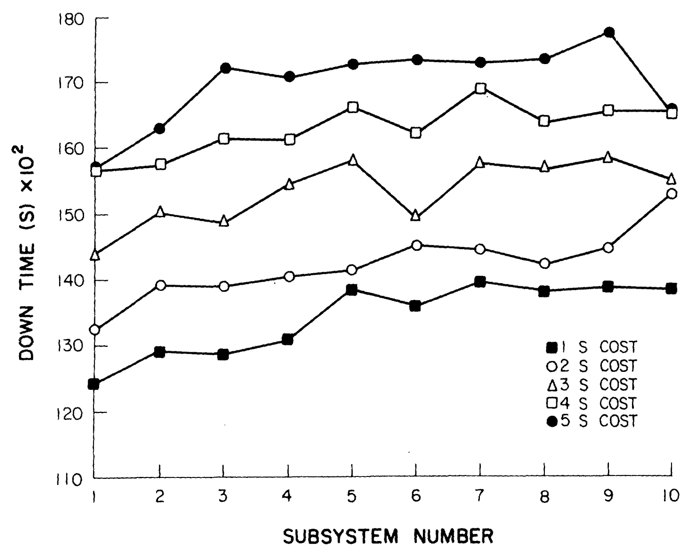


Fig. 6. Total down time for each subsystem for different costs of explicit communication.

than 5 s would cause explicit communication to result in larger down times than implicit communication. Thus the time penalties incurred by explicit communication can offset the performance decrement expected from implicit communication using an imperfect model. If the time penalties and degree of model validity can be quantified for a given application, simulation analyses can be used to determine the most efficient communication technique for that application.

Discussion

Human–computer communication is a critical design issue for human–computer systems which employ dynamic task allocation. Communication of the human's action plan to the computer can be achieved by explicit or implicit means. Appropriate use of these means of communication and of the information communicated improves system performance.

The results of the first study indicate that the use of a model to convey the human's action plan to the computer implicitly can significantly enhance performance, even when the model is imperfect. This occurs because less conflict and redundancy is present; the computer is better able to select its own actions to complement the actions of the human. The availability of a model of the human is not sufficient for successful implementation of such a system, however. An appropriate algorithm for employment of the model must also be developed. A poor choice of algorithm can lead to poorer performance than that obtained with no communication at all.

The results of the second simulation indicate that the time cost of explicit communication can be traded against the misinformation cost of implicit communication using an imperfect model. Situations are likely to arise in which employment of one or the other technique is advantageous. If the time costs of explicit communication and the predictive validity of candidate models can be quantified for a given application, the simulation methodology utilized in this correspondence offers a means to determine effective communication subsystems for human–computer interaction.

This work demonstrates the need for models of human decisionmaking performance and algorithms which describe how to use such models effectively. Given an appropriate algorithm for use of the model, implicit communication based upon a model achieving quite modest levels of predictive validity can enhance system performance relative to a no-communication baseline. Thus implementation of implicit communication within human–computer systems need not require the development and use of complex all-encompassing models of human performance. This work also demonstrates the need to define efficient dialog styles for systems employing explicit communication between human and computer decision makers in multiple-task time-constrained situations. It thus serves as a basis for research in model, algorithm, and dialog design to increase human–computer system performance and to utilize computers within these systems in a manner more compatible with the human's capabilities.

References

[1] W. B. Rouse, "Human-computer interaction in multiple task situations," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-5, pp. 384–392, 1977.
[2] ——, "Human-computer interaction in the control of dynamic systems," *ACM Computing Surveys*, vol. 13, pp. 71–99, 1981.
[3] ——, "Human interaction with an intelligent computer in multi-task situations," in *Proc. 11th Annu. Conf. Manual Control*, NASA–Ames Research Center, NASA TM X-62, 464, 1975, pp. 130–143.
[4] J. S. Greenstein, "The use of models of human decision making to enhance human–computer interaction," in *Proc. 1980 IEEE Int. Conf. Cybernetics and Society*, 1980, pp. 968–970.
[5] M. K. Tulga and T. B. Sheridan, "Dynamic decisions and workload in multi-task supervisory control," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-10, pp. 217–231, 1980.
[6] J. S. Greenstein and W. B. Rouse, "A model of human decisionmaking in multiple process monitoring situations," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-12, pp. 182–193, 1982.
[7] K. R. Pattipati, D. L. Kleinman, and A. R. Ephrath, "A dynamic decision model of human task selection performance," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-13, pp. 145–166, 1983.
[8] J. S. Greenstein and M. E. Revesman, "Development and validation of a mathematical model of human decisionmaking for human-computer communication," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-16, pp. 148–154, 1986.
[9] M. E. Revesman and J. S. Greenstein, "Application of a mathematical model of human decisionmaking for human-computer communication," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-16, pp. 142–147, 1986.
[10] J. S. Greenstein and S. T. Lam, "An experimental study of dialogue-based communication for dynamic human-computer task allocation," *Int. J. Man-Mach. Studies*, vol. 23, pp. 605–621, 1985.

Adaptive Navigation of Automated Vehicles by Image Analysis Techniques

WEN-HSIANG TSAI AND YUNG-CHO CHEN

***Abstract*—Image analysis techniques are applied to adaptive automatic vehicle navigation. The proposed image-based navigation system is made adaptive to follow any selected path embedded in a curve-type path network. This is achieved with three major capabilities of the proposed system: path network learning, reference path setup, and guided path navigation. The first capability enables the system to extract relevant information out of a given network map, and the second collects along-path reference data for a selected path from the extracted network information. During guided path navigation, consecutive path images are taken by a television camera on the vehicle and then analyzed for navigation control along path curves and for angular turning at path crossings. The control structure of the automatic navigation process is modelled as a Moore-type sequential machine in automata theory. Correct path navigation is ascertained by verifying each path crossing encountered on the road against the**

Manuscript received August 15, 1985; revised April 19, 1986.
W. H. Tsai is with the Department of Information Science and the Microelectronics and Information Science and Technology Research Center, National Chiao Tung University, Hsinchu, Taiwan 30050, Republic of China.
Y. C. Chen was with the Institute of Computer Engineering, National Chiao Tung University, Hsinchu, Taiwan 30050, Republic of China. He is now with the Institute of Information Industry, Taipei, Taiwan, Republic of China.
IEEE Log Number 8609833.

reference data by an image matching technique. Simulation of vehicle movement with a computer-controlled pantilt is also described. Simulation results show the feasibility of the proposed approach.

I. Introduction

The study of automatically guided vehicles has received considerable attention recently because of their potential applicability in several domains. Such vehicles are suggested to solve partially the increasingly complex modern transportation problems [1]–[3]. They can be used in automated warehouses for inventory control. In robotics, they serve as the legs of robots [4]. They are also applicable in pilotless aero-navigation.

A central issue of automated vehicle systems is to guide an individual vehicle to follow a desired or preselected path [5], [6]. In conventional reference systems, guidance signals are obtained either from mechanical wall followers [7] or from embedded current-excited wires [8]. Recently, sidewalls or guardrails have also been proposed as a new type of reference systems for vehicle control by radar or ultrasonic signals [9], [10].

With the advance of optical sensing and computer vision technology, it now becomes feasible to use low-cost optical sensors such as video cameras for automated vehicle guidance. This approach has been employed in ground vehicle guidance to avoid path obstacles [4], [11]–[13]. We present another application of this approach to automatic vehicle navigation to follow a preselected path. The reference on the path, from which guidance signals can be obtained, is any line or curve optically detectable with a video camera. In present-day roads and highways, painted stripes on the two sides of a traffic lane serve as such reference. For a vehicle or aircraft flying at a reasonable height from the ground, highways themselves may be used as the reference. However, the proposed automatic navigation by curve-type reference paths seems most applicable in indoor environments in which curve stripes may be painted either on the floor or on the ceiling. Typical examples of such environments are large warehouses or buildings. Thus the proposed approach will be particularly useful for warehouse or indoor transportation automation.

For automated vehicles to be more useful in real-world applications, they should be provided with the capability of navigating in a path network, instead of just along a single route. This makes flexible path scheduling and automatic path planning possible. To accomplish this goal, the navigation capability should include not only smooth following of path curves but also angular turning at path crossings. It is also desirable to make the vehicle capable of recognizing all the crossings expected during a navigation session.

To this end, a navigation control scheme, which includes both the path-turning and the crossing-recognition capabilities, is proposed here. The scheme is based on image analysis techniques. In addition, the navigation system is also made capable of extracting the information contained in any path network expected to be visited, as well as the information contained in any selected navigation path. With these capabilities, the system may be said to possess *adaptive* navigation functions and can traverse different paths in any given network, just like today's manually driven cars.

In short, three types of navigation capabilities can be identified in the proposed automated vehicle system, namely, *path network learning*, *reference path setup*, and *guided path navigation*. Each capability is built up by performing several basic steps as illustrated in Fig. 1.

A. Path Network Learning

The first step in network learning is network image segmentation. To concentrate on the study of navigation control, only simulated network maps with hand-drawn curves are imaged and processed. However, it should be noted that hand-drawn network maps with relative scaling identical to real networks are adequate for indoor applications, where curve stripes are painted on floors or ceilings as reference paths. For example, Fig. 2(a) is a network map image, and Fig. 2(b) shows the segmented network paths after image thresholding and curve thinning techniques are applied [15].

The next step is to extract useful path information for guided path navigation, including lengths, directions, and interconnection structures of the path curves in the network. Such information is organized systematically into a network database for efficient information retrieval during reference path setup.

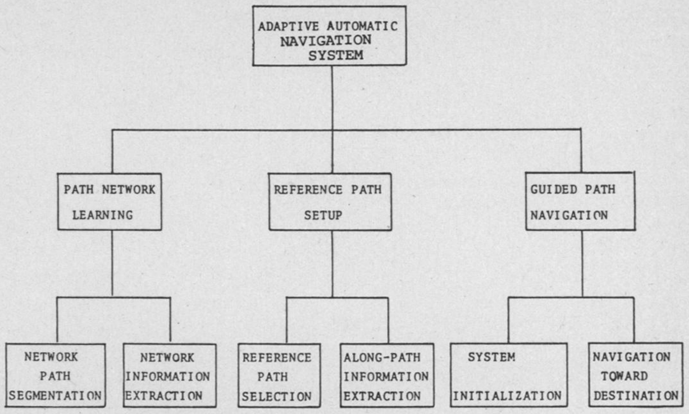
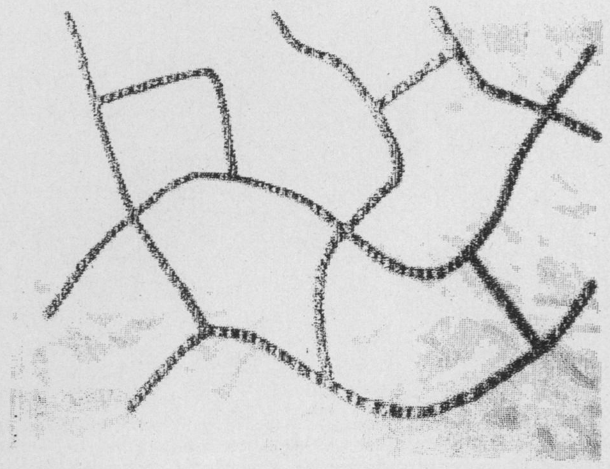
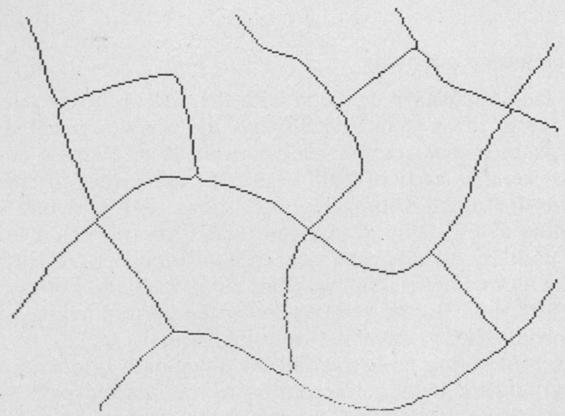

ADAPTIVE AUTOMATIC NAVIGATION SYSTEM
PATH NETWORK LEARNING
REFERENCE PATH SETUP
GUIDED PATH NAVIGATION
NETWORK PATH SEGMENTATION
NETWORK INFORMATION EXTRACTION
REFERENCE PATH SELECTION
ALONG-PATH INFORMATION EXTRACTION
SYSTEM INITIALIZATION
NAVIGATION TOWARD DESTINATION

Fig. 1. System configuration of proposed adaptive automatic navigation system.



(a)



(b)

Fig. 2. Network path segmentation. (a) Image of hand-drawn path network. (b) Segmented path network from Fig. 2(a) after image thresholding and curve thinning.

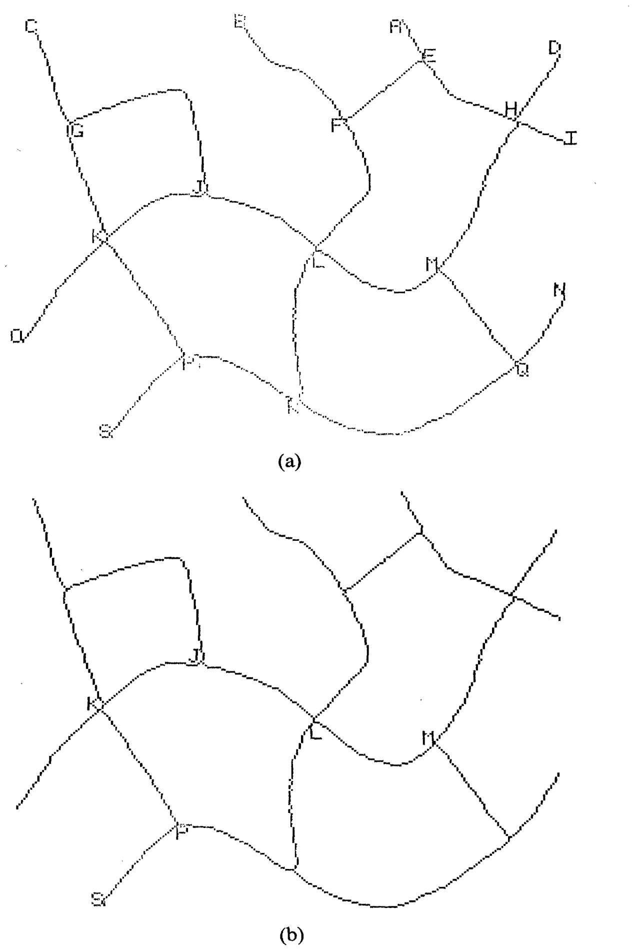


Fig. 3. Reference path selection. (a) Labelled path network of Fig. 2(b) displayed on television monitor with alphabets at all crossing and terminal points. (b) Redisplay of path network with selected reference path represented by label sequence SPKJLM.

B. Reference Path Setup

To facilitate user interaction with the system during reference path setup, the system first displays the whole segmented path network as a picture on a television monitor. All path crossing points (on the ends of path segments) are given different alphabets for identification. Fig. 3(a) shows such a picture corresponding to Fig. 2(b). A user can then arbitrarily select a reference path by specifying a sequence of alphabets, each representing a crossing or terminal point along the path. For example, with SPKJLM as the selected path, the corresponding picture returned by the system is shown in Fig. 3(b).

The next step is to extract all the along-path information out of the network database according to the selected path points. The procedure essentially is to trace the selected path through the database and collect useful information in the mean time. The details will be described in Section III. The collected information is organized as a *reference path database* or simply *reference data*.

C. Guided Path Navigation

The basic idea of image-based guided navigation is to take an image of the current vehicle location, extract relevant local information from the image, and then match the result with the reference data to generate an error signal for navigation direction correction. It will be shown in the next section that this process can be modeled as a Moore sequential machine in finite automata theory [16].

To start a guided navigation session, the proposed approach requires that the vehicle be brought as close to the starting point as possible. The control state and a set of navigation parameters are also initialized. The system then performs input scene classification at each visited spot by matching the input local image with the reference data to determine what type of scene is being observed locally. Based on the classified input and the current control state, the system enters another state and generates an error signal for direction change in the next step.

Lacking a vehicle for actual field testing, only the design principle of the control structure and the image processing techniques for automatic navigation are emphasized in this study. However, the proposed system has been simulated with a television camera and a pantilt, both controlled by a multiprocessor system [14]. The simulation result is very encouraging. Moreover, the system is designed in such a way that the interface between the image-based control and the physical vehicle is kept minimum. Therefore, the simulation setup is believed to be easily adaptable to real applications.

In the remainder of this work, the Moore machine model for system control is presented in Section II. In Section III, path network and reference path information extraction is described. Detailed image processing techniques used in the basic control steps are described in Section IV. Simulation results and discussions are included in Section V.

II. Finite-Automata Model for Image-Based Guided Path Navigation

A Moore sequential machine determines its next state and its output by its input and its current state, respectively. This is found useful for modeling the control structure of the proposed system. To facilitate the discussions in the sequel, some graph-theoretic terminologies are defined here. With a path network regarded as a graph, a terminal or crossing point is just a node in the graph. More specifically, a terminal point will be called a *terminal* node, and a crossing point a *nonterminal* node. On the other hand, the path (line or curve) segment between two nodes will be called an *edge*. Thus, a terminal node is one with only one incident edge, and a nonterminal node one with more than two incident edges. Note that a node with two incident edges is not considered to exist in a network graph here; it is simply regarded as a path point on an edge in this study. Finally, the two nodes incident to an edge will be called the *end* nodes of the edge. In the following, the camera's field of view will be called the *image window*.

A. Control States of Finite-Automata Model

When a vehicle is traversing a path embedded in a network, at least five situations may be identified according to what is "seen" in an image window:

1) the vehicle is moving along an edge;
2) the vehicle has arrived at a non-destination node (called the *expected* node) which is expected to be visited;
3) the vehicle is leaving a node (called the *old* node) just visited in the previous control steps;
4) the vehicle has arrived at a destination node; and
5) the vehicle has arrived at a non-identifiable node or a location without any edge or node in the window.

As an illustration, let Fig. 4(a) be a path network with the three labelled nodes A, B, C as a selected reference path. The above five situations are shown in Fig. 4(b)–(f), respectively, in which dotted squares are image windows. Each of the above five situations may be considered as a control state for the finite automata model. They will be denoted symbolically in the following discussions as 1) TAE (traverse along an edge), 2) REN (reach the expected node), 3) LON (leave the old node), 4) RDN (reach the destination node), and 5) RUN (reach unidentified node), respectively. Situation 3) (i.e., LON) is possible when image taking rate is high enough. That is, a node may be seen more than once through the image window before the vehicle *completely* passes it. Several cases may cause situation 5) (i.e., RUN) to occur: a) the

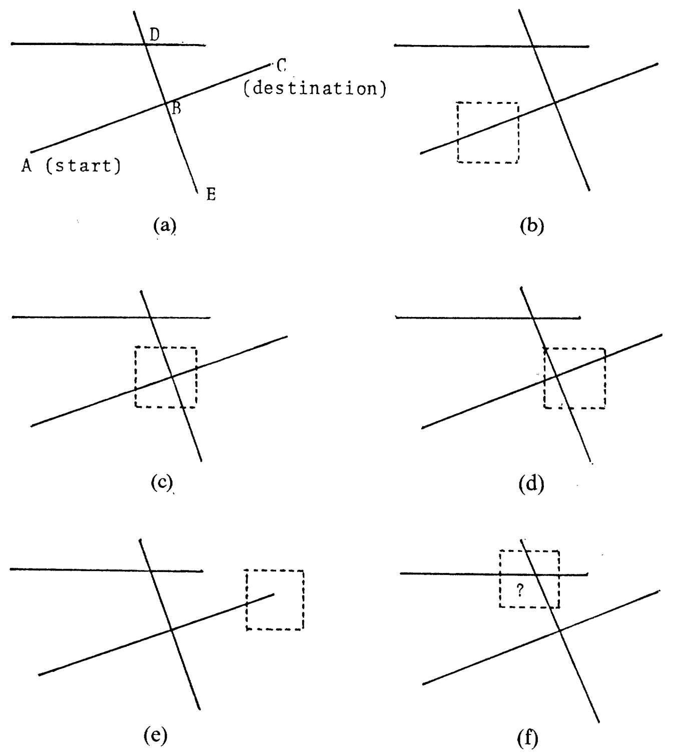


Fig. 4. Illustration of control states corresponding to input scenes. (a) Small network with reference path ABC. (b) Moving along an edge. (c) Arrive at expected node. (d) Leaving old node. (e) Arrive at destination node. (f) Arrive at nonidentifiable node.

vehicle goes astray to a node not along the selected path as in the case shown by Fig. 4(f); b) the path network map, and so the reference data, does not include all physically existing nodes along the selected path; c) image noise causes an irreal node to appear in the image window after image processing; and d) the vehicle goes astray laterally so far from its path that no path edge or node can be found in the image window.

Finally, two system parameters are mentioned here, which will be used later in this section for determining state transition for the finite-automata model. One is the length l for which the vehicle has moved along the edge currently being traversed (called the *current edge*), measured from the old node. Another is just the total length L of the current edge.

B. State Responses of Finite-Automata Model

Once the vehicle enters a control state, its response to the state is automatic and fixed, as described in the following.

1) In the TAE state, the response is to move further along the current edge in a direction determined by the current vehicle center and the next edge point to be visited.

2) In the REN state, the response is to move to the next edge which the vehicle should turn to at the expected node just reached. The move direction is determined by the current vehicle center and an edge point to be visited on the next edge.

3) In the LON state, the response is also to move further on the current edge in a direction determined similarly to 1).

4) In the RDN state, the response is simply to stop the navigation.

5) In the RUN state, the response is also to stop.

It seems that the response of 3) is identical to that of 1), but actually it is not. To move further on the current edge, which edge in the window is the current one should be determined first because in the LON state, several edges incident to the old node appear in the image window (see Fig. 4(d) for an example). Similar edge identification is also found necessary for the response of 2) in order to choose the next edge to turn to from the several ones appearing in the window (e.g., see Fig. 4(c)). The details in terms of image processing techniques will be described in Section IV. The above five responses will be symbolically denoted as MFCE (move further on the current edge), TTNE (turn to the next edge), MFON (move further near the old node), and STOP (stop) for 4) and 5) above, respectively.

C. System Inputs to Finite-Automata Model

Input scene classification is required to determine the type of scene at the spot currently being visited. A classification algorithm is present here. Graphically, only three types of scene are seen in an image window, i.e., nothing, a node, or an edge. But for the case of a node, it may be the expected node, the old node, the destination node, or an erroneous or nonidentifiable node. Therefore, totally five types of input scene can be identified with one erroneous type including the cases of nothing and non-identifiable nodes. The classification algorithm assigns an input image window to one of these five types, based on three sources of information: a) system parameters l and L; b) local reference information about the old node and the expected node; and c) whether the expected node is the destination. All the information except l is part of the reference data.

First, the ratio $r = l/L$ provides some information to check if the input scene is consistent with what is expected according to the reference data. Suppose that a node, say N, is found in the image window now. If the current value of l is approximately equal to L (i.e., $r \approx 1$), then at least it can be sure that the appearance of N is not inconsistent with the reference data (which includes the current edge and its length L). But this traverse-distance check, according to l and L, is not adequate to assure that N is indeed the correct node to be visited next because the vehicle might have erroneously navigated to an unexpected edge and reached a node N' at a distance also approximately equal to L from the old node. In other words, N and N' are non-distinguishable simply by the ratio r.

Hence a further check, called *node scene matching*, is performed in the classification procedure, using the second information source: the local reference information about the expected node. The details will be described in Section IV-B. Briefly speaking, the check is to find out whether the number of incident edges and the angles between edge pairs included in the input scene are all similar to those of the reference data. On the contrary, when r is approximately zero, i.e., when l is small compared with L, the node N found in the window is very possibly the old node because the vehicle is still very close to the old node. But to be sure, node scene matching again has to be performed to check the similarity of N to the old node. Failure of traverse-distance check or node scene matching (henceforth called *node consistency check*) means that the input window contains a nonidentifiable node.

When an input scene passes node consistency check, one more check according to the third information source above is performed to see if it is the destination node. The flow diagram shown in Fig. 5 gives a summary of the above discussions on input scene classification, where thresholds t_1 through t_4 are constrained by the following inequalities and are determined by experiments:

$$t_1 \leqslant t_2 \leqslant t_3 \leqslant t_4;\ t_1 \leqslant 0;\ t_4 \geqslant 1.$$

The result of input scene classification is a type label assigned to the input window. The five type labels are EDGE (edge), EXP NODE (expected node), OLD NODE (old node), DEST NODE (destination node), and ERROR (error).

D. State Transition of Finite-Automata Model

With previous discussions on system states, responses, and inputs, it is now ready to describe the most essential part of the proposed finite automata model, namely, its state transition.

Initially, the start point of the vehicle may be regarded as the old node. Thus the initial state of the system is LON. Two final states are RDN and RUN. Once the vehicle is in either of them,

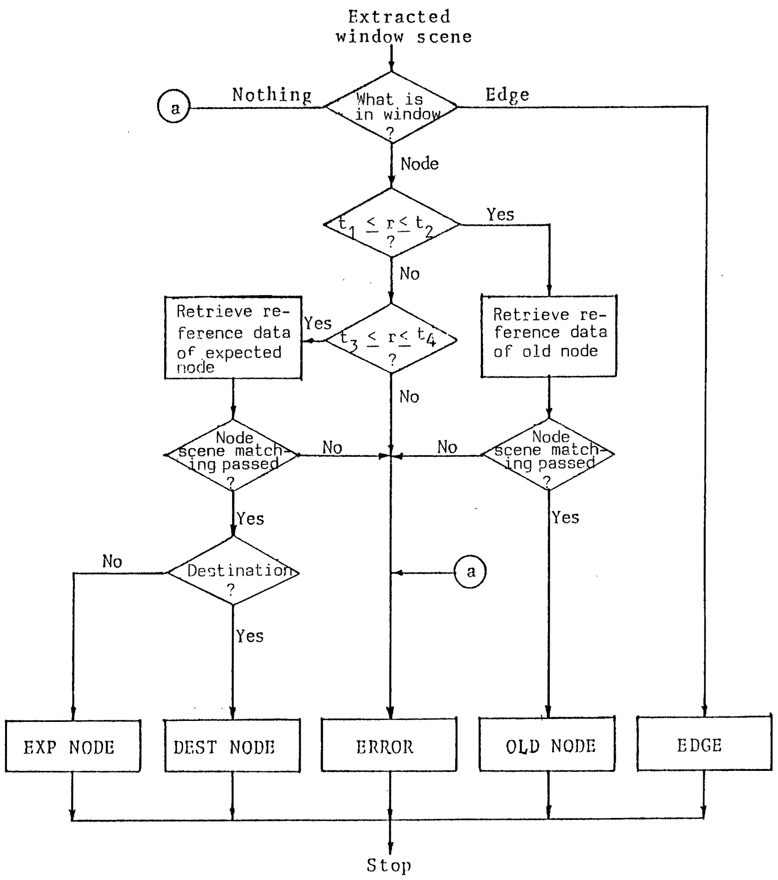


Fig. 5. Input scene classification procedure (note: $r = l/L$).

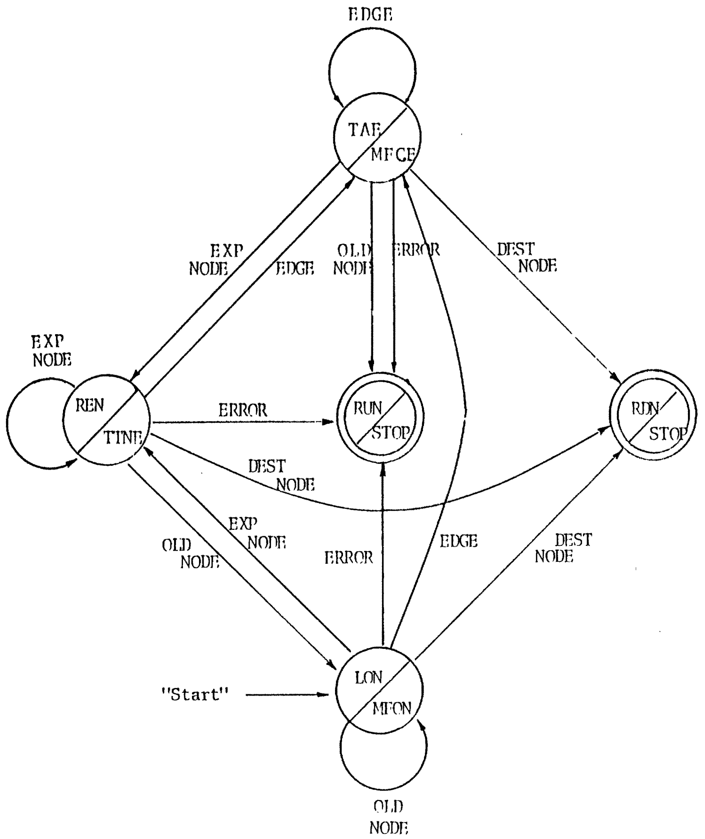


Fig. 6. State transition diagram of Moore-machine model for guided path navigation.

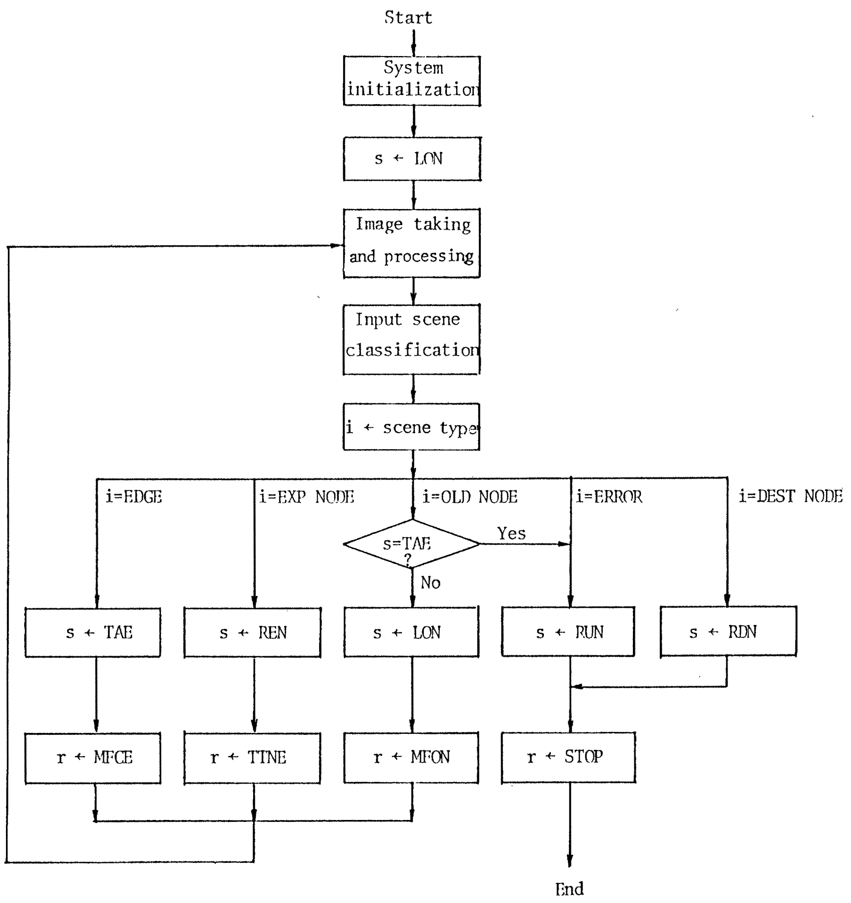


Fig. 7. Guided path navigation procedure based on state transition of Fig. 6 (s = state, i = input, r = response).

navigation is stopped. The overall state transition diagram is shown in Fig. 6 in which the initial state is marked with the "start" label and the final states are double circled. The diagram seems complicated but is really not. Briefly speaking, in each of the three nonfinal states TAE, REN, and LON the five types of inputs, EDGE, EXP NODE, OLD NODE, DEST NODE, and ERROR, bring the system to enter the five control states, TAE, REN, LON, RDN, and RUN, respectively, except that the input OLD NODE causes the TAE state to transit into the erroneous RUN state. The reason for the exception is that once the vehicle leaves an old node and is moving along an edge (i.e., in the TAE state), it will never backward to visit the old node again. A block diagram illustrating the sequential steps of the guided path navigation based on the state-transition diagram is shown in Fig. 7. For example, for the consecutive input image windows shown in Fig. 4(b)–(f), the sequences of system states, responses, and inputs, respectively, are as follows (assuming that TAE is the current state and that Fig. 4(b) is the input image window seen after the response to the current state MFCE is completed):

1) *input sequence:* EDGE, EXP NODE, OLD NODE, DEST NODE;
2) *state sequence:* TAE, REN, LON, RDN;
3) *response sequence:* MFCE, TTNE, MFON, STOP.

III. Extraction and Organization of Path Network and Reference Path Information

In this section, we describe the method we use for organizing path network and reference path information, as well as the procedures for extracting these two types of information. In the following, an edge with two end nodes i and j will be denoted as (i, j).

A. Path Network Information Extraction and Organization

In the network database, an *entry* is created for each node or edge in the network. The entry for each node a contains the number N_a of the incident edges of a, and the image coordinates X_a and Y_a of a. In addition, a direction list L_a is created for a, which includes the directions of all the incident edges of a. The entry for each edge (i, j) contains the length $L(i, j)$ of the edge.

The input to the network information extraction procedure is a graph of interconnected curve segments whose widths are one point in each direction. The first step is to collect nodes, including terminal and nonterminal ones, by checking the curve points one by one. All nodes thus collected, together with their attributes, are stored into a set of node entries. The next step in the extraction procedure is to trace all edges, starting from the collected nodes. For each node a, tracing begins from a neighboring point i of a, and then a neighboring point j of i, and so on, until a node, say b, is found. This completes the tracing of edge (a, b). The tracing process then continues from another neighboring point of a which leads to another incident edge of a, and then again from the third neighboring point, and so on, until all incident edges of a are traced. The whole process above is repeated next for another node until all nodes are processed. All node and edge attributes are also computed in the process. The final result will be a complete network database.

B. Reference Path Information Extraction and Organization

The information of a reference path is specified by a *sequential* list of entries, with each entry being created for a node encountered along the path. The entry for each node a contains 1) the number N_a of the incident edges of a; 2) the angle $T(c, b)$ to turn at node a (called the *turning angle*) from the "incoming" incident edge (c, a) to the "outgoing" edge (a, b); 3) the length $L(c, a)$ of the incoming edge (c, a); and 4) a list A_a of angle entries. Each angle entry in A_a contains the mutual angle between a pair of "neighboring" edges of a. Here by an "incoming" edge (i, j) in a reference path, we mean the incident edge of node j along which j can be reached. An outgoing edge is similarly defined. And by two neighboring edges, we mean, e.g., the edges

(B, E) and (B, C) in Fig. 4(a). Edges (B, E) and (B, D), on the contrary, are not neighboring. The angles in each angle list are computed and arranged in a special order in the extraction procedure of the reference data described next. Note that the above information for the reference path is just a subset of that for the network except that direction lists are replaced by angle lists. Angle lists are required here for the purpose of making node scene matching rotation-free.

The input to the reference path information extraction procedure, as mentioned previously, is a node–label sequence representing the selected reference path. The sequential list of entries, which forms the reference path, can be collected easily by tracing through the path nodes one by one in the network database, excepted that a direction list, instead of an angle list, is collected for each node now. The direction list collected for each node is then transformed into a corresponding angle list as described in the following.

Let $D_1, D_2, \cdots, D_m$ be the directions of all the incident edges of node a. The transformation begins by sorting all D_i values into order. Let $D'_1, D'_2, \cdots, D'_m$ be the result with D'_1 being the smallest. Since all direction values are measured with respect to a reference direction, it is easy to figure out that D'_i and D'_{i+1} are the directions of two neighboring edges, and their mutual angle, denoted as A_i, can be computed according to the following two equations:

$$A_i = D'_{i+1} - D'_i$$
$$A_m = (360° + D'_1) - D'_m, \qquad \text{for } 1 \leqslant i \leqslant m - 1.$$

Let the direction of the incoming edge of a be D'_j. Then, the A_i values are re-indexed in such a way that the one with value $A_j = D'_{j+1} - D'_j$ is the new A_1, followed by others in the counterclockwise order as the new $A_2, A_3, \cdots, A_m$. Each A_i will be called an *interedge angle* in the following discussions. These A_i values form the angle list of node a, with A_1 as the first entry in the list.

IV. Image Processing Techniques for Guided Path Navigation

In this section, we discuss the image processing techniques used in the basic control steps of guided path navigation.

A. Input Image Processing

The procedure to process an input image window basically is identical to that for path network learning, including curve segmentation (by thresholding and thinning) and window information extraction (by node collection, edge tracing, and attribute computation). Extracted window information includes a node entry and a list of incident edge directions. Such information is the window data. An edge or a node found in the input image window is the input edge or node. An example of input window is shown in Fig. 8.

A basic operation which is necessary in most of the control steps described subsequently is the decision on which edge in the input window is the incoming edge or the outgoing edge. There are four distinct cases which should be considered.

Case 1: Determine which input edge is the incoming edge along which the vehicle was approaching an expected node in the last control step (see Fig. 9(a)).

Case 2: Determine which input edge was the incoming edge while the vehicle was leaving an old node along the outgoing edge in the last control step (see Fig. 9(b)).

Case 3: Determine which input edge is the outgoing edge along which the vehicle was leaving an old node in the last control step (see Fig. 9(c)).

Case 4: Determine which input edge is the outgoing edge while the vehicle was approaching an expected node in the last control step (Fig 9(d)).

One solution to the problem of Case 1 is to compare one by one all the input edge directions with the navigation direction D_f of the last control step and choose the input edge with its direction D_i closest to the opposite of D_f as the incoming edge. Note that D_f is one of the system parameter available and is updated in every control step. It is defined as the direction of the tangent to the edge portion currently being visited by the vehicle.

A similar technique can be adopted for Case 2 above, using the turning angle T_e at the old node and the last navigation direction D_f. Actually, since D_f is approximately equal to the direction D_0 of the outgoing edge, the incoming edge can be chosen as the input edge with its direction D_i closest to $D_f - T_e$, as illustrated in Fig. 9(b). Cases 3 and 4 can be solved similarly to Cases 1 and 2, respectively. The details are omitted here.

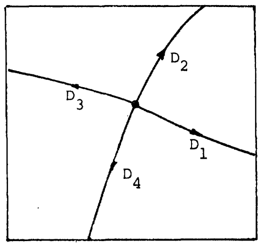


Fig. 8. Example of input window.

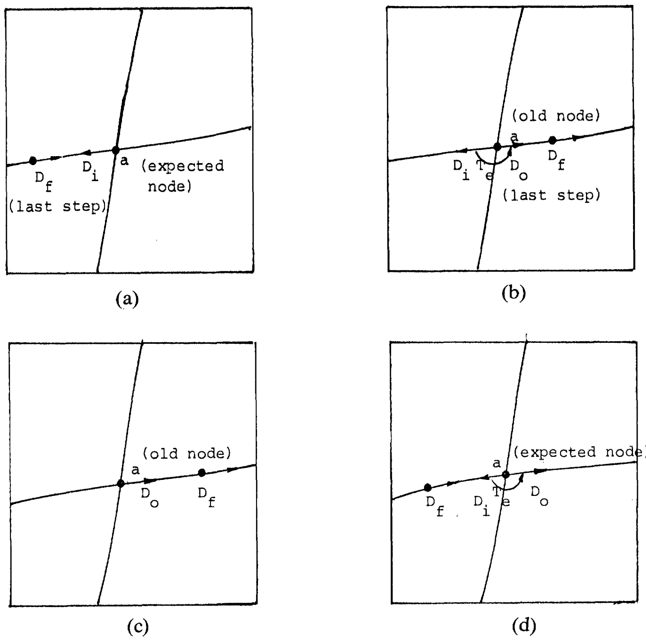


Fig. 9. Incoming/outgoing edge determination in input window. (a) *Case 1:* incoming edge is edge with its direction D_i closest to $-D_f$. (b) *Case 2:* incoming edge is edge with its direction D_i closest to $D_f - T_e$. (c) *Case 3:* outgoing edge is edge with its direction D_o closest to D_f. (d) *Case 4:* outgoing edge is edge with its direction D_o closest to $-D_f + T_e$.

B. Node Scene Matching

The procedure for node scene matching is shown in Fig. 10, which, as mentioned previously, includes two distinct types of consistency checks. The first is about the consistency of incident edge numbers, and the second about the consistency of interedge angles. To complete the second check, the system performs the following steps.

Step 1: Transform the direction list of the input node in the window data into a corresponding interedge angle list.

Step 2: Find corresponding interedge angle pairs, each pair consisting of an interedge angle of the input node and a corresponding one of the expected or the old node.

Step 3: Measure the similarity of all corresponding interedge angle pairs according to a mismatch measure, and then decide if the input node matches the reference data.

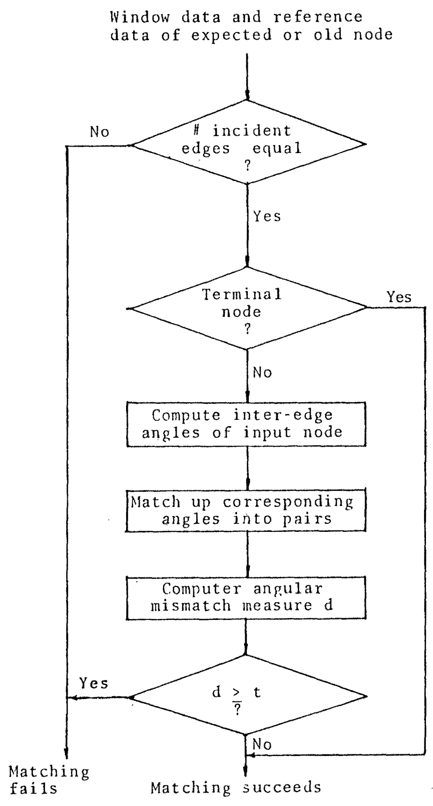

Window data and reference data of expected or old node
# incident edges equal ?
No
Yes
Terminal node ?
Yes
No
Compute inter-edge angles of input node
Match up corresponding angles into pairs
Computer angular mismatch measure d
d ≥ t ?
Yes
No
Matching fails
Matching succeeds

Fig. 10. Flowchart of node scene matching procedure.

The first step can be accomplished in the same way as used for setting up the angle lists in the reference data (see Section III-B) except that the final operation to re-index the computed angles is not executed because the incoming edge in the window is unknown yet. To perform the second step, we first determine which edge in the window is the incoming edge (see Section IV-A). Re-indexing of computed interedge angles for the input window is then performed in the same way as mentioned in Section III-B. Finally, we pair up all interedge angles in the input window with those in the reference data. This completes the second step above.

The mismatch measure d proposed for step 3 is defined as the sum of all the differences between the interedge angle pairs, or

$$d = \sum_{i=1}^{m} |A_i^r - A_i^w|$$

where A_i^r and A_i^w are two corresponding angles paired up in step 2. The consistency check of interedge angles is passed if d is less than a preselected threshold t. If the input is just a terminal node, then the consistency check of interedge angles is ignored.

C. System Initialization and System Parameters

In the system initialization procedure, an image window is first taken, path curves are then extracted, and the start point is searched. The image processing techniques used are similar to those for network processing. The incident edge of the start point is then traced and approximated as a straight line. Its direction D_0 is also computed. The final step is to initialize a set of system parameters described as follows, which are updated whenever the system states are changed.

1) *Navigation directions* D_f: Initialized as D_o.
2) *Old node* N_o: Initialized as the start point in the reference data.
3) *Expected node* N_e: Initialized as the next node to be visited.
4) *Control state* s: Initialized as LON, i.e., leaving the old node that is the start point.
5) *Navigation length on current edge* l: Initialized as 0.

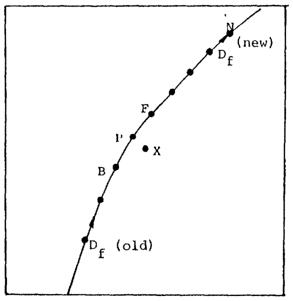


Fig. 11. Input window for illustration of MFCE response.

With the old and the expected nodes specified, the following items in the reference data can be retrieved, which should also be considered as system parameters. Assume that a denotes the starting point (i.e., the old node) and b, the expected node.

6) *Total length of the current edge* L: Initialized as the value $L(a, b)$ in the node entry of b, which is the distance for the vehicle to travel from the old node to the expected node.
7) *Turning angle* T_o *at the old node*: No initial value.
8) *Turning angle* T_e *at the expected node*: Initialized as the value $T(a, c)$ in the node entry of b, where c is the third node to be visited in the reference path.
9) *Interedge angles of the expected node*: Initialized as those contained in the angle list of b.
10) *Numbers of incident edges of the old and the expected nodes, respectively*: Initialized as the corresponding numbers contained in the node entries of a and b, respectively.

All the above parameters are used in input scene classification, especially in node scene matching, except the control state s which is used for state transition.

D. Response MFCE—Moving Further on Current Edge

MFCE is the response to the TAE (traversing along an edge) state. Shown in Fig. 11 is a typical image window "seen" on the vehicle through the camera when the vehicle is in the TAE state. The first step of the MFCE response is to find the edge point P nearest to the window center X. X should be on the current edge (i.e., P is identical to X), if the vehicle always navigates on the path without going astray laterally. But in real navigation, this is seldom the case. The remaining steps of MFCE is just to find a point N on the current edge for the vehicle to move ahead to in the next control step. N should be on the forward direction of navigation with respect to P on the current edge. To find N, two neighboring edge points B and F of P are retrieved from the image data, and their directions with respect to P is computed. The one, say F, with its direction closer to the last navigation direction D_f is selected. Now, N can be found by tracing the current edge from P through F until a point at a distance of a fixed number of points from P is reached, which is then selected as the desired point N. The direction D of N with respect to X, the vehicle center, is finally computed and used for correction of the vehicular navigation direction. Let D' be the vehicular navigation direction of the last control step. Then the error signal ΔD for vehicular direction change is just the difference between D and D'. On the other hand, the new navigation direction D_f of the control system is updated as the direction of N with respect to its preceding edge point. D_f and D should not be confused. The latter is used for guiding the navigation of the physical vehicle while the former is used for input scene classification.

E. Response TTNE—Turning to New Edge

Fig. 12 illustrates the following. Assume that a is the input node around which the edge turning is to be made. The first step is to identify the incoming edge and the outgoing edge in the input window (see Section IV-A). We can then ignore all the

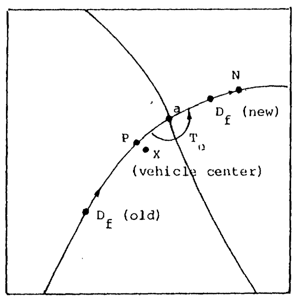


Fig. 12. Image window for illustration of TTNE response at expected node.

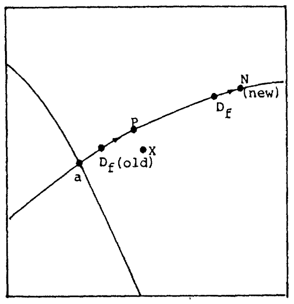


Fig. 13. Image window for illustration of MFON response at old node.

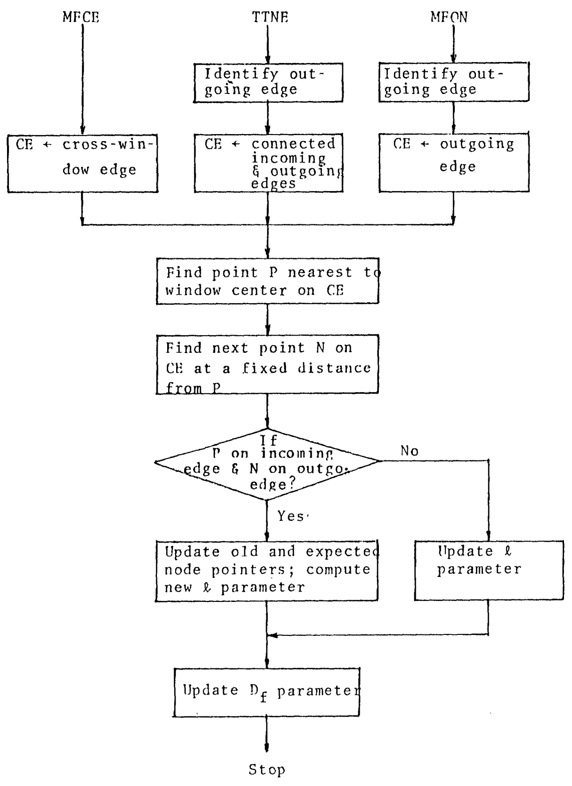

MFCE
TTNE
MFON
Identify outgoing edge
Identify outgoing edge
CE ← cross-window edge
CE ← connected incoming & outgoing edges
CE ← outgoing edge
Find point P nearest to window center on CE
Find next point N on CE at a fixed distance from P
If P on incoming edge & N on outgo. edge?
No
Yes
Update old and expected node pointers; compute new ℓ parameter
Update ℓ parameter
Update D_f parameter
Stop

Fig. 14. Steps of three types of response MFCE, TTNE, and MFON (CE = current edge).

other incident edges and connect the incoming and the outgoing edges as a single edge. What remains to do in TTNE now is identical to the response steps of MFCE described in the previous section except that the selected next point N to move to may fall on the incoming edge or on the outgoing edge.

F. Response MFON—Moving Further Near Old Node

MFON is the response to the LON state in which the old node is "seen" again in the input window, although the vehicle is already traversing along the outgoing edge. The first step is to identify the outgoing edge in the window (see Section IV-A and Fig. 13). A special case here is the first MFON response to the system initial LON state, in which the old node identified in input scene classification is just the start point with only one incident edge. In such a case, the incident edge is directly regarded as the outgoing edge. After the outgoing edge is identified and regarded as the current edge, all remaining steps are identical to those performed in the response of MFCE. We mention here that the system state is updated immediately after an input scene is classified. As a summary of the three types of response described in this and in the last sections, a flowchart of the response steps is included in Fig. 14, which emphasizes the overlapped steps in the three types of response.

V. Simulation Results and Discussions

The configuration of the hardware system which has been set up for simulation is shown in Fig. 15, including a pantilt, a television monitor, and a television camera mounted on the pantilt, all connected to and controlled by a multi-microprocessor system [14]. A hand-drawn curve-type network map is hung on a wall far away enough from the pantilt-camera set to avoid perspective effect on image taking. This configuration simulates indoor ground navigation with the network painted on a high ceiling or aero-navigation with the paths on the ground.

To facilitate pantilt control, extensive data measurement has been performed to set up a piecewise linear model that expresses the relationship between control time and pantilt move distance [18]. Based on this model, the pantilt can be moved for any desired distance after being triggered for a corresponding amount of time, via the use of a self-constructed control interface.

During the learning stage of the simulation, the camera zoom is adjusted in such a way that the whole network map can be included within the camera's field of view. Fig. 2(a) shows one of such images. During the navigation stage, the camera zoom is readjusted so that only a small portion of the map can be seen within the view. This simulates the availability of local scenes to the system during the along-path navigation.

Since the image scales of an identical scene are different under different camera zooms, the lengths computed for the image taken with one zoom must be converted into corresponding lengths for the image taken with another zoom. This is required to make scale-free matching in scene classification as well as scale-free distance computation for system parameter updating. The method used in this study is to measure the areas A_1 and A_2 of a fixed-size block square imaged with two different zooms, and use the following equality for length conversion:

$$L2 = L1 * (A_2/A_1)^{1/2}$$

which is based on the triangulation principle that the area ratio is equal to the square of the length ratio. In real applications, square-area measurement might be impractical, but a similar length conversion equation will be required for scale adjustment.

As a simulation example, with the path network as shown in Fig. 2 and the selected path as shown in Fig. 4 which are the results of the network learning stage and the reference path setup stage, respectively, the consecutive image windows taken during the navigation session are shown in Fig. 16. The plus sign at the center of each window represents the vehicle center, and the × represents the next edge point (denoted as N previously) to move toward in the next step. In Fig. 17, we mark down all the vehicle center locations with a plus sign and all the next edge points with a square on the network map. As can be seen, though the vehicle goes astray from the path edges at the end of most control steps, the selected path is followed to its end successfully. This means that image-based navigation control of the simulated vehicular movement by the pantilt is quite effective, and edge turnings supported by node scene matchings are also feasible.

In the simulation, at the end of each control step, to avoid taking blurred images, the pantilt is controlled to stop a little while for image taking. The pantilt is held still afterward until image analysis is completed and proper decision made. Each

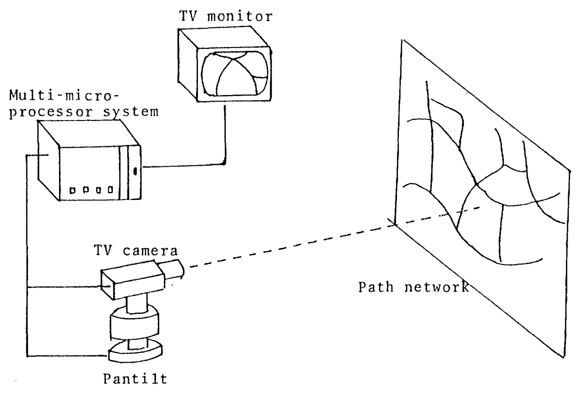


Fig. 15. Simulation system configuration.

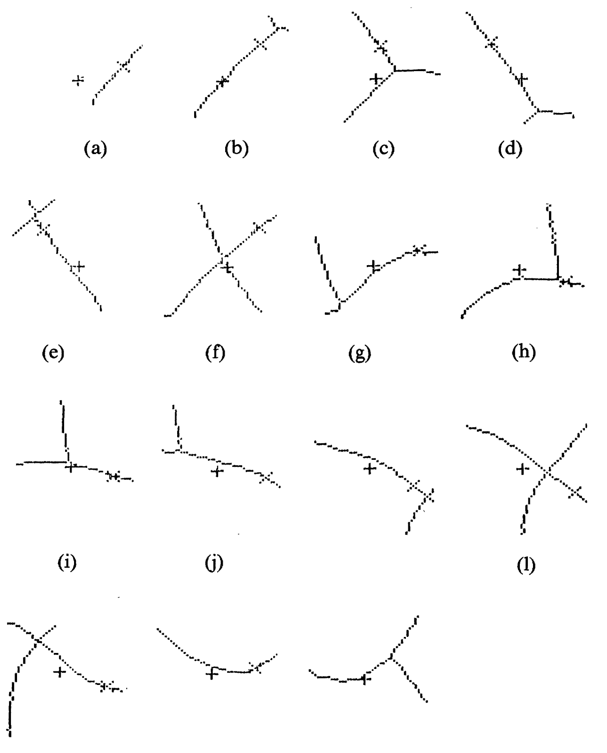


Fig. 16. Image windows of simulated navigation session along reference path shown in Fig. 4 selected from network shown in Fig. 2.

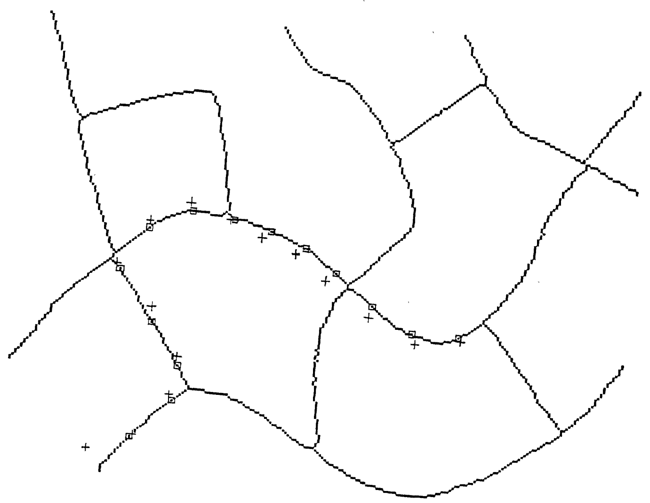


Fig. 17. Overview of simulated navigation session of Fig. 16. Squares denote next edge points, and plus signs vehicle centers.

control step takes time about three seconds. For real applications, the vehicle must be allowed to continue its movement during image taking and decision making. With no movable vehicle available for field test, no attempt has been made on estimating vehicle speed ranges and corresponding image sampling rates for real navigation. We emphasize in this research only the feasibility study on the application of image analysis techniques to automatic vehicle navigation, and this is shown possible by the simulation results as described previously.

It should be noted that several assumptions have been implicitly made in the previous discussions on the characteristics of the path network.

1) Path nodes should be spaced far enough apart so that only a single node will appear at one time in the image window.

2) The vehicle must navigate slowly enough so that no path node will be skipped during successive image taking.

3) The edges should be smooth enough (not wavy) so that measured l values can be consistent with the L value in the network database.

4) Only one edge is allowed to appear in the image window during the TAE state.

5) The angles between edges must be large enough.

Basically, the above assumptions can be satisfied more easily if the image window size is selected to be smaller (or equivalently, if the network image is taken from a smaller distance) and if the vehicle navigates more slowly.

Finally, we mention that no error recovery capability has been included in the proposed system so far; as long as the system enters an erroneous state, it simply stops. Actually, more intelligence can be added to the system to handle some unexpected situations. For example, if a non-identifiable node appears in the middle of two reference nodes, the system can simply instruct the vehicle to proceed ahead to see if the expected node can be encountered next at the expected distance from the old node. Moreover, if the vehicle gets lost without "seeing" any path curve in the window, it may try to move backward and proceed a smaller distance from the last starting position. Additional devices, such as infrared beacons, also may be used for absolute position sensing. This makes the system more robust in error recovery. If the navigation is performed in outdoor environments, certain special along-path scenes, such as rivers, bridges, buildings, mountains, etc., may also be utilized for consistency checks. It is also desirable to extend the system's learning capability so that partial network maps may be learned as the vehicle explores forward in the network.

Acknowledgment

The authors are grateful to one of the referees for his many useful comments on the paper, especially on the implicit assumptions on the path network characteristics, which led to great improvement of the paper.

References

[1] R. E. Plotkin, "Automation of the highways, an overview," *IEEE Trans. Veh. Technol.*, vol. VT-18, no. 2, pp. 77–80, Aug. 1969.

[2] R. E. Fenton *et al.*, "Fundamental studies in automatic vehicle control," Transport. Control Lab., The Ohio State Univers., Columbus, OH, Sept. 1980.

[3] ——, "Advances toward the automatic highway," *Highway Res. Record, Highway Research Board*, no. 344, pp. 1–20, 1971.

[4] H. P. Moravec, "The Stanford cart and the CMU rover," *Proc. IEEE*, vol. 71, pp. 872–884, July 1983.

[5] W. H. Cormier and R. E. Fenton, "On the steering of automated vehicles—a velocity-adaptive controller," *IEEE Trans. Veh. Technol.*, vol. VT-29, no. 4, pp. 375–385, Nov. 1980.

[6] R. E. Fenton, G. Melocik, and K. W. Olson, "On the steering of automated vehicles—Theory and experiment," *IEEE Trans. Automat. Contr.*, vol. AC-21, pp. 306–315, June 1976.

[7] S. E. Shladover, "Optimal and suboptimal steering control of rubber-tired guideway vehicles," M.S. thesis, Dept. Mech. Eng., MIT, Cambridge, MA, Feb. 1974.

[8] K. W. Olson, "Wire reference configurations in vehicle lateral control," *IEEE Trans. Veh. Technol.*, vol. VT-26, pp. 161–172, May 1977.

[9] R. J. Mayhan and R. A. Bishel, "A two-frequency radar for vehicle automatic lateral control," *IEEE Trans. Veh. Technol.*, vol. VT-31, no. 1, Feb. 1982.
[10] G. T. Clemence and G. W. Hurlbut, "The application of acoustic ranging to the automatic control of a ground vehicle," *IEEE Trans. Veh. Technol.*, vol. VT-32, no. 3, Aug. 1983.
[11] D. B. Gennery, "A stereo vision system for an autonomous vehicle," in *Proc. 15th Int. Joint Conf. Artif. Intell.*, Aug. 1977.
[12] H. P. Moravec, "Towards automatic visual obstacle avoidance," in *Proc. 15th Int. Joint Conf. Artif. Intell.*, Aug. 1977.
[13] S. Tsugawa *et al.*, "An automobile with artificial intelligence," in *Proc. 15th Int. Joint Conf. Artif. Intell.*, Aug. 1977.
[14] W. H. Tsai *et al.*, "Architecture of a multi microprocessor system for parallel processing of image sequences," in *Proc. IEEE Computer Soc. Workshop Computer Architecture for Patt. Anal. Image Database Management*, 1981, pp. 104–111.
[15] A. Rosenfeld and A. C.. Kak, *Digital Picture Processing*, vol. II. New York: Academic, 1982.
[16] T. L. Booth, *Finite Automata and Sequential Machines*. New York: Wiley, 1967.
[17] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*. New York: Wiley, 1973.
[18] Y. C. Chen and W. H. Tsai, "Design and simulation of an image-based system for automatic conveyance guidance," Tech. Rep., National Chiao Tung University, Institute of Computer Eng., Hsinchu, Taiwan, R.O.C., June 1983 (in Chinese).
[19] A. Rosenfeld and A. C. Kak, *Digital Picture Processing*, vol. I. New York: Academic, 1982.

A New Algorithm for Graph Monomorphism Based on the Projections of the Product Graph

FOLORUNSO A. AKINNIYI, ANDREW K. C. WONG, MEMBER, IEEE, AND DEBORAH A. STACEY

Abstract—**A new algorithm is presented for detecting graph monomorphisms for a pair of graphs. This algorithm entails a tree search based on the projections of the product graph called the net of the two graphs. It uses the minimum number of neighbors of the projected graphs to detect infeasible subtrees. The algorithm, in comparison with that of Deo and coworkers, is more efficient in its storage space utilization and average execution time. It does not suffer from the ambiguity which arises in Deo *et al.*'s work when cyclic graphs are matched. Applications to attributed graph monomorphisms are included.**

I. Introduction

The graph monomorphism problem, also known as the subgraph isomorphism problem, is one which finds out whether or not a one-to-one vertex mapping between two graphs that preserve incidence relations exists. The graphs are assumed to be finite. The formalism and terminology used are based on Berge [3] and Ghahraman *et al.* [11].

Graph monomorphism has wide applications in areas such as scene analysis, information storage and retrieval, chemical documentation, and network analysis. In scene analysis, a graph morphism task is to compare an image to a reference scene for purposes of recognition and/or classification. Ideally, the graph morphism problem is that of isomorphism. However, if occlusion occurs, graph monomorphism or the largest common subgraph isomorphism is more applicable. Even the well-known traveling salesman problem can be formulated as one of finding an optimal cost graph monomorphism [11] in which each itinerary is mapped onto a transportation network to achieve the best itinerary.

It is well-known that the graph monomorphism problem belongs to the class of NP-complete problems [10]. Problems in this class are inherently intractable; that is, any algorithm designed to solve the general problem will require exponential time complexity. Over the past two decades, considerable efforts have been devoted to the problems related to graph isomorphism. A comprehensive survey of the major works can be found in [16]. Unfortunately, hitherto, no polynomial time algorithm has been found. Attempts to solve the problem have been largely concentrated in four areas:

1) The first area is the design of polynomial time algorithms for various special cases of the problem. For instance, the tree isomorphism algorithm which runs in $O(n)$ (see [1, pp. 84–86]). Also, there is the $O(n \log n)$ algorithm of Hopcroft and Tarjan [14] for detecting isomorphism of a pair of planar graphs.

2) The second area is the design of polynomial time heuristics which iteratively partition the vertices of the two graphs into classes and then refine the partitioned classes until a one-to-one vertex mapping is (or is not) achieved. Unfortunately, these heuristics fail for graphs of high order. The most celebrated example is that of Corneil and Gotlieb [8]. Another is that of Sussenguth [19] which extends Ungers' isomorphism algorithm to include graph monomorphism.

3) The third area is theoretical works such as finding the mathematical functions [5, 18, 21] which are invariant for isomorphism and proving that certain classes of the isomorphism problem are polynomially equivalent [6]. That is, such problems have the same degree of difficulty as the general isomorphism problem.

4) The final area is the use of backtracking for exhaustive search. Algorithms in this category utilize the refinement techniques based on the neighbors of the two graphs to prune the search tree. Notable procedures are the depth first search algorithm of Deo *et al.* [9]; the distance matrix algorithm of Schmidt [17] which suffers from its inapplicability to graph monomorphism; Ullmann's subgraph isomorphism [22]; and that of Cheng *et al.* [6] which utilizes Berztiss' [4] elementary k-formula and Ullmann's tree search to develop a pseudo-parallel algorithm for subgraph isomorphism.

Recently, Ghahraman *et al.* [11] proposed a backtracking algorithm based on the product (called the net) of two graphs, namely, the pattern graph which represents the domain of the morphism, and the base graph representing the range. According to this algorithm, a subgraph known as the star pattern subnet (SPS) is formed for each vertex in the net. The feasibility of each vertex is then determined by solving the problem of maximum matching in a bipartite graph (MMBG) which is associated with the SPS of that vertex. By investigating the existence of SPS's whose projections coincide with the pattern graph, the existence of a monomorphism can be ascertained. The utilization of the MMBG criterion (called the strong necessary condition in [11]) leads to an effective and fast pruning of the search tree with the result that monomorphisms are found near the root. However, the iterative solution of the MMBG problem does not enhance the overall efficiency of the algorithm. The best algorithm still requires $O(n^{5/2})$ time complexity [15], where n is the order of the graph.

In this correspondence, a new algorithm, reminiscent of the net formulation, is proposed. Unlike that of [11], the new algorithm does not require the explicit derivation of the SPS nor the solution of the MMBG problem. Rather, the weighted projections of the SPS (called the weighted edge subgraph (WES)) and the minimum number of neighbors (MINN) of the resulting projected star subgraph are used in conducting the tree search.

Manuscript received October 10, 1983; revised April 30, 1986.
F. Akinniyi was with the Systems Design Engineering Department, University of Waterloo, Waterloo, ON, Canada. He is now with the Federal Agricultural Coordinating Unit, Ibadan, Nigeria, on leave from the University of Lagos, Lagos, Nigeria.
A.K.C.Wong and D.A.Stacey are with the Systems Design Engineering Department, University of Waterloo, ON, Canada, N2L 3G1.
IEEE Log Number 8609838.