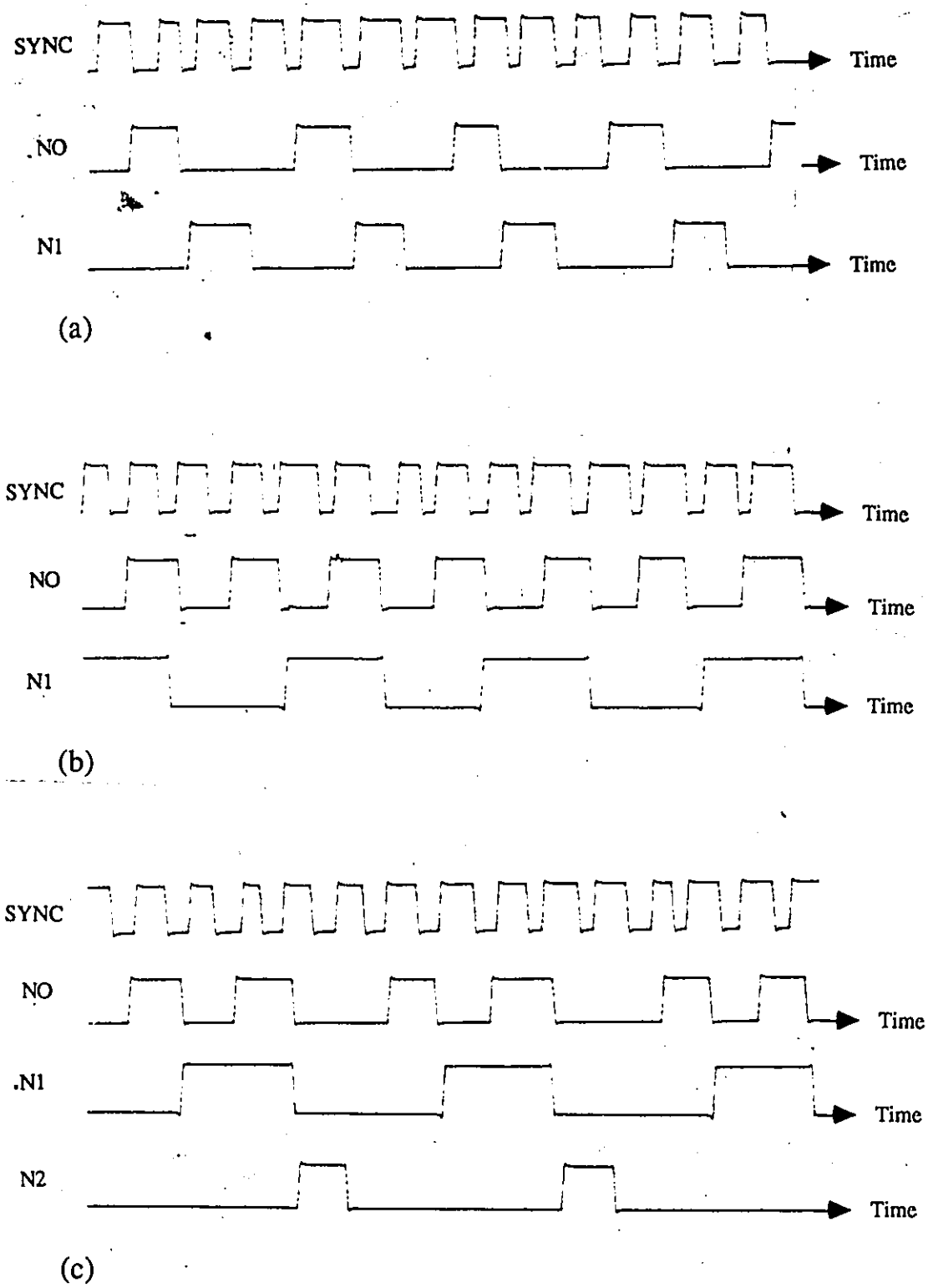


Figure 5.8 - Counter Outputs for Different c values (notation consistent with Figure 5.5)
(a) c=3, 5 V/cm, 0.5 sec/cm
(b) c=4, 5 V/cm, 0.5 sec/cm
(c) c=5, 5 V/cm, 0.5 sec/cm

It was found for code value transitions of either low to high or high to low that the binary output of the counter responded correctly (Figure 5.9). Having completed the discussion of and

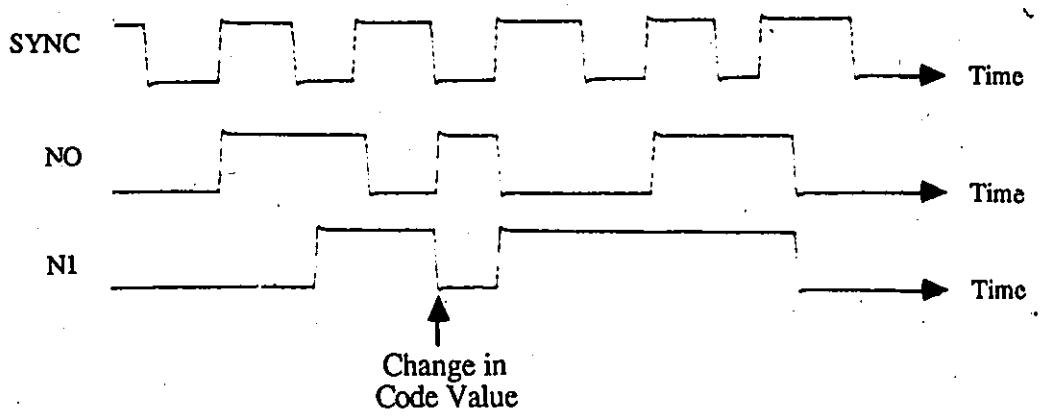


Figure 5.9 - Correct Resynchronization Response

the testing of this part of the system, the details of the absolute position measurement function will be presented.

5.5 Absolute Position Measurement Function

The discussion of this function is dependent on certain experimental characteristics. Firstly, in the test setup the absolute position was encoded with a resolution of n=8 bits and a quantization step of q=2.54 cm. Secondly, instead of using n=8 code reading heads, only one code reading head was used to investigate the code track. Two factors contributed to the decision of using only one reading head. One was the fact that fewer reading heads allow for a smaller turning radius for the AGV. Another factor was based on the fact that an electronic solution is more cost effective than a mechanical one requiring the building of precise mechanical elements.

Obtaining the Code

With these two experimental traits mentioned, the first stage of the absolute position measurement function, the obtaining of the current pseudorandom binary code (PRBC) will be presented in reference to Figure 5.10. With one reading head, the 8 bits of the position code are not read in parallel but assembled sequentially in an 8-bit shift register. The LOAD pulse from the synchronization logic clocks the shift register to read in the incoming bits of the PRBC. Because the shift register operates in only one direction while the AGV can travel in either direction on the guidepath, the contents of the shift register must be properly interpreted. This task is accomplished by feeding the AGV's direction of travel and the output of the shift register into multiplexers to give the correct code output of {PRBC(i) | i=8, ..., 1}.

In addition the clock for the multiplexers must only happen when the pseudorandom contents of the shift register definitely is a complete PRBS window. This problem arises because in the worst case it takes 16 steps (8 for the first code track resynchronization plus 8 more to obtain a complete code position) in the same direction, from an unknown starting point, to obtain a fully valid PRBS window. The validation circuitry depicted in the top of Figure 5.10 takes care of this minor drawback. The ENABLE signal is used as a control line for beginning the operation of the absolute position measurement function and/or as a reset of the 4-bit counter whenever the AGV changes its direction of motion. In either case with an ENABLE activation, the VALID output is switched to the "LO" logic value. After such a change occurs VALID remains "LO" until 16 consecutive AGV steps, in the same direction, produce the LOAD pulses that bring the counter contents to 16. At this point the counter freezes in this state and the VALID signal goes "HI", meaning that the position measurements are now considered correct. This VALID signal and a delayed LOAD pulse combine to produce the LOAD_VALID signal which is the appropriate clock for the multiplexers.

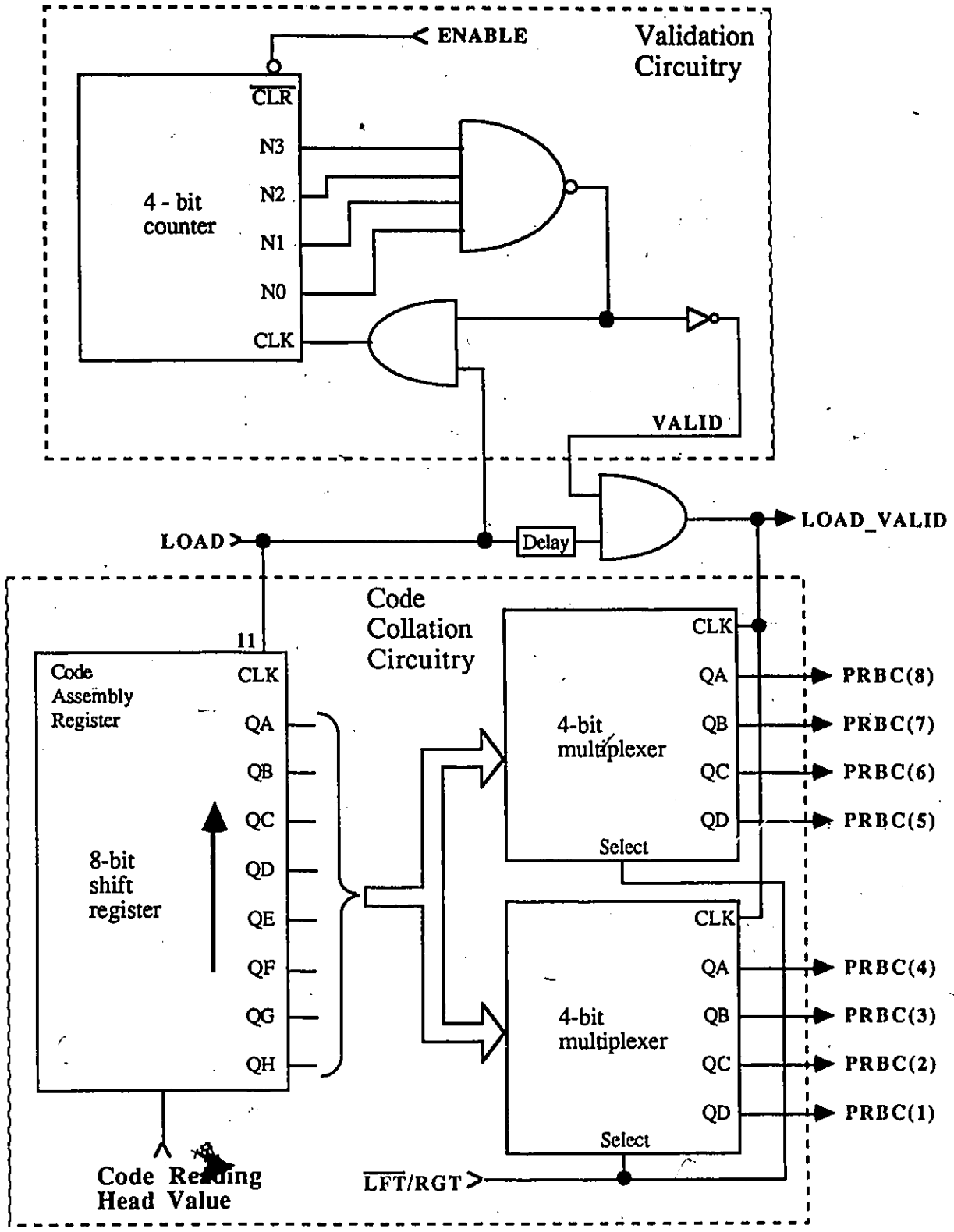


Figure 5.10 - Collecting the Correct PRBS Code

Converting the Code

Having obtained the 8-tuple {PRBC(i) | i=8, ..., 1}, the second stage of the absolute position measurement function is to convert it to the natural binary code (NBC) 8-tuple of {NBC(i) | i=8, ..., 1}. Appendix A contains the code conversions to be made for this experiment, while Appendix B contains a general purpose PRBS generation program written in the C language. The pseudorandom/natural code conversion that will be detailed in this section is the serial/parallel combinational approach.

The control logic and the reverse PRBS feedback logic are exemplified in Figure 5.11. Using LOAD_VALID and MS as input, the control logic generates the high frequency CLK, which drives the code conversion part of the measurement function, and the CLEAR_COUNT, which clears the sequential counter for each new PRBS window. For the reverse feedback logic, the LOAD_VALID in conjunction with the CLK compels the 8-bit shift register to load the 8-tuple {PRBC(i) | i=8, ..., 1} corresponding to the AGV's current position. After the shift register has been properly loaded the reverse feedback equation, shown in the diagram, is applied until the CLK signal stops.

Another significant part of the code conversion implementation is the parallel milestone identification and pseudorandom/natural code conversion, illustrated in Figure 5.12. As can be seen the outputs from the shift register performing the reverse feedback are appropriately connected to four 8-input nand gates. These nand gates perform the four milestone matches in parallel, as shown. If there is a milestone match then the output matching signal MS goes "LO" and stops the CLK generation to produce a stable output position. The code conversion is taken care of later by the two outputs, B8 and B7.

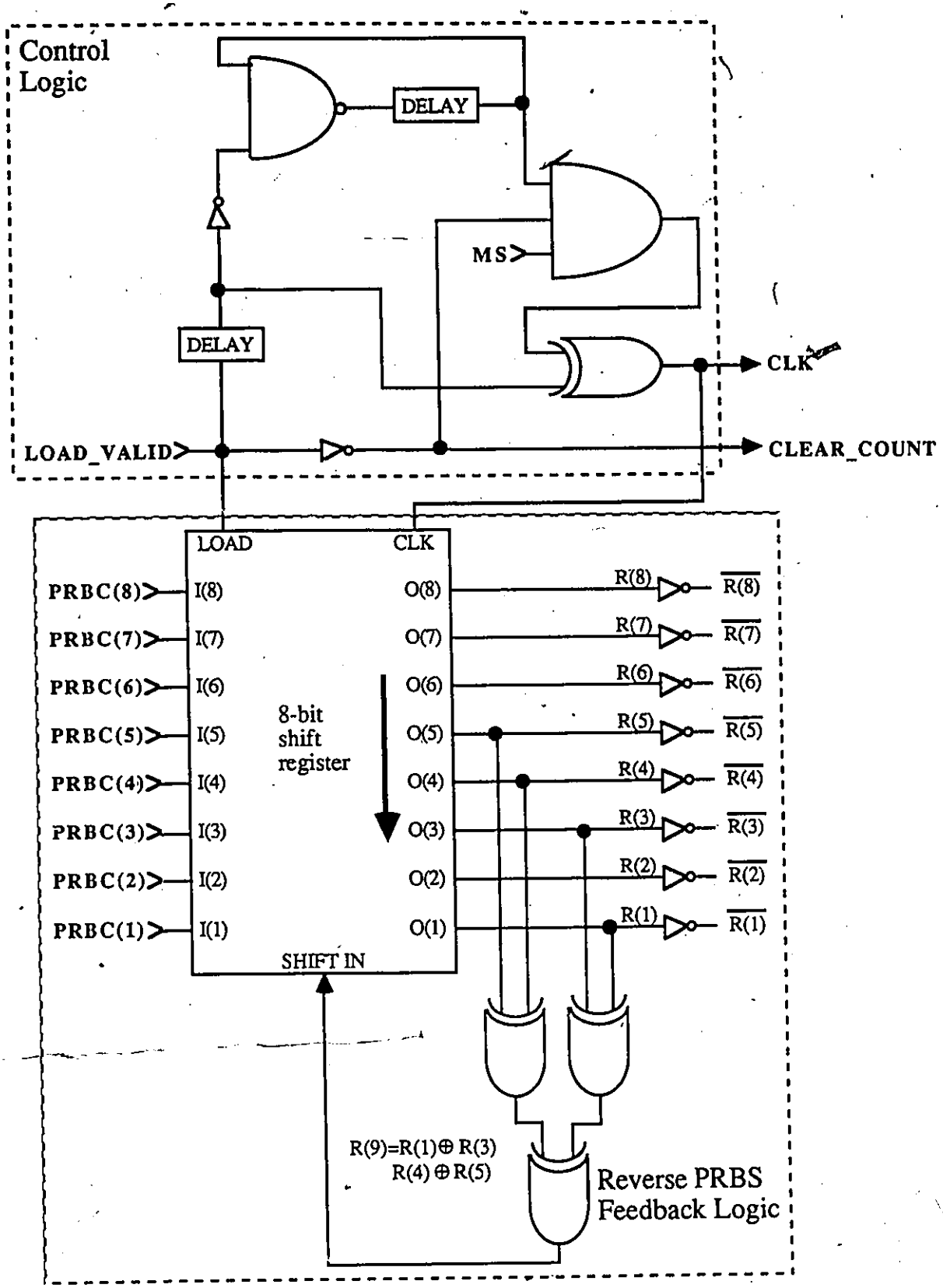


Figure 5.11 - Control and Reverse Feedback Logic

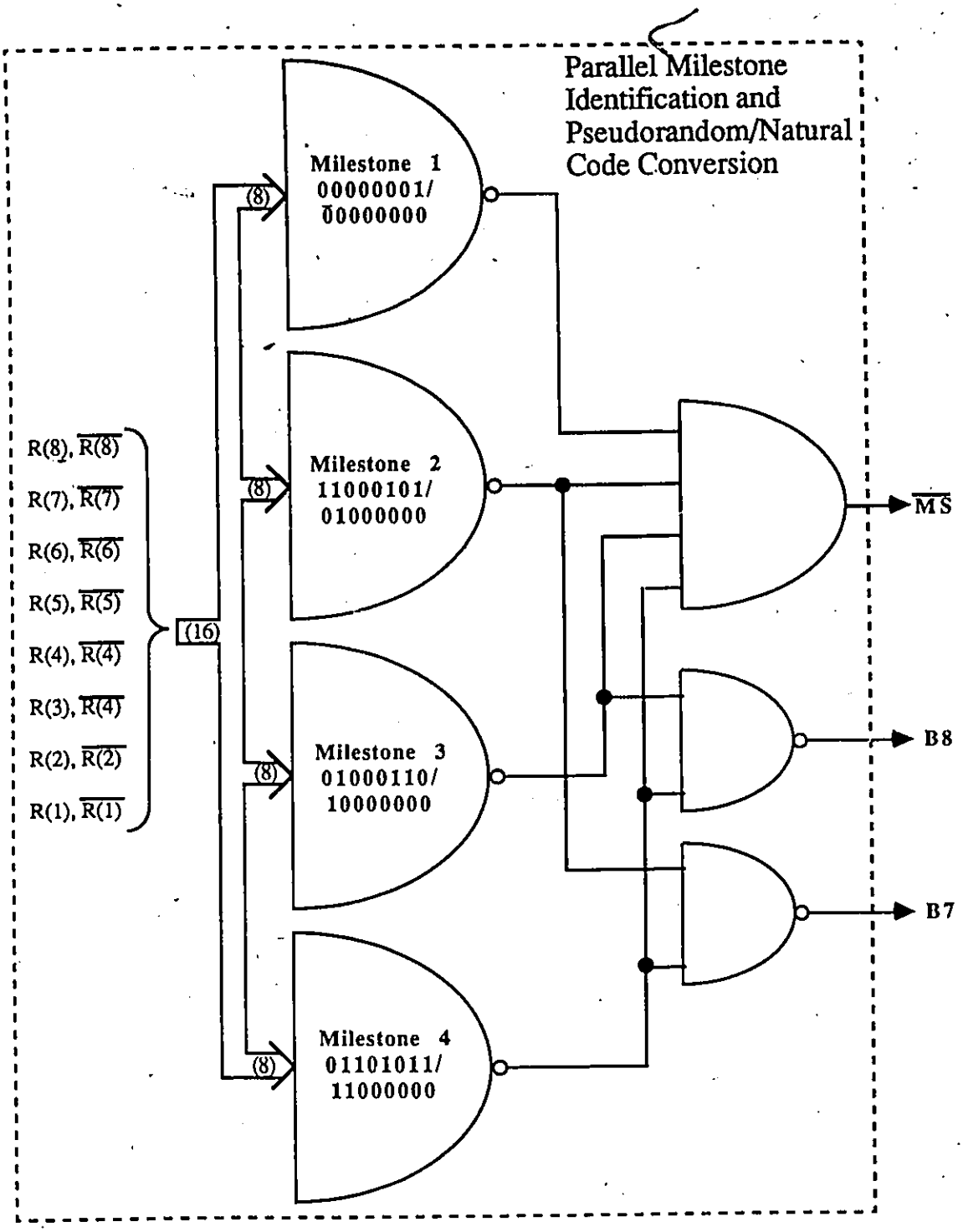


Figure 5.12 - The Parallel Milestone Identification

The last step of the absolute position measurement function must produce the overall absolute position with all minor corrections made (Figure 5.13). First of all the serial part of the

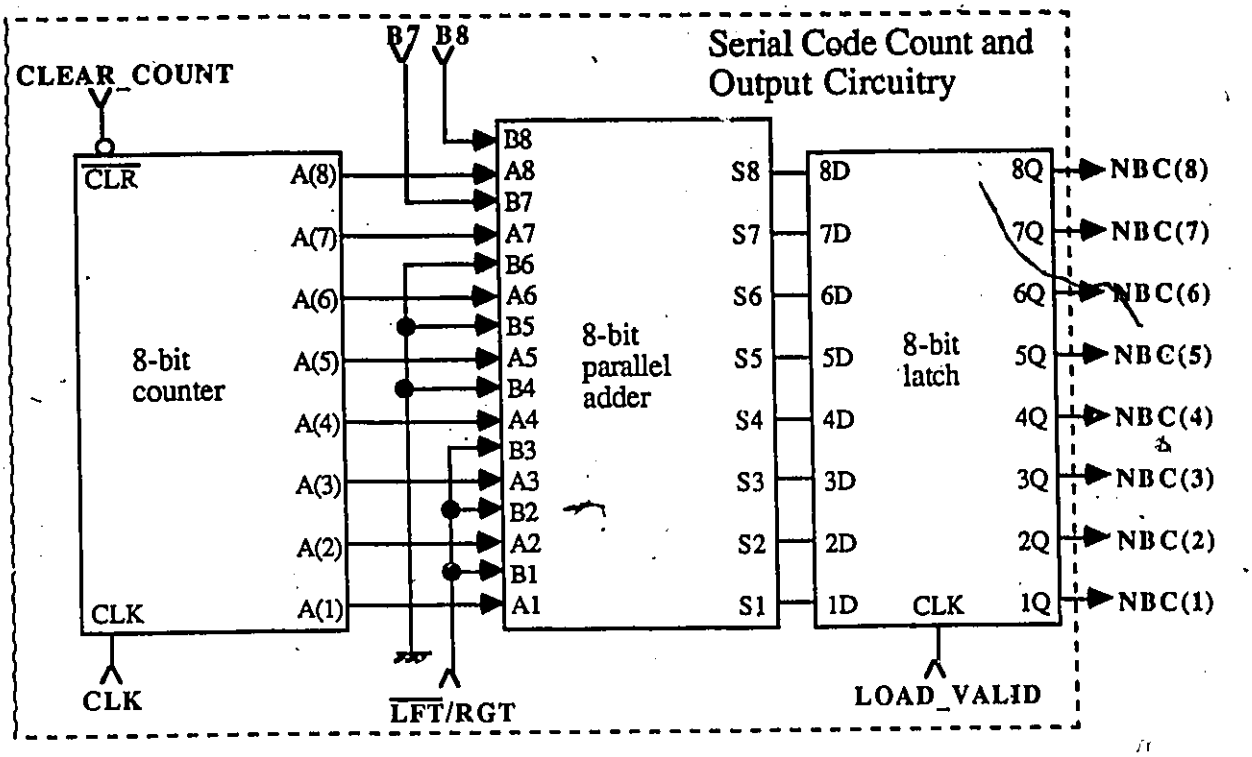


Figure 5.13 - The Natural Binary Output

code conversion algorithm is implemented by an 8-bit counter with CLK and CLEAR_COUNT as inputs. The corrections to the final position are made through the help of an 8-bit parallel adder as follows:

(a) outputs of the serial counter are fed into one 8-bit input of the adder.

(b) outputs, B8 and B7, are connected to the two most significant bits of the second input of the adder to amend for the particular milestone match.

(c) the direction of AGV travel, $\overline{\text{LFT}}$/RGT, is connected to the three least significant bits of the second input of the adder to compensate by +7 when moving right. this systematic error happens because with one reading head the 8-tuple window obtained when moving right is offset by the amount 7, while when moving left the window is properly obtained.

(d) all other second inputs of the adder are connected to ground.

Hence, the 8-bit parallel adder manages all the incidental amendments to obtain the correct 8-bit natural binary position at its output. The adder's output is latched when it is stable by an 8-bit latch to give the natural binary code corresponding to the current position, {NBC(i) | i=8, ..., 1}.

Please take note that in Appendix C are given the detailed circuit diagrams for all the different components of the hardware.

5.6 Testing and Systemic View of the AGV System

The complete operational test setup is given in Figure 5.14. The experimental AGV is tethered to the power supplies and the parallel port of the BCC52X BASIC Computer/Controller. This computer is further connected to a terminal for programming purposes.

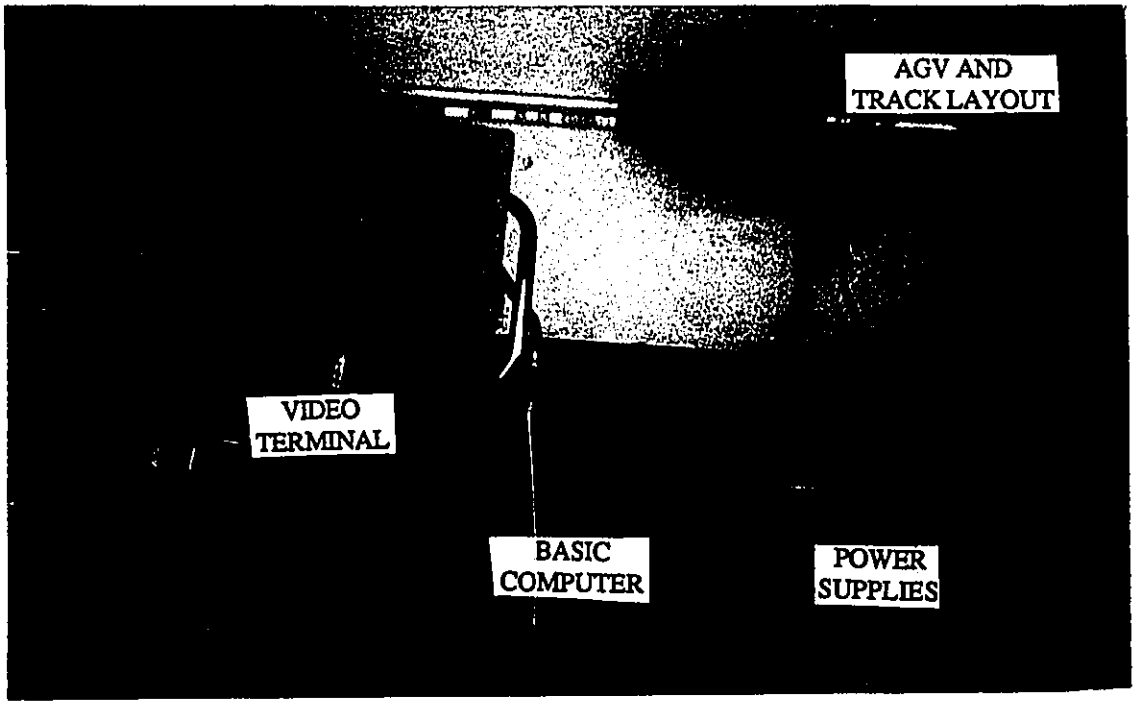


Figure 5.14 - Setup for Complete Testing

The specific functional diagram of this testbed is shown in Figure 5.15. For the absolute position measurement function, there are 2 control lines and 8 input lines. The 2 lines controlled through portA are the ENABLE, for resetting the measurement function, and the $\overline{\text{LFT}}$/RGT, for informing the position measurement board of the direction of travel. The 8 input lines from portC

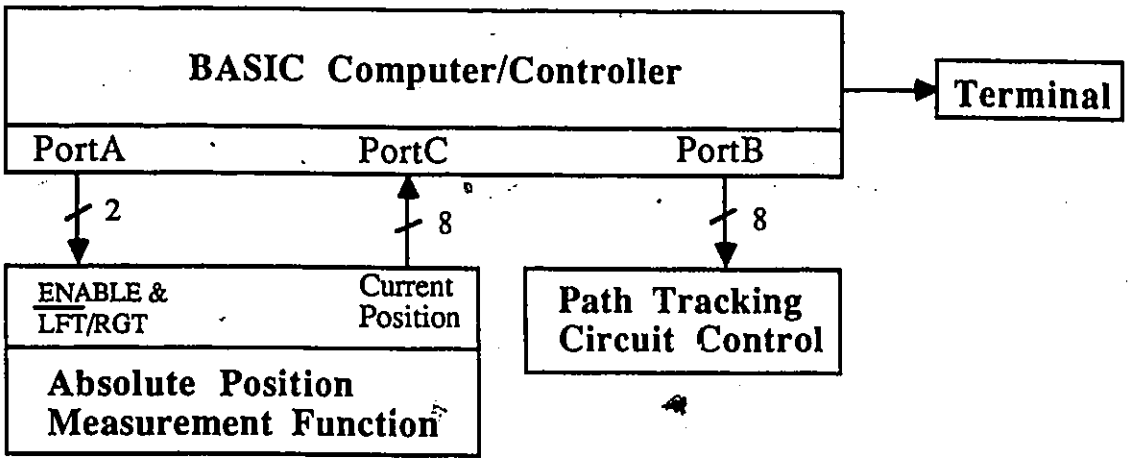


Figure 5.15 - Block Diagram Overview of the System

are the 8 bits of the natural binary position, {NBC(i) | i=8, ..., 1}. For the path tracking circuit control there are 8 lines controlled through portB. These 8 lines allow path tracking circuit control as follows:

bit 7: direction of travel
- 0 - forward
- 1 - backward

bits 5-6: speed control bits
- 00 - no speed
- 01→10→11 - increasing speed

bit 4: type of control
- 0 - automatic path tracking
- 1 - manual

bits 0-3: if under automatic control then
alignment of sensors relative to the track
else if under manual control then
control of angle of front wheels

Having fully described the systemic control of the experimental AGV system, a simple BASIC program was written to test the operation.

```
10    PORTA=0C800H:PORTB=0C801H:PORTC=0C802H:PORTCTROL=0C803H
20    PORTSET=89H:LEFTEN=16:RIGHTEN=90:HALT=8
30    AUTLEFT=37:AUTRIGHT=40
40    XBY(PORTCTROL)=PORTSET
```

```
50      INPUT "FINAL POSITION" POSIT
60      XBY(PORTB)=AUTRIGHT:XBY(PORTA)=RIGHTEN
70      DO:UNTIL XBY(PORTC)=POSIT
80      XBY(PORTB)=HALT
```

Lines 10 through 30 initialize the important variables. Line 40 initializes the parallel port of the BASIC computer such that portA and portB are output while portC is input. The stopping position of the AGV system is entered at line 50. The automatic path control of the AGV is started at the slowest speed and the setting of the two control lines of the absolute position measurement board are accomplished by line 60. Lines 70 waits until the AGV's position is equal to the entered final position and line 80 stops the AGV after the condition in line 70 is met.

The running of this program produced a favorable result. Illustrated in Figure 5.16 is the before and after running the above program, if the value of the position entered when prompted by the terminal corresponds to the given arrow.

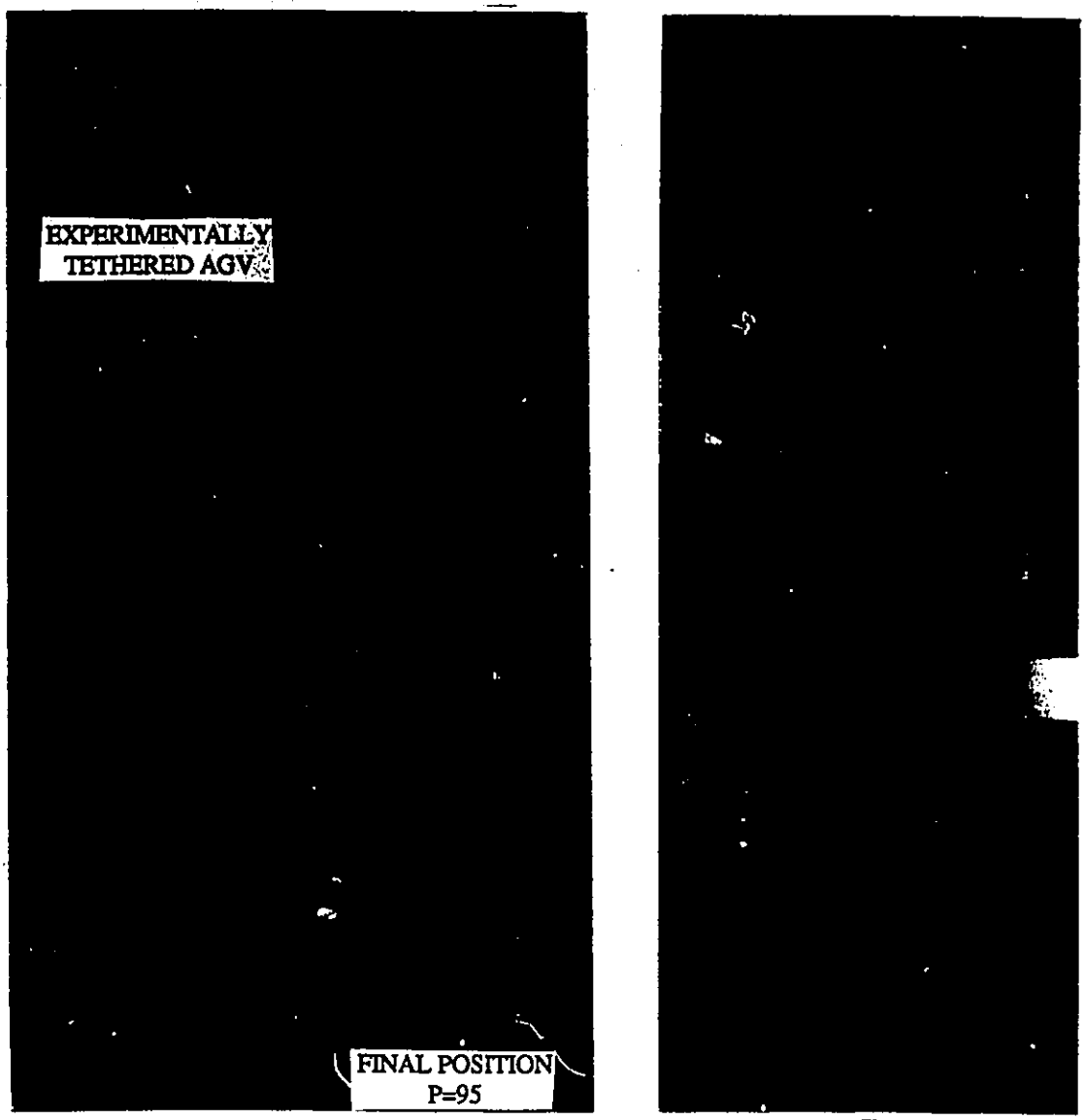


Figure 5.16 - A Test Run

5.7 Conclusion

The hardware components necessary to implement the two enhancements for the PRBS encoding of the absolute position of an AGV were built and tested. From the outcome, it has been shown that the proposed improvements are feasible.

CHAPTER 6 - Conclusion and Recommendations

6.1 Conclusion

The two techniques discussed and developed in this thesis contribute to the promotion of pseudorandom absolute position measurement as a cost-effective, stand-alone, upgrading feature, particularly suitable for the existing optically-guided AGVs which represent a significant portion of the current industrial AGV population. With this approach, all that is required external to the AGV within the manufacturing environment, is the simple addition of a pseudorandom binary sequence laid along the guidepath.

Moreover, the basis for future experimental investigation with respect to AGV navigation and control with this absolute position measurement capability has been developed and is presently available. The avoidance of obstacles on the guidepath, followed by recovery of the AGV's location can be studied easily with the ability to switch between manual and automatic control when necessary. Additionally, the 8052AH microcontroller and power supplies could be moved on board the AGV to remove the tethering and thereby facilitate the testing procedures.

6.2 Recommendations for Future Work

A foremost suggestion would be to extend the principles that were applied from pseudorandom sequences in the one dimensional case, to pseudorandom arrays in the two dimensional case. A pseudorandom array is an area of binary digits such that a window of size, say $n_1 \times n_2$, within that array would always be unique and hence correspond to a specific two dimensional position. This naturally leads to the solving of code reading and code conversion problems for a pseudorandom window in two dimensions. In general, the difficulties associated with two dimensional codes would be compounded, for example the orientation of the AGV relative to the encoding would be a problem of four directions, whereas previously it was only two.

Another area of exploration would be a visual approach to guidepath following and/or code information reading for either a one or two dimensional pseudorandom case. This approach can be viewed as an alternative or as an addition to the existing system. As an alternative, there would be obvious difficulty coping with the real-time image processing, but the AGV would possibly be less constrained by the guidepath and better able to cope with obstacles. In the combinational approach, the visual sense could provide a means for searching and guiding the AGV to the encoded regions but once there, the AGV would revert to using the easy to process magnetic or infrared sensors. Furthermore, this visual method could improve the reliability of the AGV system by checking and ensuring that the AGV is in a encoded track or area region, as opposed to the proximity sensors perceiving that the AGV is in an encoded region when it is not.

6.2.1 Investigation into a Specific Visual Problem

The one dimensional problem, of replacing optoelectronic sensing for the incrementally encoded free wheel and the circuitry for the synchronization logic with a visually based method written in the C language was explored.

In our laboratory, this sight technique was able to be investigated with the PIP512, a video digitizer board which plugs into an expansion slot of an IBM PC or compatible computer. The PIP512, which has a resolution of 512 x 512 pixels with eight bits per pixel, is capable of operating in continuous or single frame grab mode. Data transfer is possible from the IBM PC to the PIP board and from the PIP board to the IBM PC.

With accessibility to this board and the accompanying software routines, an algorithm testing the aforementioned notion is possible. What the algorithm must take into account is explicated in Figure 6.1. In case (A), the guidepath is to the left of the code track, while in case (B) the guidepath is to the right of the guidepath. In both of these cases, there are two subcases - top line has only guidepath present or top line has guidepath and code track present. With the

assumption that the ratio of guidepath width to PRBS quantization width is known, the program must determine which track is the PRBS and then properly scan the pseudorandom code track.

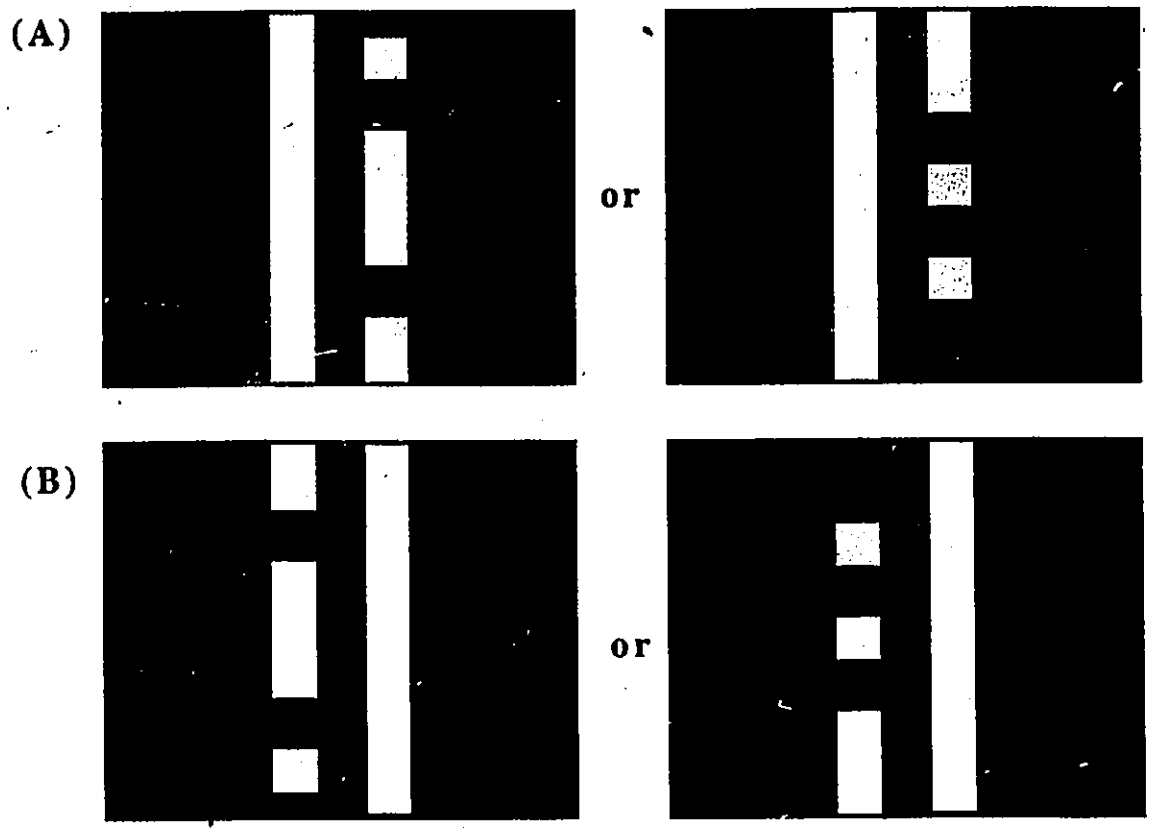


Figure 6.1 - Variations of the Visual Scanning Algorithm

Given in Appendix D are two similar programs that perform the code scanning visually. Shown on the next page by Figure 6.2, is the flow chart for the first of the two programs. The program in the flow chart begins by initializing the variables and the state of the PIP board. The next step is to produce a binary image where the background is black (intensity=0) and the foreground is white (intensity=255). In this test program, the white corresponds to where the guidepath is located and where the PRBS code track is of value "1". In order to obtain this easily processed binary image of the track, three parameters of the program can be adjusted. These parameters are the binary cutoff, the intensity level used as the black/white threshold level, the

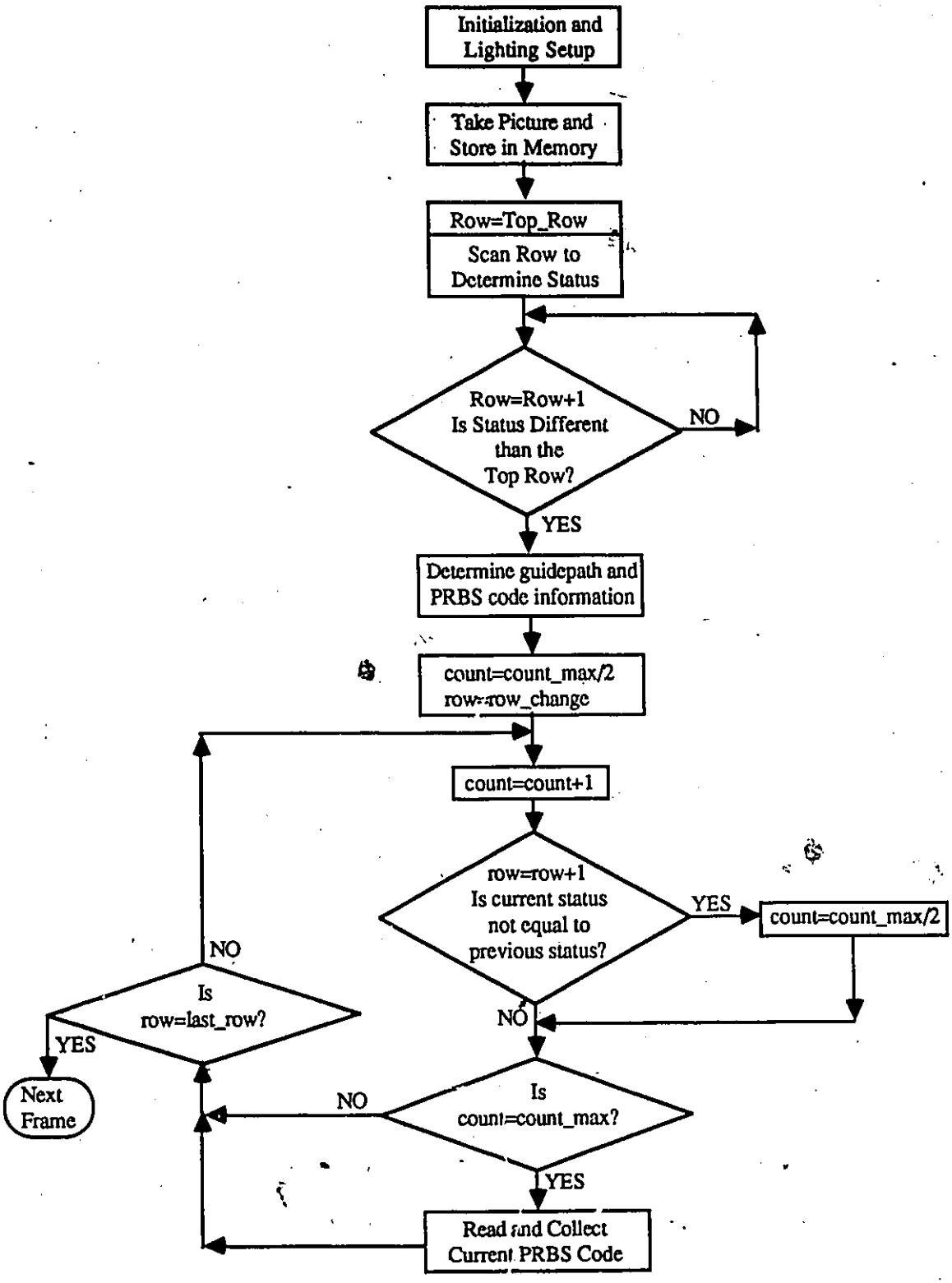


Figure 6.2 Flow Chart for a Visual Scanning Approach

offset, and the gain (values that control the analog voltage range of digitization) of the PIP512 board. If this programming approach fails, then a certain external lighting change may be necessary. After completing this initialization and lighting setup, the PIP512 board performs a single frame grab, which is fully processed by scanning the rows from top to bottom. To begin with, the scanning is continued until the code track changes from its original status to something different. This transition line, the first resynchronization location, must be determined to definitely locate the guidepath and code track and to begin the correct code scanning of the code track. With respect to the code scanning part of the program, the SYNC pulses of the incrementally encoded free wheel are replaced by the value of the variable **count** in the program. Besides this minor change, the 8-tuple corresponding to the current AGV's location is collected in the same way as before.

The execution of the second program is similar except for the fact that the PIP512 board is operated in continuous frame grab mode and thus, the processing is accomplished not by a top to bottom scan but by continuously reading in the changing top row. The primary advantage of this second case is the fact that inter-frame correlation is not required.

The proper execution of these programs exemplifies the fact that such an approach is feasible and could be extended to cope with guidepath tracking. The installation of the camera on the AGV with real-time testing with feedback would ultimately be necessary.

Appendix A

Natural Binary to Pseudorandom Binary Code Conversion Table

Natural Binary Code Value	Pseudorandom Binary Code Value	Natural Binary Code Value	Pseudorandom Binary Code Value
0	00000001	50	10101011
1	00000010	51	01010111
2	00000101	52	10101110
3	00001011	53	01011100
4	00010110	54	10111000
5	00101100	55	01110000
6	01011000	56	11100000
7	10110001	57	11000001
8	01100011	58	10000011
9	11000111	59	00000110
10	10001111	60	00001100
11	00011110	61	00011000
12	00111101	62	00110001
13	01111010	63	01100010
14	11110100	64	11000101
15	11101000	65	10001010
16	11010000	66	00010101
17	10100001	67	00101011
18	01000011	68	01010110
19	10000111	69	10101100
20	00001111	70	01011001
21	00011111	71	10110011
22	00111111	72	01100110
23	01111111	73	11001100
24	11111111	74	10011001
25	11111110	75	00110010
26	11111100	76	01100101
27	11111001	77	11001011
28	11110010	78	10010111
29	11100100	79	00101111
30	11001000	80	01011111
31	10010000	81	10111111
32	00100001	82	01111110
33	01000010	83	11111101
34	10000101	84	11111011
35	00001010	85	11110111
36	00010100	86	11101111
37	00101001	87	11011110
38	01010011	88	10111100
39	10100111	89	01111001
40	01001111	90	11110011
41	10011111	91	11100110
42	00111110	92	11001101
43	01111101	93	10011011
44	11111010	94	00110111
45	11110101	95	01101110
46	11101010	96	11011101
47	11010101	97	10111011
48	10101010	98	01110111
49	01010101	99	11101110

Natural Binary Code Value	Pseudorandom Binary Code Value	Natural Binary Code Value	Pseudorandom Binary Code Value
100	11011100	150	10110000
101	10111001	151	01100001
102	01110010	152	11000010
103	11100101	153	10000100
104	11001010	154	00001000
105	10010101	155	00010001
106	00101010	156	00100010
107	01010100	157	01000101
108	10101001	158	10001011
109	01010010	159	00010111
110	10100101	160	00101110
111	01001010	161	01011101
112	10010100	162	10111010
113	00101000	163	01110101
114	01010001	164	11101011
115	10100010	165	11010111
116	01000100	166	10101111
117	10001001	167	01011110
118	00010010	168	10111101
119	00100101	169	01111011
120	01001011	170	11110110
121	10010110	171	11101101
122	00101101	172	11011011
123	01011010	173	10110111
124	10110100	174	01101111
125	01101000	175	11011111
126	11010001	176	10111110
127	10100011	177	01111100
128	01000110	178	11111000
129	10001100	179	11110000
130	00011001	180	11100001
131	00110011	181	11000011
132	01100111	182	10000110
133	11001110	183	00001101
134	10011100	184	00011010
135	00111001	185	00110100
136	01110011	186	01101001
137	11100111	187	11010011
138	11001111	188	10100110
139	10011110	189	01001101
140	00111100	190	10011010
141	01111000	191	00110101
142	11110001	192	01101011
143	11100011	193	11010110
144	11000110	194	10101101
145	10001101	195	01011011
146	00011011	196	10110110
147	00110110	197	01101101
148	01101100	198	11011010
149	11011000	199	10110101

Natural Binary Code Value	Pseudorandom Binary Code Value	Natural Binary Code Value	Pseudorandom Binary Code Value
200	01101010	227	00110000
201	11010100	228	01100000
202	10101000	229	11000000
203	01010000	230	10000001
204	10100000	231	00000011
205	01000001	232	00000111
206	10000010	233	00001110
207	00000100	234	00011101
208	00001001	235	00111010
209	00010011	236	01110100
210	00100111	237	11101001
211	01001110	238	11010010
212	10011101	239	10100100
213	00111011	240	01001000
214	01110110	241	10010001
215	11101100	242	00100011
216	11011001	243	01000111
217	10110010	244	10001110
218	01100100	245	00011100
219	11001001	246	00111000
220	10010010	247	01110001
221	00100100	248	11100010
222	01001001	249	11000100
223	10010011	250	10001000
224	00100110	251	00010000
225	01001100	252	00100000
226	10011000	253	01000000
		254	10000000

Appendix B

Program for General Purpose Pseudorandom Binary Sequence Generation

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>

/* Set the bit length (3 to 14) and
    the starting code of the pseudorandom binary sequence. */

#define SR_LENGTH 8
#define START_CODE 1
#define CODE_LENGTH (int) pow(2,SR_LENGTH)-1

int forward[8]={-1,-1,-1,-1,-1,-1,-1,-1},
    reverse[8]={-1,-1,-1,-1,-1,-1,-1,-1};

/* Store the forward and reverse feedback equations with the
    selected bit length by using the forward and reverse arrays. */

void initialization()
{
        if (SR_LENGTH==3)
        {       forward[0]=SR_LENGTH-1; forward[1]=SR_LENGTH-3;
                reverse[0]=SR_LENGTH-1; reverse[1]=SR_LENGTH-2;
        }
        if (SR_LENGTH==4)
        {       forward[0]=SR_LENGTH-1; forward[1]=SR_LENGTH-4;
                reverse[0]=SR_LENGTH-1; reverse[1]=SR_LENGTH-2;
        }
        if (SR_LENGTH==5)
        {       forward[0]=SR_LENGTH-2; forward[1]=SR_LENGTH-5;
                reverse[0]=SR_LENGTH-1; reverse[1]=SR_LENGTH-3;
        }
        if (SR_LENGTH==6)
        {       forward[0]=SR_LENGTH-1; forward[1]=SR_LENGTH-6;
                reverse[0]=SR_LENGTH-1; reverse[1]=SR_LENGTH-2;
        }
        if (SR_LENGTH==7)
        {       forward[0]=SR_LENGTH-3; forward[1]=SR_LENGTH-7;
                reverse[0]=SR_LENGTH-1; reverse[1]=SR_LENGTH-4;
        }
        if (SR_LENGTH==8)
        {       forward[0]=SR_LENGTH-2; forward[1]=SR_LENGTH-3;
                forward[2]=SR_LENGTH-4; forward[3]=SR_LENGTH-8;
                reverse[0]=SR_LENGTH-1; reverse[1]=SR_LENGTH-3;
                reverse[2]=SR_LENGTH-4; reverse[3]=SR_LENGTH-5;
        }
        if (SR_LENGTH==9)
        {       forward[0]=SR_LENGTH-4; forward[1]=SR_LENGTH-9;
                reverse[0]=SR_LENGTH-1; reverse[1]=SR_LENGTH-5;
        }
        if (SR_LENGTH==10)
        {       forward[0]=SR_LENGTH-3; forward[1]=SR_LENGTH-10;
                reverse[0]=SR_LENGTH-1; reverse[1]=SR_LENGTH-4;
        }
```

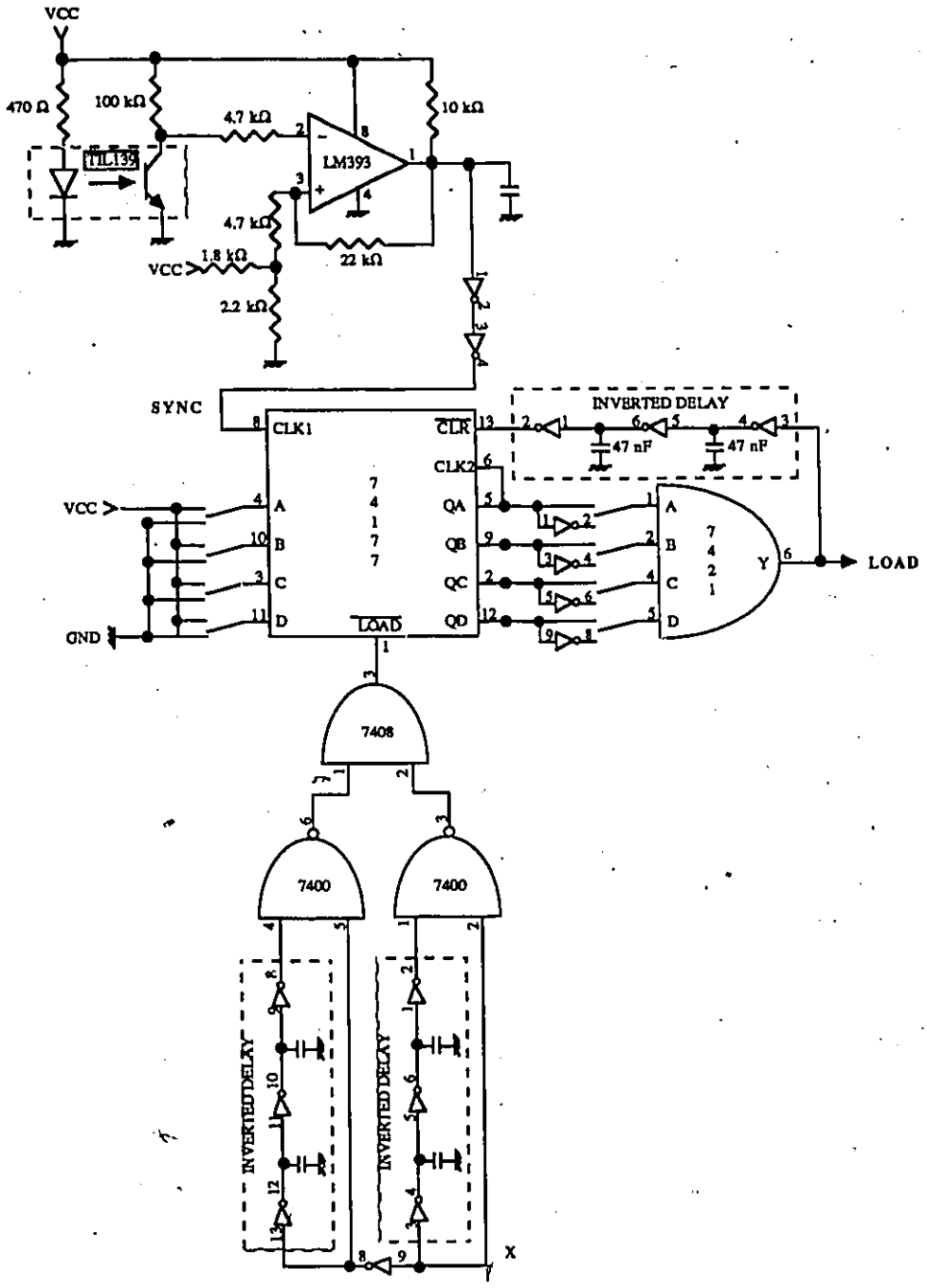
```
        if (SR_LENGTH==11)
        {       forward[0]=SR_LENGTH-2; forward[1]=SR_LENGTH-11;
                reverse[0]=SR_LENGTH-1; reverse[1]=SR_LENGTH-3;
        }
        if (SR_LENGTH==12)
        {       forward[0]=SR_LENGTH-1; forward[1]=SR_LENGTH-4;
                forward[2]=SR_LENGTH-6; forward[3]=SR_LENGTH-12;
                reverse[0]=SR_LENGTH-1; reverse[1]=SR_LENGTH-2;
                reverse[2]=SR_LENGTH-5; reverse[3]=SR_LENGTH-7;
        }
        if (SR_LENGTH==13)
        {       forward[0]=SR_LENGTH-4; forward[1]=SR_LENGTH-10;
                forward[2]=SR_LENGTH-6; forward[3]=SR_LENGTH-13;
                reverse[0]=SR_LENGTH-1; reverse[1]=SR_LENGTH-5;
                reverse[2]=SR_LENGTH-7; reverse[3]=SR_LENGTH-11;
        }
        if (SR_LENGTH==14)
        {       forward[0]=SR_LENGTH-4; forward[1]=SR_LENGTH-13;
                forward[2]=SR_LENGTH-8; forward[3]=SR_LENGTH-14;
                reverse[0]=SR_LENGTH-1; reverse[1]=SR_LENGTH-5;
                reverse[2]=SR_LENGTH-9; reverse[3]=SR_LENGTH-14;
        }
}

/* This procedure takes an integer variable, converts it to its
   binary equivalent, and stores it in a character variable. */

void int_to_bin(int num, char *pb)
{
        int i;
        for (i=SR_LENGTH-1;i>=0;i--,num>>=1)
                pb[i]=(01&num)+'0';
        pb[SR_LENGTH]='\0';
}

/* Given a character variable containing a pseudorandom binary
    code, a single forward or single reverse feedback shift are
    performed by the following two procedures respectively */

void forward_feedback_shift(char *pb)
{       int i=0,xor=0;
        while (forward[i]!=-1)
        {       if (pb[forward[i]]!='0')
                        xor++;
                i++;
        }
        for (i=0;i<=SR_LENGTH-2;i++)
                pb[i]=pb[i+1];
        if (xor/2!=(float)xor/2)
                pb[SR_LENGTH-1]='1';
        else
                pb[SR_LENGTH-1]='0';
}
```

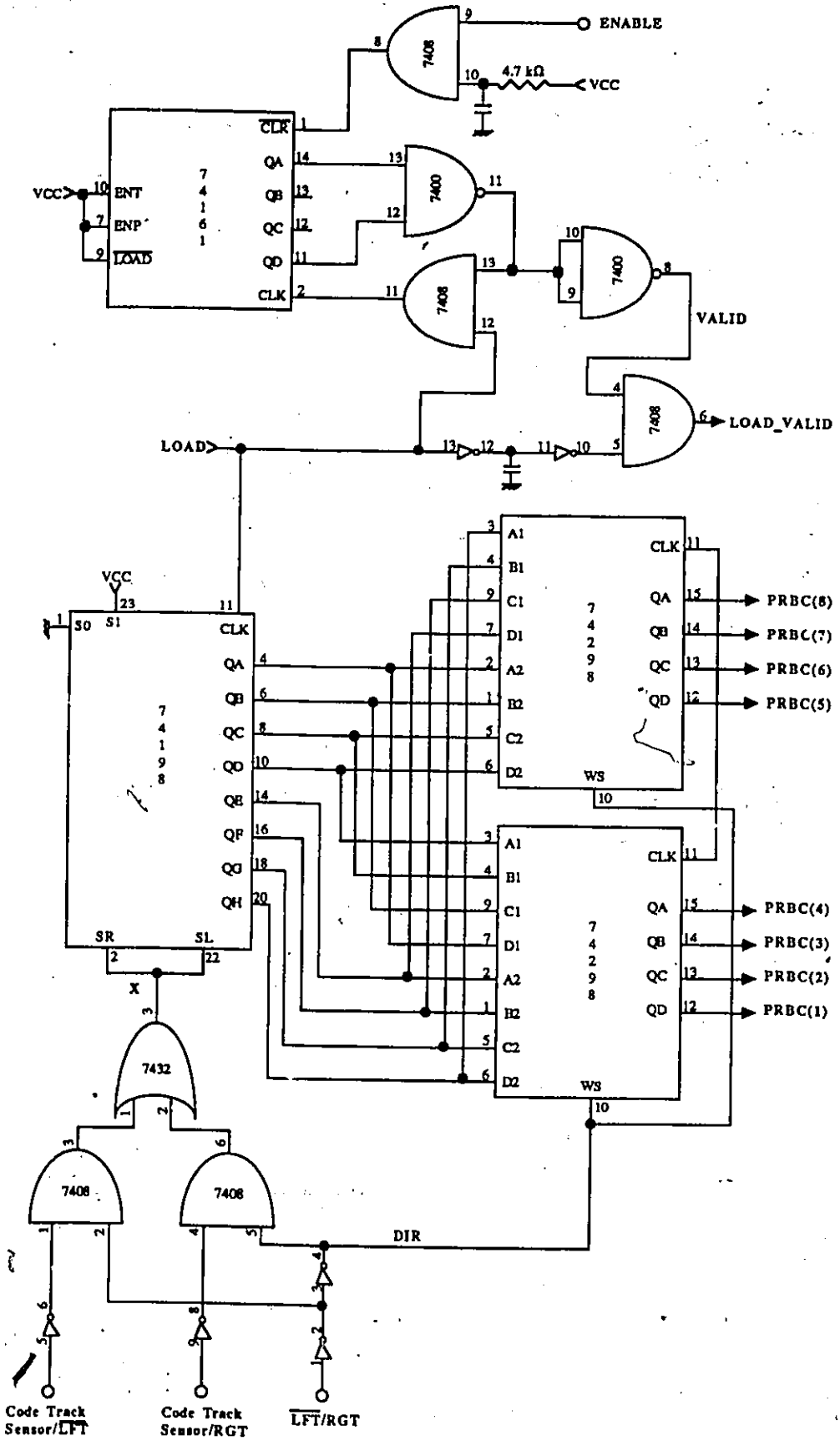
```
void reverse_feedback_shift(char *pb)
{       int i=0,xor=0;
        while (reverse[i]!=-1)
        {       if (pb[reverse[i]]!='0')
                        xor++;
                i++;
        }
        for (i=SR_LENGTH-1;i>=1;i--)
                pb[i]=pb[i-1];
        if (xor/2!=(float)xor/2)
                pb[0]='1';
        else
                pb[0]='0';
}

/* Given a pseudorandom binary code stored in a character
    variable, this routine swaps the order of the bits
    and stores it back in the original variable. */

void binary_code_flip(char *pb)
{       int i;
        char temp[1];
        for (i=0; i<SR_LENGTH/2;i++)
        {       temp[0]=pb[i];
                pb[i]=pb[SR_LENGTH-1-i];
                pb[SR_LENGTH-1-i]=temp[0];
        }
}

/* The following procedure generates the complete code conversion
    table using the forward feedback equations; whereas
    the next procedure uses the reverse feedback equations.
    In either case, the results are printed on the screen. */

forward_prbs_generation()
{
        char code[SR_LENGTH+1],binstr[SR_LENGTH+1];
        int k;
        int_to_bin(START_CODE,code);
        for (k=0;k<=CODE_LENGTH;k++)
        {
                int_to_bin(k,binstr);
                printf ("position[%d,%s]= ",k,binstr);
                printf ("%s\n",code);
                forward_feedback_shift(code);
        }
}


reverse_prbs_generation()
{
        char code[SR_LENGTH+1],binstr[SR_LENGTH+1];
        int k;
        int_to_bin(START_CODE,code);
```

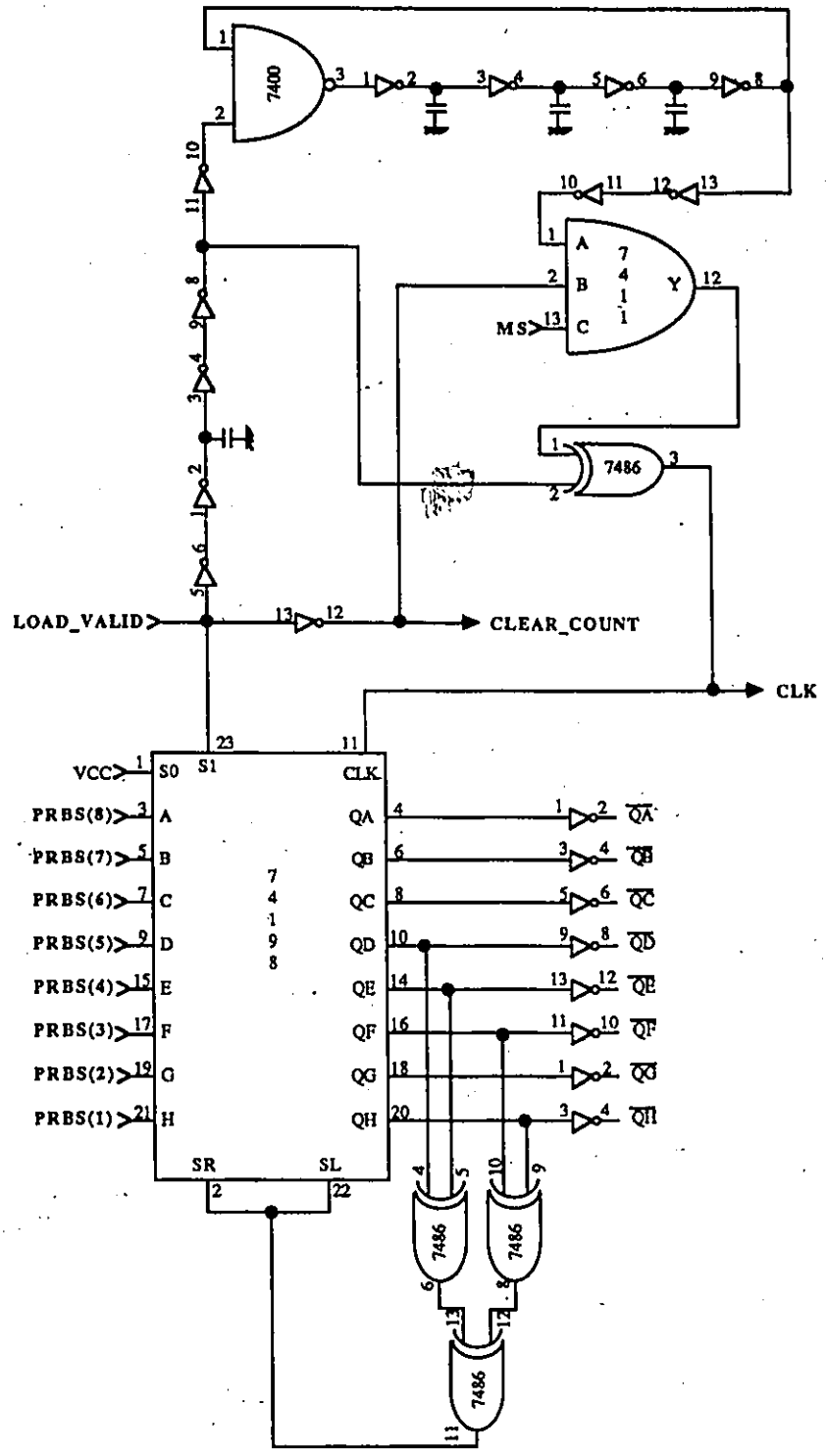
```
        for (k=CODE_LENGTH;k>=0;k--)
        {
                int_to_bin(k,binstr);
                printf ("position[%d,%s]= ",k,binstr);
                reverse_feedback_shift(code);
                printf ("%s\n",code);
        }
}

/* This routine prints out the binary values corresponding to one
    pseudorandom binary sequence on the screen. */

periodic_prbs_code()
{       char code[SR_LENGTH+1],binstr[SR_LENGTH+1];
        int j,k,rem;
        int_to_bin(START_CODE,code);
        printf ("prbs_code . . . . . .\n\n");
        printf ("        %s\n",code);
        for (k=0;k<=CODE_LENGTH-SR_LENGTH;k+=SR_LENGTH)
        {       for (j=1;j<=SR_LENGTH;j++)
                        forward_feedback_shift(code);
                strcpy(binstr,code);
                if (k>CODE_LENGTH-2*SR_LENGTH)
                {       rem=(CODE_LENGTH)-(CODE_LENGTH)/SR_LENGTH*
                                                SR_LENGTH;
                        binstr[rem]='\0';
                }
                printf("        %s\n",binstr);
        }
}

/* Main procedure where after calling initialization one can call
    the above procedures in any order. */

main ()
{
        initialization();

        forward_feedback_shift();

        periodic_prbs_code();
}
```

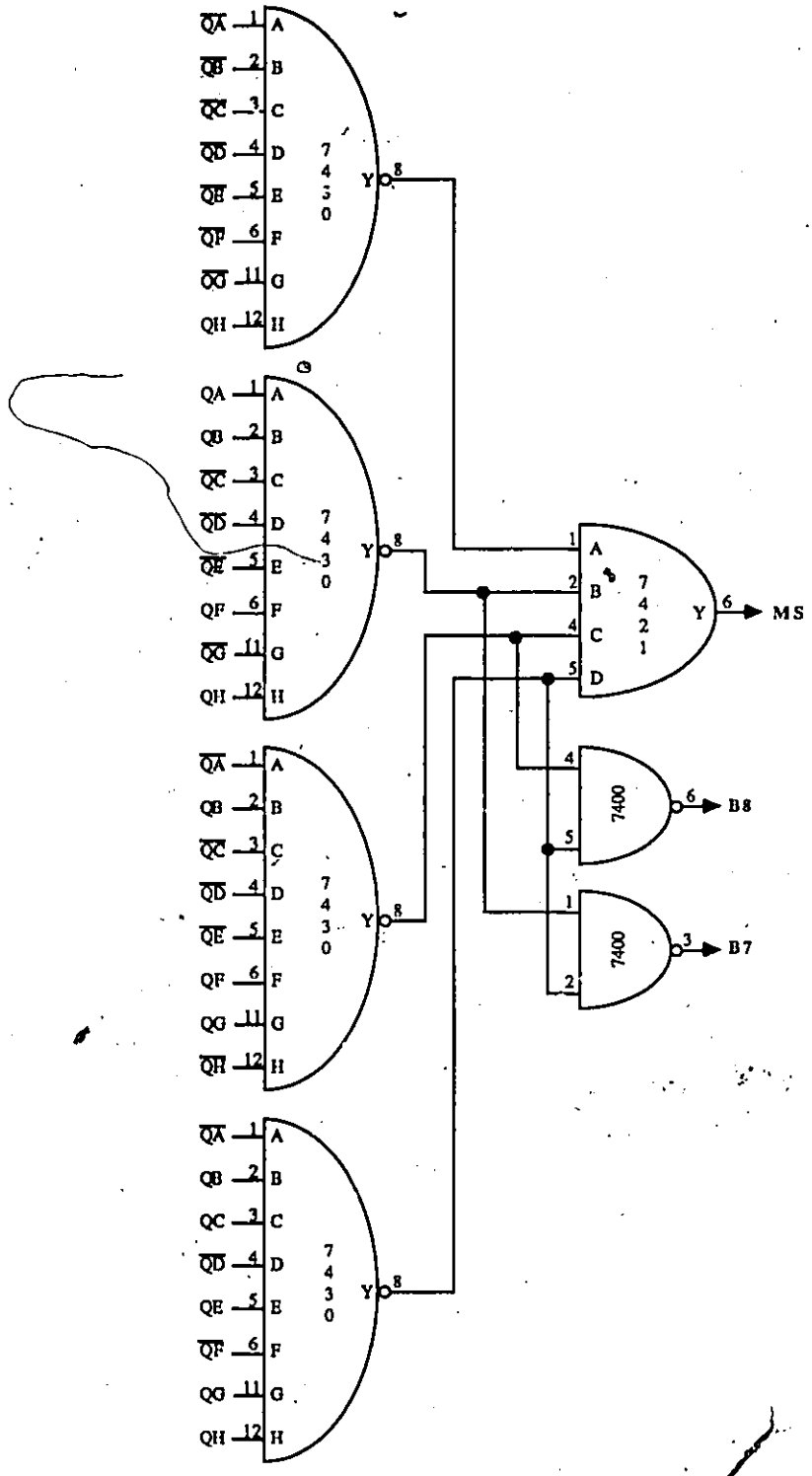
Appendix C

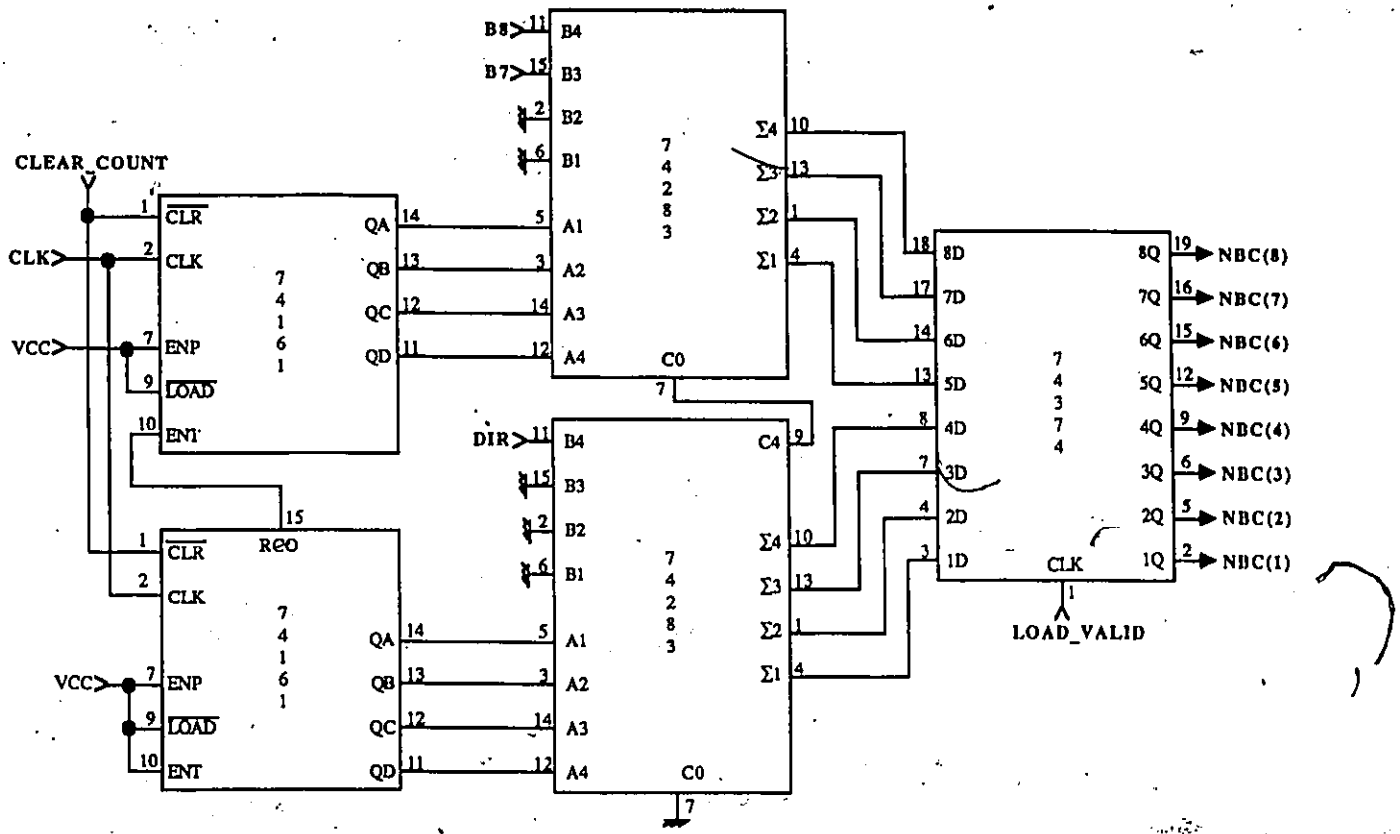
Detailed Circuit Diagrams of the Hardware


VCC
470 Ω
100 kΩ
4.7 kΩ
LM393
10 kΩ
TIL139
4.7 kΩ
22 kΩ
VCC
1.8 kΩ
2.2 kΩ
SYNC
INVERTED DELAY
47 nF
47 nF
CLK1
CLR
CLK2
7
4
1
7
7
VCC
A
B
C
D
QA
QB
QC
QD
LOAD
GND
7
4
2
1
Y
LOAD
7408
7400
7400
INVERTED DELAY
INVERTED DELAY
X


ENABLE
7408
4.7 kΩ
VCC
CLR
QA
QB
QC
QD
CLK
ENT
ENP
LOAD
7400
7408
7400
VALID
7408
LOAD_VALID
LOAD
VCC
S1
S0
SR
SL
7418
7429
A1
B1
C1
D1
A2
B2
C2
D2
WS
PRBC(8)
PRBC(7)
PRBC(6)
PRBC(5)
PRBC(4)
PRBC(3)
PRBC(2)
PRBC(1)
X
7432
7408
7408
DIR
Code Track Sensor/LFT
Code Track Sensor/RGT
LFT/RGT


7400
7411
MS
7486
LOAD_VALID
CLEAR_COUNT
CLK
VCC
PRBS(8)
PRBS(7)
PRBS(6)
PRBS(5)
PRBS(4)
PRBS(3)
PRBS(2)
PRBS(1)
S0
S1
CLK
A
B
C
D
E
F
G
H
QA
QB
QC
QD
QE
QF
QG
QH
SR
SL
7419
8


7430
7421
7400
7400
MS
B8
B7


CLEAR_COUNT
CLK
VCC
VCC
DIR
B8
B7
LOAD_VALID
NBC(8)
NBC(7)
NBC(6)
NBC(5)
NBC(4)
NBC(3)
NBC(2)
NBC(1)
CLR
CLK
ENP
LOAD
ENT
QA
QB
QC
QD
RCO
B4
B3
B2
B1
A1
A2
A3
A4
C0
Σ4
Σ3
Σ2
Σ1
C4
8D
7D
6D
5D
4D
3D
2D
1D
8Q
7Q
6Q
5Q
4Q
3Q
2Q
1Q
7416161
74283
74374

Appendix D

Visual Code Scanning Programs

```
#include <stdio.h>
#include <process.h>

/* All the important constants are defined below. */

#define BASE_ADDR 0x26C
#define VIDEO_STD 0
#define CHANNEL 2
#define EXT_SYNC 1
#define MAP 0
#define COLOR 0
#define START 0
#define LUTVALUES 256
#define MAX_PIXVAL 255
#define CHAR_MAX '\377'
#define CHAR_MIN '\000'
#define ESCAPE '\033'
#define DONE 'q'
#define UP 'u'
#define DOWN 'd'
#define BI_CUTOFF 'b'
#define GAIN 'g'
#define OFFSET 'o'
#define YES 'y'
#define NO 'n'
#define DISABLE 0
#define ENABLE 1
#define QUAD 0
#define WINDLENGTH 512
#define WINDWIDTH 480
#define WINDLEFTCOL 0
#define WINDRIGHTCOL 512
#define ASPECT_RATIO 0.88
#define SR_LENGTH 8

/* Declaration of the global variables. */

int bi_cutoff,gain,offset;
char *lutbuffer,*rowbuffer;
int guidepath_start,guidepath_width,prbs_start,prbs_width;
int status_present,rowchange;
int code_track_position;

/* Clears the screen by using an ANSI sequence. */

void clear_screen ()
{
        printf ("%c[2J",ESCAPE);
}
```

```
/* Initializes the PIP512 board to a known stable state. */

void initialization ()
{
        fg_init (BASE_ADDR);
        fg_video (VIDEO_STD);
        fg_chan (CHANNEL);
        fg_sync (EXT_SYNC);
        fg_sbuf (DISABLE);
}

/* The parameters, which affect the quality of the binary
    image are set to their starting values. */

void starting_point_init ()
{
        extern int bi_cutoff,gain,offset;
        extern char *lutbuffer;
        int index;
        bi_cutoff=128;
        gain=160;
        offset=100;
        for (index=0;index<bi_cutoff;
                        *(lutbuffer+index)=CHAR_MAX,index++);
        for (index=bi_cutoff; index<LUTVALUES;
                        *(lutbuffer+index)=CHAR_MIN,index++);
        fg_lutd (MAP,COLOR,START,LUTVALUES,lutbuffer);
        fg_lutm (MAP);
        fg_gain (gain);
        fg_offset (offset);
}

/* The following routine allows the modification of the
    aforementioned parameters to produce a good quality binary
    image. */

void modify_values ()
{
        extern int bi_cutoff,gain,offset;
        extern char *lutbuffer;
        char choice,parameter;
        parameter=BI_CUTOFF;
        clear_screen ();
        printf ("bi_cutoff=%3d gain=%3d offset=%3d\n"
                                        ,bi_cutoff,gain,offset);
        do
        {  choice=getch();
           if (choice==BI_CUTOFF || choice==GAIN || choice==OFFSET)
                parameter=choice;
           else if (choice==UP)
                { if (parameter==BI_CUTOFF)
                        {  if (bi_cutoff<LUTVALUES)
                                { *(lutbuffer+bi_cutoff)=CHAR_MAX;
                                  ++bi_cutoff;
                                  fg_lutd (MAP,COLOR,START,
                                        LUTVALUES,lutbuffer);}}
```

```
                else if (parameter==GAIN)
                     {  if (gain<MAX_PIXVAL)
                          { ++gain;
                            fg_gain (gain);}}
                        else if (parameter==OFFSET)
                             {  if (offset<MAX_PIXVAL)
                                  {  ++offset;
                                     fg_offset (offset);}}}
          else if (choice==DOWN)
               { if (parameter==BI_CUTOFF)
                     {  if (bi_cutoff>DISABLE)
                          { --bi_cutoff;
                            *(lutbuffer+bi_cutoff)=CHAR_MIN;
                            fg_lutd (MAP,COLOR,START,
                                     LUTVALUES,lutbuffer);}}
                  else if (parameter==GAIN)
                     {  if (gain>DISABLE)
                        --gain;
                        fg_gain (gain);}
                  else if (parameter==OFFSET)
                     {  if (offset>DISABLE)
                          { --offset;
                            fg_offset (offset);}}}
             clear_screen ();
             printf ("bi_cutoff=%3d gain=%3d offset=%3d\n"
                                  ,bi_cutoff,gain,offset);}
        while (choice!=DONE);
}

/* The above two procedures together constitute the lighting
    setup procedure. */

void lighting_setup ()
{
        starting_point_init ();

        modify_values ();
}

/* The PIP512 performs a frame grab - taking and storing a picture
    of 512 x 512 pixels. */

void take_picture()
{
        fg_cgrab(DISABLE);
        fg_snap (ENABLE);
        fg_sbuf (ENABLE);
}
```

```
/* This procedure determines the row where the first code track
    transition takes place and then calculates the location and
    width of the guidepath and the code track. */

void starting_point ()
{
      extern int guidepath_start,guidepath_width
                              ,prbs_start,prbs_width;
      extern int status_present,rowchange;
      extern char *rowbuffer;
      char temp;
      int rowread,column;
      int status_old,status;

      /* This double loop determines the value of rowchange. */

      for (rowread=0;rowread<WINDWIDTH;rowread++)
      { fg_rowr (rowread,QUAD,rowbuffer);
        status_present=0;
        temp=*rowbuffer;
        for (column=WINDLEFTCOL+1;column<WINDRIGHTCOL;column++)
        { if (temp!=*(rowbuffer+column))
              { temp=*(rowbuffer+column);
                status_present++; }}
          if (rowread==0)
              status_old=status_present;
          if (status_present!=status_old)
              {rowchange=rowread;
               break;}
        }

      /* Obtaining the track positional information follows. */

      if (status_present==2)
            rowread=rowchange;
      else if (status_present==4)
            rowread=rowchange-1;
      fg_rowr (rowread,QUAD,rowbuffer);
      status=0;
      for (column=WINDLEFTCOL;column<WINDRIGHTCOL;column++)
      { temp=*(rowbuffer+column);
        if (status==0)
              { if (temp!=CHAR_MIN)
                      { status=1;
                        guidepath_start=column;}}
        else if (status==1)
              { if (temp!=CHAR_MAX)
                      { guidepath_width=column-guidepath_start;
                        break; }}
      }
```

```
        if (status_present==2)
                rowread=rowchange-5;
        else if (status_present==4)
                rowread=rowchange+5;
        fg_rowr (rowread,QUAD,rowbuffer);
        status=0;
        for (column=guidepath_start+3*guidepath_width/2;
                                        column<WINDRIGHTCOL;column++)
        { temp=*(rowbuffer+column);
                if (status==0)
                        { if (temp!=CHAR_MIN)
                                { status=1;
                                  prbs_start=column;}}
                else if (status==1)
                        { if (temp!=CHAR_MAX)
                                { prbs_width=column-prbs_start;
                                  code_track_position=0;
                                  break; }}
        }
        if (status==0)
        { for (column=guidepath_start-guidepath_width/2;
                                                column>=0;column--)
                { temp=*(rowbuffer+column);
                        if (status==0)
                                { if (temp!=CHAR_MIN)
                                        { status=1;
                                          prbs_width=column;}}
                        else if (status==1)
                                { if (temp!=CHAR_MAX)
                                        { prbs_width-=column;
                                          prbs_start=column;
                                          code_track_position=1;
                                          break; }}
                }
        }
}

/* With the data from the previous procedure, the code scanning
    routine can easily obtain  the pseudorandom binary code
    values. */

void auto_synchronization_scanning ()
{
        extern int guidepath_start,guidepath_width,
                                                prbs_start,prbs_width;
        extern int status_present,rowchange;
        extern char *rowbuffer;
        int rowread,column,count,count_max,status,i;
        char position[SR_LENGTH+1];
        strcpy (position,"00000000");
        count_max=guidepath_width/ASPECT_RATIO;
        count=count_max/2;
        printf ("count=%d\n",count);
```

```
        for (rowread=rowchange;rowread<WINDWIDTH;rowread++)
        { fg_rowr (rowread,QUAD,rowbuffer);
          count++;
          status=2;
          for (column=prbs_start-prbs_width/2;
                column<prbs_start+3*prbs_width/2;column++)
              { if (*(rowbuffer+column)!=CHAR_MIN)
                    {.status=4;
                                break; }}
          if (status_present!=status)
              { status_present=status;
                count=count_max/2;}
          if (count==count_max)
              { count=0;
                fg_setind(0);
                fg_moveto(0,rowread);
                fg_lineto(511,rowread);
                for (i=0;i<=SR_LENGTH-2;i++)
                      position[i]=position[i+1];
                if (status_present==2)
                      position[SR_LENGTH-1]='0';
                else if (status_present==4)
                      position[SR_LENGTH-1]='1';
              }
        }
}

/* The above 2 routines make up the PRBS code scanning
    routines. */

void PRBS_code_scanning ()
{
        starting_point ();

        auto_synchronization_scanning ();
}

/* The main procedure sets up the buffers and calls the
    appropriate procedures to correctly scan one complete
    picture. */

main()

{
        extern char *lutbuffer,*rowbuffer;
        lutbuffer=(char *) calloc(LUTVALUES,sizeof (char));
        rowbuffer=(char *) calloc(WINDLENGTH,sizeof (char));

        initialization ();
        lighting_setup ();
        take_picture();
        PRBS_code_scanning();
}
```

```
/* This program is very similar to the previous one, except for
    the following changes:
    (a) the fundamental change is in the way that the next row
        for processing is read in.  In the previous program, the
        new row was simply the next row in the buffer; however,
        in this program the next row is always the first row in
        the buffer because the frame buffer is being updated
        dynamically.
    (b) This row reading change affects three procedures from the
        previous program. Firstly, the take_picture procedure has
        been replaced by a row_read procedure (always the first
        row).  The other 2 routines, which have been affected by
        this alteration are starting_point and
        auto_synchronization_scanning, have been modified to
        properly account for this difference.  Note that the
        functional purposes of the latter 2 procedures is the
        same. */

#include <stdio.h>
#include <process.h>

#define BASE_ADDR 0x26C
#define VIDEO_STD 0
#define CHANNEL 2
#define EXT_SYNC 1
#define MAP 0
#define COLOR 0
#define START 0
#define LUTVALUES 256
#define MAX_PIXVAL 255
#define CHAR_MAX '\377'
#define CHAR_MIN '\000'
#define ESCAPE '\033'
#define DONE 'q'
#define UP 'u'
#define DOWN 'd'
#define BI_CUTOFF 'b'
#define GAIN 'g'
#define OFFSET 'o'
#define YES 'y'
#define NO 'n'
#define DISABLE 0
#define ENABLE 1
#define QUAD 0
#define WINDLENGTH 512
#define WINDWIDTH 480
#define WINDLEFTCOL 0
#define WINDRIGHTCOL 512
#define ASPECT_RATIO 0.88
#define SR_LENGTH 8
```

```
int bi_cutoff,gain,offset;
char *lutbuffer, *rowbuffer;
int guidepath_start,guidepath_width,prbs_start,prbs_width;
int status_present,rowchange;
int code_track_position;

void clear_screen ()
{
        printf ("%c[2J",ESCAPE);
}

void initialization ()
{
        fg_init (BASE_ADDR);
        fg_video (VIDEO_STD);
        fg_chan (CHANNEL);
        fg_sync (EXT_SYNC);
        fg_cgrab (ENABLE);
}

void starting_point_init ()
{
        extern int bi_cutoff,gain,offset;
        extern char *lutbuffer;
        int index;
        bi_cutoff=128;
        gain=160;
        offset=100;
        for (index=0;index<bi_cutoff;
                        *(lutbuffer+index)=CHAR_MAX,index++);
        for (index=bi_cutoff; index<LUTVALUES;
                        *(lutbuffer+index)=CHAR_MIN,index++);
        fg_lutd (MAP,COLOR,START,LUTVALUES,lutbuffer);
        fg_lutm (MAP);
        fg_gain (gain);
        fg_offset (offset);
}

void modify_values ()
{
        extern int bi_cutoff,gain,offset;
        extern char *lutbuffer;
        char choice,parameter;
        parameter=BI_CUTOFF;
        clear_screen ();
        printf ("bi_cutoff=%3d gain=%3d offset=%3d\n"
                                        ,bi_cutoff,gain,offset);
        do
        {  choice=getch();
           if (choice==BI_CUTOFF || choice==GAIN || choice==OFFSET)
                parameter=choice;
```

```
            else if (choice==UP)
                { if (parameter==BI_CUTOFF)
                        {  if (bi_cutoff<LUTVALUES)
                                { *(lutbuffer+bi_cutoff)=CHAR_MAX;
                                  ++bi_cutoff;
                                  fg_lutd (MAP,COLOR,START,
                                                LUTVALUES,lutbuffer);}}
                    else if (parameter==GAIN)
                        {  if (gain<MAX_PIXVAL)
                                { ++gain;
                                  fg_gain (gain);}}
                    else if (parameter==OFFSET)
                        {  if (offset<MAX_PIXVAL)
                                { ++offset;
                                  fg_offset (offset);}}}
          else if (choice==DOWN)
               { if (parameter==BI_CUTOFF)
                        {  if (bi_cutoff>DISABLE)
                                { --bi_cutoff;
                                  *(lutbuffer+bi_cutoff)=CHAR_MIN;
                                  fg_lutd (MAP,COLOR,START,
                                                LUTVALUES,lutbuffer);}}
                    else if (parameter==GAIN)
                        {  if (gain>DISABLE)
                                --gain;
                                fg_gain (gain);}
                    else if (parameter==OFFSET)
                        {  if (offset>DISABLE)
                                { --offset;
                                  fg_offset (offset);}}}
          clear_screen ();
          printf ("bi_cutoff=%3d gain=%3d offset=%3d\n"
                                        ,bi_cutoff,gain,offset);}
        while (choice!=DONE);
}


void lighting_setup ()
{
        starting_point_init ();

        modify_values ();
}

void row_read()
{
        fg_cgrab(DISABLE);
        fg_rowr (0,QUAD,rowbuffer);
        fg_cgrab (ENABLE);
}
```

```
void starting_point ()
{
        extern int guidepath_start,guidepath_width
                                ,prbs_start,prbs_width;
        extern int status_present;
        extern char *rowbuffer;
        char temp;
        int column,temp_start,temp_width;
        int status_old,status;

        row_read();
        status_old=0;
        for (column=WINDLEFTCOL;column<WINDRIGHTCOL;column++)
        { temp=*(rowbuffer+column);
          if (status_old==0)
                { if (temp!=CHAR_MIN)
                        { guidepath_start=column;
                          status_old=1;}}
          if (status_old==1)
                { if (temp!=CHAR_MAX)
                        { guidepath_width=column-guidepath_start;
                          status_old=2;}}
          if (status_old==2)
                { if (temp!=CHAR_MIN)
                        { prbs_start=column;
                          status_old=3;}}
          if (status_old==3)
                { if (temp!=CHAR_MAX)
                        { prbs_width=column-prbs_start;
                          status_old=4;
                          break;  }}
        }

        do
        { row_read();
          status_present=0;
          temp=CHAR_MIN;
          for (column=WINDLEFTCOL;column<WINDRIGHTCOL;column++)
                {  if (temp!=*(rowbuffer+column))
                        { temp=*(rowbuffer+column);
                          status_present++; }}
        }
        while (status_present==status_old);


        if (status_present==2)
        { status=0;
          for (column=WINDLEFTCOL;column<WINDRIGHTCOL;column++)
                { temp=*(rowbuffer+column);
                    if (status==0)
                        { if (temp!=CHAR_MIN)
                                { status=1;
                                  temp_start=column;}}
```

```
                else if (status==1)
                    { if (temp!=CHAR_MAX)
                            { temp_width=column-temp_start;
                              break; }}
                }
        if (guidepath_start<=temp_start-temp_width/2 ||
                guidepath_start>=temp_start+temp_width/2)
                { prbs_start=guidepath_start;
                  prbs_width=guidepath_width;
                  guidepath_start=temp_start;
                  guidepath_width=temp_width;
                }
        }

        else if (status_present==4)
        {  status=0;
           for (column=WINDLEFTCOL;column<WINDRIGHTCOL;column++)
                { temp=*(rowbuffer+column);
                  if (status==0)
                        { if (temp!=CHAR_MIN)
                                { temp_start=column;
                                  status=1;}}
                  if (status==1)
                        { if (temp!=CHAR_MAX)
                                { temp_width=column-temp_start;
                                  status=2;}}
                  if (status==2)
                        { if (temp!=CHAR_MIN)
                                { prbs_start=column;
                                  status=3;}}
                  if (status==3)
                        {if (temp!=CHAR_MAX)
                                { prbs_width=column-prbs_start;
                                  break;  }}
                }
        if (guidepath_start<=temp_start-temp_width/2 ||
                guidepath_start>=temp_start+temp_width/2)
            { prbs_start=temp_start;  prbs_width=temp_width;}
        }

        if (2*prbs_width<=guidepath_width)
                prbs_width=guidepath_width;
}


void auto_synchronization_scanning ()
{
        extern int guidepath_start,guidepath_width
                                ,prbs_start,prbs_width;
        extern int status_present;
        extern char *rowbuffer;
        int column,count,count_max,status,i,j=0;
        char position[SR_LENGTH+1],temp;
```

```
        strcpy (position,"00000000");
        count_max=guidepath_width/ASPECT_RATIO;
        count=count_max/2;
        do
        { row_read();
          count++;      j++;
          status=2;
          for (column=prbs_start-prbs_width/2;
                column<prbs_start+3*prbs_width/2;column++)
          { temp=*(rowbuffer+column);
                if (status==2)
                        { if (temp!=CHAR_MIN)
                          status=3;}
                if (status==3)
                        { if (temp!=CHAR_MAX)
                                { status=4;
                                  break; }}
          }
          if (status_present!=status)
                { status_present=status;
                        count=count_max/2;}
          if (count==count_max)
                { count=0;
                  for (i=0;i<=SR_LENGTH-2;i++)
                        position[i]=position[i+1];
                  if (status_present==2)
                        position[SR_LENGTH-1]='0';
                  else if (status_present==4)
                        position[SR_LENGTH-1]='1';
                  printf ("position=%s\n",position);
                }
          }
          while (j!=1000);
}

void PRBS_code_scanning ()
{
        starting_point ();

        auto_synchronization_scanning ();
}

main()

{
        extern char *lutbuffer,*rowbuffer;
        lutbuffer=(char *) calloc(LUTVALUES,sizeof (char));
        rowbuffer=(char *) calloc(WINDLENGTH,sizeof (char));

        initialization ();
        lighting_setup ();
        PRBS_code_scanning();
}
```

Bibliography

[1] Marsh, Peter (consultant), Robots. New York, New York: Salamander Books Limited, 1985.

[2] Fu, K.S., Gonzalez, R.C., and Lee, C.S.G., Robotics: Control, Sensing, Vision, and Intelligence. New York, New York: McGraw-Hill Book Company, 1987.

[3] Day, Charles R. Jr., "Robots Man the Mail Carts", Industry Week, Volume 223, No. 4, November 12, 1984, pp. 113-114.

[4] Miller, Richard K., Automated Guided Vehicles and Automated Manufacturing. Dearborn, Michigan: Society of Manufacturing Engineers, 1987.

[5] Zygmont, Jeffrey, "Guided Vehicles Set Manufacturing in Motion", High Technology, December 1986, pp. 16-21.

[6] Bohlander, Ronald A., "Understanding AGVS Today", Proceedings of AGVS86 Seminar and Exhibition, Atlanta, Georgia, October 1986, pp. 1-14.

[7] Bohlander, Ronald A., "Making Science Work on the Shop Floor: A Review of New Developments and Research in AGVS", Proceedings of AGVS87 Seminar and Exhibition, Pittsburgh, Pennsylvania, October 1987, pp. 7-23.

[8] Hollier, R.H. (editor), Automated Guided Vehicle Systems, Springer-Verlag, United Kingdom: IFS (Publications) Limited, 1987.

[9] Hammond, G.C., "Evolutionary AGVS - From Concept to Present Reality", Stratford-upon-Avon, United Kingdom, Executive Briefing on Automated Guided Vehicles, November 1986.

[10] Müller, T., "AGVS in Europe - Current Techniques and Future Trends", Stratford-upon-Avon, United Kingdom, Executive Briefing on Automated Guided Vehicles, November 1986.

[11] McEllin, P., "AGV Designs", Automated Guided Vehicle Systems, Springer-Verlag, United Kingdom: IFS (Publications) Limited., 1987, pp. 53 - 63.

[12] Boegli, P., "A Comparative Evaluation of AGV Navigation Techniques", Proceedings of the 3rd International Conference on Automated Guided Vehicle Systems, Stockholm, Sweden, October 1985.

[13] Premi, S.K., and Besant, C.B., "A Review of Various Vehicle Guidance Techniques that can be used by Mobile Robots or AGVs", Proceedings of the 2nd International Conference on Automated Guided Vehicle Systems, Stuttgart, Germany, June 1983, pp. 195-209.

[14] Hongo, Takero, et al., "An Automatic Guidance System of a Self-Controlled Vehicle", IEEE Transactions on Industrial Electronics, Vol. IE-34, No. 1, February 1987. pp. 5-10.

[15] Komatsu, N., and Nakano, K., "Dead Reckoning Guidance Combined with Guide Path Method for AGVs", Proceedings of the 5th International Conference on Automated Guided Vehicle Systems, Tokyo, Japan, October 1987, pp. 123-136.

[16] Tsukagoshi, T., et al., "Magnetic Lattice Range Guide for AGVs", Proceedings of the 5th International Conference on Automated Guided Vehicle Systems, Tokyo, Japan, October 1987, pp. 137-144.

[17] Murata, M., et al., "Ultrasonic Guided Autonomous Vehicle", Proceedings of the 5th International Conference on Automated Guided Vehicle Systems, Tokyo, Japan, October 1987, pp. 145-156.

[18] Kopp, G., "Wireless Guidance of AGVs", Proceedings of the 5th International Conference on Automated Guided Vehicle Systems, Tokyo, Japan, October 1987, pp. 157-170.

[19] Miura, T., et al., "Automated Guided Vehicle using Magnetic Marker", Proceedings of the 3rd International Conference on Automated Guided Vehicle Systems, Stockholm, Sweden, October 1985, pp. 181-185.

[20] Walker, S.P., et al., "Free-Ranging AGV and Scheduling System", Proceedings of the 3rd International Conference on Automated Guided Vehicle Systems, Stockholm, Sweden, October 1985.

[21] Solberg J.J., and McGillen, C.D., "An Intelligent Factory Transport System", Proceedings of the 3rd International Conference on Automated Materials Handling, Birmingham, United Kingdom, March, 1986.

[22] "An Unmanned Vehicle that is not Restricted to Fixed Routes", Industrial Contractors Holland, a brief introduction to FROG (free ranging on grid) vehicle.

[23] Kabuka, Mansur R., and Arenas, Alvaro E., "Position Verification of a Mobile Robot Using Standard Pattern", IEEE Journal of Robotics and Automation, Vol. RA-3, No. 6, December 1987, pp. 505-516.

[24] Marriott, M., "An Overview of Bar Code Technology", Proceedings of the 3rd International Conference on Automated Materials Handling, Birmingham, United Kingdom, March, 1986.

[25] Larcombe, M.H.E., "Tracking Stability of Wire Guided Vehicles", Proceedings of the 1st International Conference on Automated Guided Vehicles, Stratford-upon-Avon, United Kingdom, June 1981, pp. 137-144.

[26] Schraft, R.D., et al., "Mobile Robots - Key Elements for the Integration of Transport and Handling Functions", Proceeding of the 15th International Conference on Industrial Robots, Tokyo, Japan, September 1985.

[27] Mandel, Kostia, and Duffie, Neil A., "On-Line Compensation of Mobile Robot Docking Errors", IEEE Journal of Robotics and Automation, Vol. RA-3, No. 6, December 1987, pp. 591-598.

[28] Adams, Walt, "Automated Guided Vehicle Control Systems Design, Application, and Selection", Proceedings of AGVS86 Seminar and Exhibition, Atlanta, Georgia, October 1986, pp. 34-47.

[29] Egbelu, Pius J., and Tanchoco, J.M.A., "Potentials for Bi-directional Guide-Path for Automated Guided Vehicle Based Systems", International Journal of Production Research, Vol. 24, No. 5, September/October 1986, pp. 1075-1097.

[30] McEllin, P., "AGV Design for FMS", Proceedings of the 3rd International Conference on Automated Guided Vehicle Systems, Stockholm, Sweden, October 1985, pp. 345-355.

[31] Rygh, O.B., "New AGVS Applications", Proceedings of the 3rd International Conference on Automated Guided Vehicle Systems, Stockholm, Sweden, October 1985, pp. 357-361.

[32] Turpin, D.R., "Inertial Guidance: Is it a viable guidance system for AGVS", Proceedings of the 4th International Conference on Automated Guided Vehicle Systems, Chicago, USA, June 1986, pp. 301-320.

[33] Yeung, Dit-Yan, and George A. Bekey, " A Decentralized Approach to the Motion Planning Problem for Multiple Mobile Robots", 1987 IEEE International Conference on Robotics and Automation, Raleigh, North Carolina, March 1987, pp. 1779-1784.

[34] Grossman, David D., "Traffic Control of Multiple Robot Vehicles", IEEE Journal of Robotics and Automation, Vol. 4, No. 5, October 1988, pp. 491-497.

[35] Lindgren, H., "Centralised and Decentralised Control of AGVs", Proceedings of the 3rd International Conference on Automated Guided Vehicle Systems, Stockholm, Sweden, October 1985.

[36] MacWilliams, F. Jessie, and Sloane, Neil J.A., "Pseudo-Random Sequences and Arrays", Proceedings of the IEEE, December 1976, pp. 1715-1729.

[37] Heath, F.G., and Gribble, M.W., "Chain Codes and their Electronic Applications", The Institution of Electrical Engineering, Monograph No. 392 M, July 1960, pp. 50-57.

[38] Schöbi, P., "Memory Structure Speeds Generation of Pseudonoise Sequences", Computer Design, March 1981, pp. 130-142.

[39] Ronse, Christian, Feedback Shift Registers, Springer-Verlag, United Kingdom: Lecture Notes in Computer Science, 1984, pp. 3-4.

[40] E.M. Petriu, and J.S. Basran, "Resource Management for a Multi-Arm Robotic Assembly Cell", IEEE Canadian Programmable Control and Automation Technology Conference, Toronto, Ontario, October 1988, pp. 13A2-4, 1-5.

[41] E.M. Petriu, "Absolute-type Position Transducers using a Pseudorandom Encoding", IEEE Transactions on Instrumentation and Measurement, Vol. IM-36, December 1987, pp. 950-955.

[42] E.M. Petriu, "Absolute Position Recovery for Automated Path-Guided Vehicles", presented at ROBOTS 12 and VISION '88 Conference and Exhibition, Detroit, Michigan, June 5-9, 1988.

[43] E.M. Petriu, "New Pseudorandom/Natural Code Conversion Method", Electronic Letters, Vol. 24, No. 22, October 27, 1988, pp. 1358-1359.

[44] E.M. Petriu, and J.S. Basran, "On the Position Measurement of Automated Guided Vehicles", IEEE Transactions on Instrumentation and Measurement, Vol. 38, No. 3, June 1989, pp. 799-803.

[45] J.S. Basran, E.M. Petriu, and Frans C.A. Groen, "Developments in the Measuring of the Absolute Position of Automated Guided Vehicles", IEEE Instrumentation and Measurement Technology Conference, April 25-27, 1988, Washington, D.C., pp. 132-137.

[46] E.M. Petriu, J.S. Basran, and Frans C.A. Groen, "New Cost-Effective Techniques Enhance AGV Absolute Position Measurement", submitted for possible publication in a special issue of IEEE Transactions on Instrumentation and Measurement.