PERGAMON

Robotics and Computer Integrated Manufacturing 15 (1999) 353–364

Robotics and Computer Integrated Manufacturing

www.elsevier.com/locate/rcim

A new approach to vision-based unsupervised learning of unexplored indoor environment for autonomous land vehicle navigation☆

Guan-Yu Chen, Wen-Hsiang Tsai*

Department of Computer and Information Science, National Chiao Tung University, 1001 Ta-Hsuch Road, Hsinchu, Taiwan 300, Republic of China

Abstract

A vision-based approach to unsupervised learning of the indoor environment for autonomous land vehicle (ALV) navigation is proposed. The ALV may, without human's involvement, self-navigate systematically in an unexplored closed environment, collect the information of the environment features, and then build a top-view map of the environment for later planned navigation or other applications. The learning system consists of three subsystems: a feature location subsystem, a model management subsystem, and an environment exploration subsystem. The feature location subsystem processes input images, and calculates the locations of the local features and the ALV by model matching techniques. To facilitate feature collection, two laser markers are mounted on the vehicle which project laser light on the corridor walls to form easily detectable line and corner features. The model management subsystem attaches the local model into a global one by merging matched corner pairs as well as line segment pairs. The environment exploration subsystem guides the ALV to explore the entire navigation environment by using the information of the learned model and the current ALV location. The guidance scheme is based on the use of a pushdown transducer derived from automata theory. A prototype learning system was implemented on a real vehicle, and simulations and experimental results in real environments show the feasibility of the proposed approach.

Keywords: Unsupervised learning; Autonomous land vehicle navigation; Computer vision; Model matching; Pushdown transducer; Environment exploration

1. Introduction

Applications of vision-based autonomous land vehicles (ALV's) were intensively studied in recent years. In many ALV applications, model-based guidance approaches are often employed for ALV navigation. However, the traditional method of establishing environment models — manual measurement of the navigation environment — is a time-consuming work. It is thus desired to design a system for automatic learning of navigation environments. Several environment learning systems were developed in the recent years [1–9] to meet this requirement. Many other mobile robot learning systems were also constructed [11–14] for use in applications other than navigation environment modeling, such as navigation, goal reaching, obstacle avoidance, etc.

For environment learning, Lebègue and Aggarwal [1,2] developed an integrated system to generate architectural CAD models using a mobile robot. The system consists of a segment detector, a tracker, and a CAD modeler. A basic assumption of their study is that the navigation environment is with prominent 3D orientations. Such an assumption stands in most building corridors. Nashashibi et al. [3] proposed an approach to building a rough geometric model for a 3D terrain using a laser range finder. They also gave algorithms to build snapshot models with planar faces from range data. By performing 3D data fusion between the snapshot models, the proposed approach can build a reliable 3D model incrementally [4]. Ishiguro et al. [5] presented a strategy for establishing the model of an unknown environment by a mobile robot. Panoramic sensing was used to perceive the structure of the environment in their implementation. Kurz [6] introduced an approach to generating environmental maps based on ultrasonic range data. Free-space can be partitioned into situation areas by means of a learning classifier. Then the situation areas can be attached to graph nodes by dead-reckoning and finally a map of the free-space in the form of a graph

☆This paper is supported under the project NSC-87-2213-E-009-114 of National Science Council, the Republic of China.

*Corresponding author. Tel.: +886(3)-571-2121 ext 56621; fax: 886(3)-572-1490.

E-mail address: whtsai@cis.nctu.edu.tw (W.H. Tsai)

0736-5845/99/$ - see front matter
PII: S0736-5845(99)00033-2

representation is generated. Dean et al. [7] formulated map learning as a problem of inferring the structure of a reduced deterministic finite automaton from noisy observations and also provided an exploration algorithm to learn the correct structure of the automaton. Pan and Tsai [8] proposed an integrated approach to automatic model learning and path generation for vision-based ALV guidance in building corridors. In Chen and Tsai [9], an incremental environment learning system for ALV navigation was proposed. Rough initial environment models are first constructed, and after each navigation session, the proposed system can update the environment model according to the information collected in the previous navigation.

In most of the above-mentioned environment learning systems, certain involvement from human operators is required in the learning process. For example, in our previous work [9], the ALV should be driven manually by a human operator around the environment for initial learning. However, for some applications, it is impractical to get human's involvement. A typical example of such applications is the use of an autonomous mobile robot to work in a nuclear plant or other dangerous regions. For these applications, all operations, including the learning process, should be fully automatic. As a result, the capability to explore an unknown navigation environment automatically is required. An unsupervised environment learning system for this purpose is proposed in this study.

A major problem in unsupervised learning of navigation environments is systematic traverse of an environment. It is desired that once an ALV is started for self-navigation, the ALV will visit every spot in the environment systematically and return to the start location. The ALV should not be stuck in any dead-end, nor should it fall into navigation loops. It needs certain theory to design a guidance engine to achieve this goal. It is found in this study that automata theory provides a good solution. A pushdown transducer, called *navigation transducer*, was designed in this study to guide the ALV to explore unknown environments with no supervision. The sensed local environment features are encoded into symbols for use as input to the navigation transducer by a preprocessing unit. And output symbols, which represent ALV actions, are generated by the navigation transducer to guide the ALV to traverse the environment systematically.

Furthermore, the proposed system is aimed for use in real environments. Accordingly, many practical problems need be solved, e.g., how to locate the environment feature, how to deal with the uncertainty caused by noise in feature detection, how to encode the local environment features into symbols for use as input to the navigation transducer, etc. All these questions are answered in this study. Simulations and convincing experimental results show the feasibility of the proposed approach.

In short, major contributions of this study include the design of the navigation transducer, which is based on automata theory and serves as an ALV guidance engine for unsupervised and systematic environment exploration, as well as the provision of effective algorithms for solving practical problems encountered in real environment applications.

The remainder of this paper is organized as follows. The principles of the proposed learning system are described in Section 2. In Section 3, the detailed procedures for feature location and model updating are described. In Section 4, the detailed procedures for automatic environment exploration based on the use of the proposed navigation transducer are described. In Section 5, several simulation and experimental results are presented. Finally, some conclusions and a few suggestions for further works are given in Section 6.

2. Principles of proposed learning system

In the proposed system, computer vision techniques are employed to locate indoor environment features. The selected environment features are the obstacles detected in the sensed images. In indoor corridor environments, obstacles are usually the walls and the doors in buildings. In the proposed system, two laser markers are employed to help identifying the locations of obstacles. The laser markers are properly installed so that their projection marks on the obstacles in the environment are all at the same height. In a typical scene of the experimental environment, as shown in Fig. 1, the laser markers project bright laser light onto the obstacles in the environment, which forms line segments and corners in sensed images.

Fig. 1. A typical scene of the experimental environment.

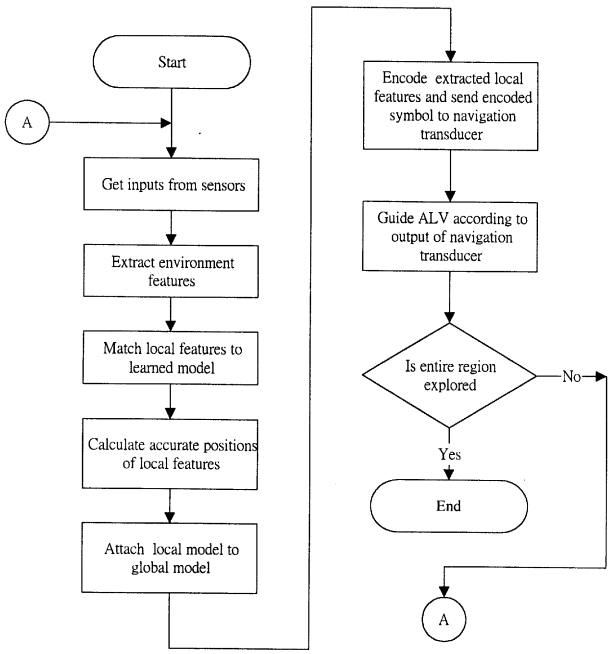

Fig. 2. Flowchart of proposed learning system.

The line segments are first found by image processing. Then by computer vision techniques, the location of the line segment features are calculated. The corresponding features of these obstacles appear to be line segments in the top view. A line-segment model matching algorithm is used to find the correspondence between the sensed local model and the learned global model. The matching result then are used to locate the ALV and construct environment models. Furthermore, for automatic environment exploration, a navigation transducer is employed to guide the ALV in the proposed learning system, and a preprocessing unit is used to encode local environment features into symbols for use as input to the navigation transducer. Each output symbol of the navigation transducer corresponds to a certain ALV operation. A path planning unit then interprets the output symbol and generates a series of low-level ALV control commands to guide the ALV to explore the entire navigation environment. The entire learning system thus can be divided into three subsystems: a feature location subsystem, a model management subsystem and an environment exploration subsystem. A flowchart of the proposed learning procedure is shown in Fig. 2. A detailed description of the learning algorithm is described as follows.

Algorithm 1. Unsupervised learning of unexplored environment for ALV navigation

Step 1. Perform camera and laser marker calibration.
Step 2. Drive the ALV manually to an initial location and start the ALV.
Step 3. Set the initial global model as empty.
Step 4. Capture an image from the camera.
Step 5. Extract environment features from the captured image.
Step 6. Calculate the location of the extracted environment features and set up a local model by collecting the extracted local features (see Section 3.1 for the detail).
Step 7. If the global model is non-empty, then match the local model with the global model and recalculate the accurate position of the local features by the matching result (see Sections 3.1 and 3.2 for the detail).
Step 8. Attach the local model to the global model (see Section 3.3 for the detail).
Step 9. Encode the extracted local features and send the encoded symbol to the navigation transducer as input (see Section 4 for the detail).
Step 10. If the output of the navigation transducer is 'stop', then stop the ALV; otherwise, perform the ALV operation corresponding to the transducer output, and go to Step 4 to start another cycle.

The proposed ALV learning system consists of three subsystems, a feature location subsystem, a model management subsystem, and an environment exploration subsystem. The feature location subsystem is designed to extract and locate the environment features. The model management subsystem builds and keeps track of the learned global model. The environment exploration subsystem consists of a navigation transducer, a preprocessing unit and a path-planning unit. The preprocessing unit encodes the extracted local environment features into symbols for use as input to the navigation transducer. The navigation transducer serves as the guidance kernel, which leads the ALV to explore the entire navigation environment automatically and systematically. The path planning unit interprets the output symbols of the navigation transducer and generates a series of low-level ALV commands. The interaction of these subsystems are shown in Fig. 3.

There is a basic assumption for the experimental environment: the obstacles, namely, the walls and doors, in the environment are all in two orthogonal directions. With this assumption, the environment features can be treated as a set of orthogonal line segments. In comparison with the matching algorithms for arbitrary line segment patterns, the matching algorithms for orthogonal line segment patterns are simpler and faster. In this study, an orthogonal-line-segment- pattern matching algorithm for locating the ALV and environment features is developed. Furthermore, with this assumption, the design of the preprocessing unit and the navigation transducer could also be simplified. In real building, this assumption is true in most indoor space.

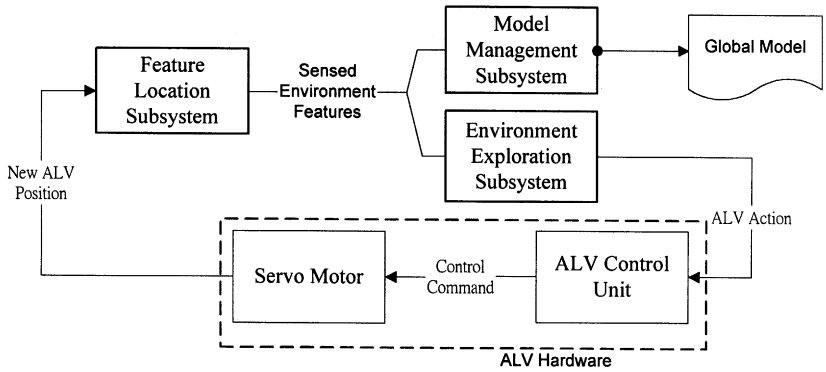

Fig. 3. Interaction between three environment learning subsystems.

3. Strategies for feature location and model updating

3.1. Ideas of locating environment features

With no depth information, a single image is insufficient for locating any line segment with no heuristic. However, if an end point of a line segment is located on a known plane with known height, the location of the end point may be uniquely decided. In this study, the laser markers are first calibrated in such a way that all their projection marks appear at the same height, say, h_0. As a result, all line segments are known to be on the plane $z' = h_0$. The equations for calculating the location of a point located on a known plane in the vehicle coordinate system can be found in [15].

If the vehicle location is known, the location of a line in the global coordinate system can also be obtained (as shown in Appendix A). In our approach, a rough estimation of the vehicle location is first obtained by the use of an odometer equipped in the ALV, which provides the navigation distance during a cycle, as well as the use of a photo-encoder, which tells the turn angle of the front wheels. Then by the use of the result of matching the collected line segment features with those in the learned global model, the error in the rough estimation of the vehicle location can be corrected, and so safe ALV guidance is feasible. Then the accurate location of the local features can be recalculated according to the corrected ALV location.

The estimated position and orientation of the ALV are calculated as follows in our approach. When the ALV moves away from a known position, the new position of the ALV can be estimated by using the moving distance S and the turn angle of the front wheels. The derivations of the equations to calculate the estimated ALV location can be found in [8] and are reviewed in the following. As shown in Fig. 4, assume that the vehicle is located at A. After moving a distance S forward, the vehicle will be at a new location B, which is the desired estimated ALV location. Let the relative location of B with respect to

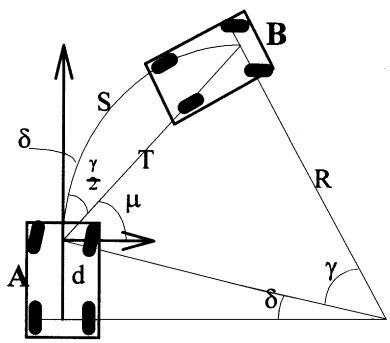

Fig. 4. The vehicle location before and after the ALV moves a distance S forward.

A be denoted by a vector $\mathbf{T}$. The rotation radius R can be written as:

$$R = d/\sin\delta, \tag{1}$$

where d is the distance between the front wheels and the rear wheels, and δ is the turn angle of the front wheels. And the angle γ can be determined as

$$\gamma = S/R. \tag{2}$$

So, the length of vector $\mathbf{T}$ can be solved to be

$$\|\mathbf{T}\| = R\sqrt{2(1-\cos\gamma)}, \tag{3}$$

and the direction of vector $\mathbf{T}$ is

$$u = \frac{\pi}{2} - \delta - \frac{\gamma}{2}. \tag{4}$$

The coordinates of location B in the vehicle coordinate system with respect to location A can thus be computed by

$$x_B = \|\mathbf{T}\|\cos u, \quad y_B = \|\mathbf{T}\|\sin u. \tag{5}$$

After the front wheel location of the ALV is determined, the rear wheel location $(\tilde{x}_B, \tilde{y}_B)$ of the ALV can also be determined to be

$$\tilde{x}_B = x_B + d\sin\gamma, \quad \tilde{y}_B = y_B - d\cos\gamma. \tag{6}$$

Since the global coordinates of location A are known, and since the vehicle coordinates of location B with respect to location A can be obtained from Eq. (5), the global coordinates of location B can be calculated by coordinate system transformations. Thus the desired estimated ALV location is obtained.

After performing the line segments matching algorithm, the displacement (x_t, y_t, γ) from the local model to the learned global model can be obtained. Note that this displacement is also the displacement from the estimated vehicle location to the actual one. As a result, the actual vehicle location (x'_p, y'_p, ω) can be obtained by:

$$x'_p = \hat{x}_p + x_t, \quad y'_p = \hat{y}_p + y_t, \quad \omega = \hat{\omega} - \gamma, \tag{7}$$

where $(\hat{x}_p, \hat{y}_p, \hat{\omega})$ is the estimated ALV location, and $(x_t, y_t, \gamma)$is the displacement vector obtained from the matching result.

3.2. Algorithm for model matching

There are several algorithms for line segment pattern matching. For example, the Generalized Hough Transform [16] (GHT) is a popular approach to arbitrary pattern matching. It also works for line segment pattern matching, which is desired in this study. However, if the dimension of the counting space for the GHT is large due to the large sizes of the matched patterns, a lot of memory space and a significant computing time will be required to perform the algorithm. Thus it is desired to develop a faster and simpler matching method for real-time applications. Another problem arises when computer vision inaccuracy and image processing errors are involved. This makes perfect matching impossible, and a "fuzzy" matching algorithm is required. Furthermore, sometimes a newly detected line segment feature in the local model may not exist in the learned global model, and so the algorithm should also be capable of partial matching.

In this study, a new line segment matching algorithm is proposed to meet the above three requirements. Since all line segment features in both the local model and the extracted global model are parallel to the x'-axis or the y'-axis, there is no rotation between the two models. The proposed matching algorithm is designed to find only the translation from the local model to the extracted global model.

The proposed matching algorithm is based on the following idea. A line segment L_x parallel to the x'-axis can be described with a three-tuple (y_c, x_1, x_2), where $y' = y_c$ is the line equation of L_x, and (x_1, y_c) and (x_2, y_c) are the two end points of L_x. Similarly, a line segment L_y parallel to the y'-axis can be described with a three-tuple (x_c, y_1, y_2), where $x' = x_c$ is the line equation of L_y, and (x_c, y_1) and (x_c, y_2) are the two end points of L_y. Such descriptions may be interpreted to be obtained through the following operations: for a line segment L_x parallel to the x'-axis, mark a symbol on the y'-axis (at $y' = y_c$) to indicate where the line is located and mark two symbols on the x'-axis (at $x' = x_1$ and $x' = x_2$, respectively) to indicate where the two end points are located. For line segments parallel to the y'-axis, similar operations can be performed. For each line segment in a pattern (or a model), the above operations are performed. In this way, a set of symbols on the x'-axis and another set on the y'-axis are obtained. The locations of the marked symbols can be used to describe the locations of the line segments in the pattern. Therefore, by matching the locations of the marked symbols of two patterns, the displacements between the two patterns can be found.

The proposed matching algorithm is based on the above idea. In the proposed matching algorithm, the marked symbols are quantified, so a set of the marked symbols can be regarded as the range of a function either on x'-axis or on y'-axis. These functions are defined to be the *signatures* of a line-segment pattern. In this representation, the matching work is performed in the following strategy. The signatures of both the input and the reference patterns are obtained first. Then, a set of correlation values is calculated in terms of the signatures by correlating the two patterns with different displacements. Since higher correlation implies higher similarity, the displacement corresponding to the maximum correlation value is therefore the exact displacement between the two patterns. In this way, the translations between the two patterns along both the x'-axis and the y'-axis can be obtained separately. In the discrete case, the domains of the signature functions may be taken to be cells which are said to form signature spaces. The steps of the matching algorithm are formalized as follows.

Algorithm 2. Matching orthogonal line segment patterns

Input: An input line segment pattern **N** and a reference line segment pattern **L**.

Output: The displacement vector (x_t, y_t) which transforms **N** to **L** through translation.

Steps:

Step 1. Set up the x'-signature and y'-signature spaces for **N** and **L** and set the values in the cells of the signature spaces to be zero. Denote the x'-signature and the y'-signature of **N** to be S_x^N and S_y^N, respectively, and the x'-signature and the y'-signature of L to be S_x^L and S_y^L, respectively.

Step 2. For a line segment $\mathbf{n}_i$ in **N**,

if $\mathbf{n}_i$ is parallel to the x'-axis and described with three-tuple (y_c, x_1, x_2),

set $S_x^N(x_1) = S_x^N(x_1) + c$, $S_x^N(x_2) = S_x^N(x_2) + c$

and

$S_y^N(y_c) = S_y^N(y_c) + l_i$,

where (x_1, y_c) and (x_2, y_c) are the two end points of $\mathbf{n}_i$, l_i is the length of $\mathbf{n}_i$, and c is a constant;

if $\mathbf{n}_i$ is parallel to the y'-axis and described with three-tuple (x_c, y_1, y_2),

$$\text{set } S_y^N(y_1) = S_y^N(y_1) + c,\ S_y^N(y_2) = S_y^N(y_2) + c,$$

and

$$S_x^N(x_c) = S_x^N(x_c) + l_i,$$

where (x_c, y_1) and (x_c, y_2) are the two end points of $\mathbf{n}_i$, l_i is the length of $\mathbf{n}_i$, and c is a constant.

Step 3. Apply Step 2 to all line segments in **N**, and apply similar operations to line segments in **L**.

Step 4. Adjust the signature value at each x with fuzzy tolerance within a neighborhood of x, i.e., set

$$\hat{S}_x^N(x) = \sum_{i=-k/2}^{k/2}\left(S_x^N(x+i)\cdot\left(\frac{1}{1+i/d}\right)\right),$$

where k is the size of the neighborhood and d is a constant. Apply similar operations to $S_y^N(y)$, $S_x^L(x)$, and $S_y^L(y)$, respectively.

Step 5. Calculate the following correlation values:

$$C_x(x_d) = \sum_x (\hat{S}_x^N(x)\cdot\hat{S}_x^L(x+x_d)), \text{ and}$$

$$C_y(y_d) = \sum_y (\hat{S}_y^N(y)\cdot\hat{S}_y^L(y+y_d)).$$

Step 6. Find the maximum of the correlation values, and set the desired values of x_t and y_t to be the corresponding displacements, i.e., find x_t and y_t such that $C_x(x_t) = \max_x C_x(x)$ and that $C_y(y_t) = \max_y C_y(y)$.

By observing the formula of the correlation, a higher value of the signature function will play a relatively more important role in the correlation. It is thus desired to obtain a relatively more reliable feature with a higher value of the signature function. Since errors in image processing may result in some short line segments, longer line segments are generally more reliable. As a result, in the proposed matching algorithm, the value corresponding to a certain 'marked symbol' is set to be the length of the corresponding line segment. Furthermore, in the implementation of our learning system, the constant c is replaced with a value that is proportional to the times of occurrence of the corresponding end point. This implies that, like a longer line segment, a multi-occurrence end point is also relatively more important in the matching processes. These two ways of 'weighting' in matching make the proposed algorithm more effective.

An example can be used to illustrate this algorithm, as shown in Fig. 5. In the left-upper corner of Fig. 5, a reference model pattern and an input model pattern are represented as black and gray line segments, respectively. In the right-upper corner of Fig. 5, the result y'-axis signature of the reference model and that of the input model are represented by black and gray graphs, respectively. In the left-bottom corner of Fig. 5, the x'-axis signature of the reference model and that of the input model are represented by black and gray graphs, respectively. Note that every higher peak in the signature corresponds to a longer line segment, while the lower peaks correspond to shorter line segments or end points.

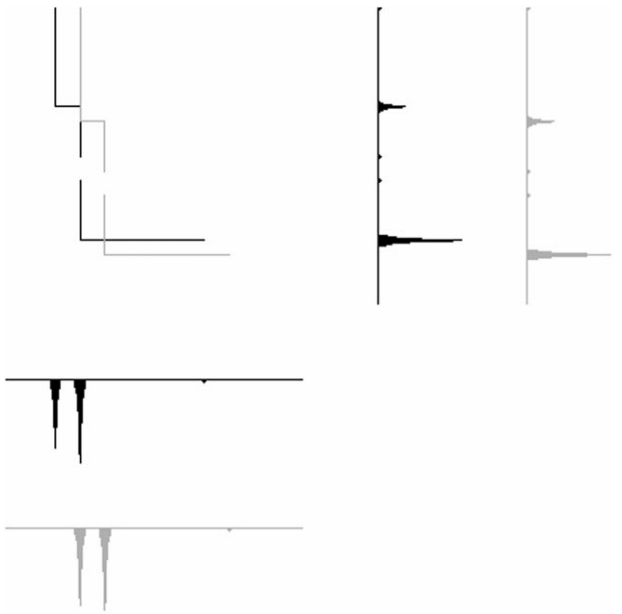

Fig. 5. Illustration of the proposed line segment algorithm.

3.3. Model updating

In the proposed learning system, the environment model was established by attaching the local model sensed in each learning cycle into the learned global model incrementally. Since all line segments are all in two orthogonal directions, there is no rotation between the local model and the learned global model. The translation between the local model and the learned global model can be found by the above proposed line-segment matching algorithm. Then the local model is translated in accordance with the result of the matching algorithm, and attached into the global model. If one line segment feature in the local model is nearly coincident with one in the global learned model, these two features are regarded as an identical one. If one line segment feature in the local model overlaps partially with one in the global learned model, these two features are merged into a larger one. This simplifies the learned global model and keeps the number of line segment features within a reasonable range. Furthermore, the model-updating algorithm also checks multi-occurrences of end points. An end point with multi-occurrences, i.e., appears more than once, is more reliable and plays a more important role in the

matching algorithm. The detailed algorithm for model updating is described as follows.

Algorithm 3. Model updating

Input: A local model **N** and a learned global model **L**.
Output: A new learned global model **M**, which is a combination of **N** and **L**.
Steps:
Step 1. Extract features from **L** within a certain window and form a new model **L**′.
Step 2. Add all line segments in **L** but not in **L**′ to **M**.
Step 3. For a line segment $\mathbf{n}_i$ in **N**:
If $\mathbf{n}_i$ is parallel to the x'-axis and described with (y_c, x_1, x_2), then
(1) for every line segment $\mathbf{l}_j$ in **L**′ parallel to the x'-axis and described with (y'_c, x'_1, x'_2):
Case 1 (Fully overlapping line segments): if $|y'_c - y_c| < c_1$, $|x'_1 - x_1| < c_2$, and $|x'_2 - x_2| < c_3$, add a line segment with two end points at $((x_1 + x'_1)/2, (y_c + y'_c)/2)$ and $((x_2 + x'_2)/2, (y_c + y'_c)/2)$ to **L**′ and remove $\mathbf{l}_j$ from **L**′;
Case 2 (Partially overlapping line segments): if $|y'_c - y_c| < c_1$, $x_2 > x'_1$ and $x'_2 > x_1$, add a line segment with two end points at $(\min(x_1, x'_1), (y_c + y'_c)/2)$ and $(\max(x_2, x'_2), (y_c + y'_c)/2)$ to **L**′ and remove $\mathbf{l}_j$ from **L**′;
(2) if there is no line segment which fully or partially overlaps with $\mathbf{n}_i$ in **L**′, add $\mathbf{n}_i$ to **L**′, where (x_1, y_c) and (x_2, y_c) are the two end points of $\mathbf{n}_i$, $x_2 > x_1$, (x'_1, y'_c) and (x'_2, y'_c) are the two end points of $\mathbf{l}_j$, $x'_2 > x'_1$, and c_1, c_2, c_3 are three predefined constants.
If $\mathbf{n}_i$ is parallel to the y'-axis, similar operations are performed.
Step 4: Add all line segments in **L**′ to **M**.
Step 5: Set the initial occurrence counts of the end points of every new line segment (i.e., line segments in L′) to be zero.
Step 6: For an end point p_i of the line segments in **M**, check if there exists any end point in a certain neighborhood of p_i in **N**. For each point in the neighborhood, increase the occurrence count of p_i by one. The occurrence counts of end points are used to adjust the weighting value in the proposed matching algorithm (see Section 3.2 for the details).

4. Strategies for automatic exploration without supervision

For automatic environment exploration, a learning system should be able to plan the exploration path automatically and systematically with the information of surrounding environment. In the proposed learning system, this work is mainly solved by the proposed navigation transducer. The navigation transducer takes encoded symbols corresponding to surrounding environment as inputs. According to the information of input symbols, internal states and stack status, the transducer generates output symbols corresponding to special ALV actions. To encode sensed environment features into input symbols for the navigation transducer, a preprocessing unit is proposed to do the translation works. To perform the actions corresponding to the outputs of the transducer, a path planning unit is proposed to interpret the output symbols into a series of low-level ALV control commands and control the ALV to move along the planned paths.

4.1. Proposed navigation transducer

A transducer in automata theory [17] is an automaton, whose model consists of a control unit, a stack, an input unit and an output unit. It has the function of transforming input symbols into output symbols according to its state transition rules. The stack is a memory for storing symbols used in intermediate transitions.

In our previous work [10], a navigation transducer working in the simulated grid environment was proposed. Some modifications were made for working in the real environment. The proposed navigation transducer is designed to simulate the human's behavior to walk systematically and thoroughly in a maze building. Some common rules used in this situation are described as follows. Firstly, walk along a corridor if no crossing is encountered. Secondly, when a crossing is encountered, i.e., when more than two candidate paths can be chosen, select one of them and put a mark on the chosen path to distinguish it from the unexplored ones, and then go along the chosen path. Thirdly, when an end of the current path is encountered, go backward along the current path to the previous crossing, pick another unexplored path, put a mark on the selected path, and go. Finally, if all available paths for a crossing are explored, go backward furthermore to another crossing until an unexplored path is found. In this way, one may either find the exit of the building or explore the entire building and find there is no exit.

According to the rules, the navigation transducer was designed to have four categories of input symbols, *one-way corridor*, *unexplored crossing*, *explored crossing*, and *end of corridor*, which correspond to four different surrounding environment conditions. Detailed descriptions of the process for translating the sensed surrounding environment into the input symbol is included in Section 4.2. On the other hand, it is observed in this study that seven output symbols are sufficient for modeling the ALV navigation actions in building explorations: *straight forward*, *forward right turn*, *forward left turn*, *straight backward*, *backward right turn*, *backward left turn*, and *stop the ALV*. They are taken as the possible outputs of the

navigation transducer. The strategy to perform the ALV actions corresponding to the output symbols is illustrated in Section 4.3.

The stack of the navigation transducer is used to keep track of the marks, each of which indicates whether a certain path is explored or not. When one crossing is encountered, the symbols representing all unexplored paths from this crossing are pushed down to the stack of the transducer except the selected one. The navigation transducer then guides the ALV to move along the selected path. When the ALV encounters an end of the corridor, according to the third rule, the navigation transducer guides the ALV to retreat backward to the previous crossing; when arriving the previous crossing, a new path is popped out of the stack, and the navigation transducer guides the ALV to follow this new path. Besides, when all paths from the crossing have been explored, a similar retreating procedure is performed again to guide the ALV to the previous crossing. Unfortunately, if no further information is provided, the navigation transducer will not be able to know from which path the ALV comes to this crossing, and the retreating work is so infeasible. To solve this problem, the retreating path should be pushed before other unexplored paths are pushed. By the nature of stack operations, namely, first in last out, the retreating path will be popped out only after all unexplored paths are popped out. That is, the ALV retreats from one crossing only if all paths from the crossing have been explored. This ensures the ALV to explore the entire region systematically.

The stack operations of the navigation transducer are summarized as follows.

- In the *exploring mode* indicating that the ALV is navigating in an unexplored region, if a crossing is encountered, choose a path to go, push the retreating path into the stack first and then those unexplored paths except the chosen one, and finally guide the ALV to move along the chosen path.
- In the *retreating mode* indicating that the ALV has visited a path end and is retreating to the previous crossing, if the ALV reaches the crossing, pop one path from the stack, and then guide the ALV to go along the popped path.

The state of the navigation transducer is used to check whether the ALV is navigating in the exploring mode or in the retreating mode. The state transition function of the control unit maps the input, the current state, and the current stack status into a new transducer state and a new stack status. The output function maps the input, the current state, and the current stack status into an output symbol. For a complicated transducer, there may be hundreds or thousands of transition rules and output mapping rules, so it is unfeasible to list all the rules here. Instead, only several principles for illustrating the behavior of the transducer are listed as follows:

1. For a '*one-way corridor*' input, neither the state nor the stack content need be changed. Also, take as output the symbol that guides the ALV to move along the one-way corridor.
2. For an '*end of corridor*' input:
   2.1. If the navigation transducer is in the 'exploring state', change the state of the navigation transducer to the 'retreating state' and keep the content of the stack unchanged. Also, take as output the symbol that guides the ALV to move backward along the corridor.
   2.2. If the navigation transducer is in the 'retreating state' and the stack is empty, change the state to the 'final state'. Also, take '*stop the ALV*' as the output symbol.
3. For an '*unexplored crossing*' input, push first the symbol which corresponds to the retreating path, i.e., the direction to the previous grid, onto the stack; scan next the candidate paths of the crossing in the sequence of forward, right-turn, left-turn; ignore the first candidate path, which is chosen to be the navigation direction; and push finally the symbols which correspond to the other candidate paths onto the stack. In this case, the state of the navigation transducer remains in the exploring state. Also, take as output the symbol that guides the ALV to go along the chosen path.
4. For an '*explored crossing*' input:
   4.1. If the navigation transducer is in the 'exploring state', change the state of the navigation transducer to the 'retreating state' and keep the content of the stack. Also, take as output the symbol that guides the ALV to move backward along the corridor.
   4.2. If the navigation transducer is in the 'retreating state', and if the stack is empty (i.e., if the popped symbol is the start symbol of the stack), change the state into the 'final state' and take '*stop the ALV*' as the output symbol. If the stack is nonempty, pop a symbol from the top of the stack, and if the popped symbol corresponds to an unexplored path, switch the state to the 'exploring state' and take as output the symbol that guides the ALV to go forward along the path, as implied by the popped symbol. If the popped symbol corresponds to a retreating path, keep the state in the 'retreating state' and take as output the symbol that guides the ALV to go backward along the path, as implied by the popped symbol.

4.2. Preprocessing unit

The preprocessing unit is designed to encode the local environment features into one of the four input symbols

of the navigation transducer. Since the major difference among the four input symbols is the number of available exploration paths from the current ALV position, the major objective of the encoding work is to find out all available paths from the current ALV location in the learned environment model. A navigation path is said to be available if the distance between two neighboring obstacles is longer than a pre-selected threshold value, which ensures safe navigation. In our approach, path-available tests are performed to the environment features in three certain (left, right and front) regions around the current ALV location. If no available path is found, an '*end of corridor*' symbol will be sent to the navigation transducer. If there is only one navigation path available, a '*one-way corridor*' symbol will be sent to the navigation transducer. A crossing is encountered when more than one navigation paths are available. In such a case, the location of the crossing area is first calculated. An exploration test is then performed to check if the encountered crossing is in the explored-crossing list, and either '*unexplored crossing*' or '*explored crossing*' would be sent to the transducer according to the test result. The encountered crossing will be added into the explored-crossing list after it has been explored.

4.3. Path-planning unit

The path-planning unit interprets the output symbol of the navigation transducer as a series of low-level ALV control commands. For a 'straight forward' or a 'straight backward' output, the path-planning unit generates a path along the corridor, i.e., guides the ALV to move along the middle line of the right-side wall and the left-side wall. For a 'forward right turn', a 'forward left turn', a 'backward right turn', or a 'backward left turn' output symbol, the path-planning unit generates a turning path using the information of the learned environment structure. For example, for a 'forward right turn', the path-planning unit generates three subpaths, two straight subpaths along the starting and ending directions of the turn and one circular subpath between the two straight paths, to guide the ALV to make this turn. The accurate position of the subpaths can be obtained by observing the positions of walls in the learned environment model, based on the principle of 'keeping the ALV along the middle of the corridors'.

Once the path is determined, the driving wheel direction δ can be calculated by a wheel adjustment strategy in [8]. The basic idea is to search a turn angle of the front wheels to drive the ALV as close to the desired path as possible. As shown in Fig. 6, given a reasonable moving distance S and a fixed turn angle of the front wheels, the location of the ALV can be estimated, as discussed in Section 3.1. Given a path P, either a straight line or a circular segment, define $D_P^F(\delta)$ as the distance from the midpoint between the two ALV front wheels to the given path P after the ALV traverses a certain distance S with the turn angle δ, where S may be assigned to be the average navigation distance during a cycle. Also, define $D_P^B(\delta)$ as the distance from the midpoint between the two ALV back wheels to the given path P. Finally, define measure L_P to be

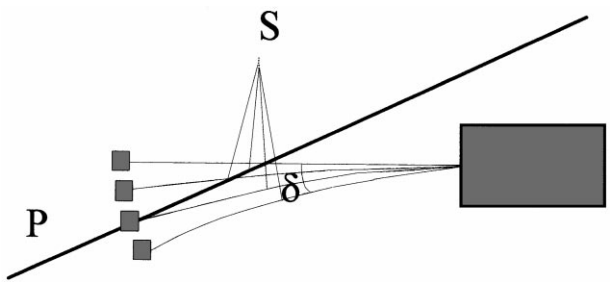

Fig. 6. Illustration of adjustment of the front wheels in a path 7.

$$L_P(\delta) = D_P^F(\delta) + D_P^B(\delta). \tag{8}$$

To find the turn angle of the front wheel to drive the ALV as close to the path as possible, an exhaustive search is performed to find the angle that produces the minimal value of L_P. The obtained angle is used as the turn angle for safe navigation.

5. Experimental results

An ALV navigation simulation system was developed to test the automatic exploration ability of the proposed system. The inputs of the simulation system are the locations of the obstacles in the navigation environment and the start position of the ALV. The simulation system generates a series of animations to show the movements of the ALV in the navigation session. The trace of the ALV in a simulated navigation session is shown in Fig. 7.

The external view of the prototype of the ALV is shown in Fig. 8(a). The ALV is computer-controlled with a modular architecture, as shown in Fig. 8(b), including four major components, namely, a vision system, a central processing unit (an Intel Pentium 133 MHz PC), a motor control system, and a DC power system. The vision system consists of a camera, a TV monitor, and a TARGA + image frame grabber. The motor control system consists of a main control board with an Intel 8085 controller, a motor driver, and two motors.

The image processing works for extracting line segments and corners in sensed images are accomplished in two phases. In the first phase, the pixels with higher gradient values and higher gray level values are extracted to form a candidate set. In the second phase, an algorithm similar to the edge-linking algorithm is performed to form a set of line segments and corners from the candidate set. Two examples of image processing results are shown in Fig. 9. The pixels in the candidate set are

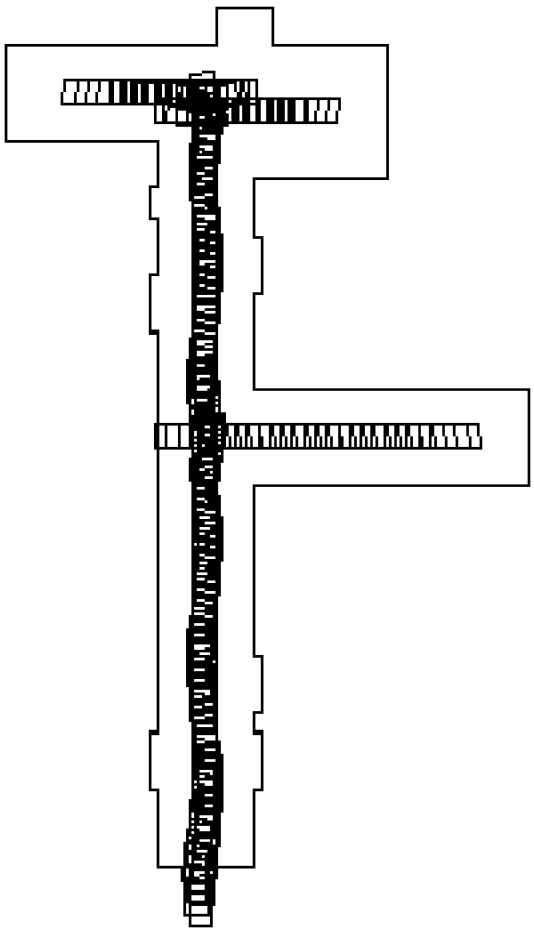

Fig. 7. The trace of the ALV in a simulated navigation session.

shown as white spots, and the extracted environment features are shown as gray line segments.

The ALV learning and navigation experiments were performed in a building corridor in National Chiao Tung University. By using the proposed approach, many successful navigation sessions have been conducted. The navigation speed of the vehicle is about 30 cm/s. The computation time of a navigation cycle ranges approximately from 1.5 to 3.5 s for different images. An example of the learned global models is shown in Fig. 10.

Fig. 8. The prototype ALV used in the experiments. (a) External view. (b) System structure.

6. Conclusion and future works

In this study, we have developed a system with the capability of automatic exploration and unsupervised learning of indoor corridor environment for ALV navigation. For unsupervised learning, a scheme to locate the environment features by computer vision techniques with the aid of laser markers, a new fast algorithm for orthogonal-line-segment-pattern matching and a systematical algorithm to construct the learned environment model have been proposed. For automatic exploration, a navigation transducer was proposed to serve as the guidance kernel. An algorithm for encoding the sensed environment structure into the input symbol to the transducer and a scheme for performing the ALV actions according to the outputs of the transducer have also been proposed. The proposed learning system has been implemented on a prototype ALV and successful navigation sessions in indoor corridor environments confirm the feasibility of the approach.

Fig. 9. Two examples of image processing results. The pixels in candidate set are shown as white spots, and the extracted environment features are shown as gray line segments.

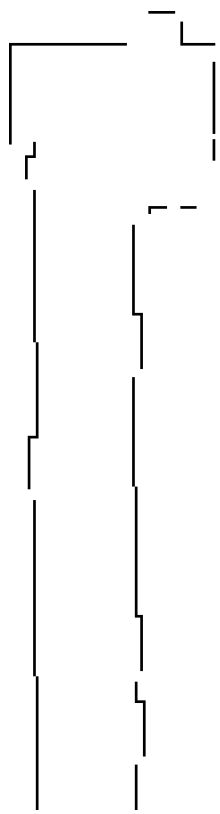

Fig. 10. An example of learned global models.

Appendix A. Coordinate systems and transformation

In the proposed ALV environment learning system, the following four coordinate systems are used to describe the vehicle location and the navigation environment.

1. The vehicle coordinate system (VCS): denoted as x–y–z. The origin V of the VCS is chosen to be at the middle point of the line segment which connects the two contact points of the two front wheels with the ground. The x-axis and y-axis are on the ground and parallel to the short and the long sides of the vehicle body, respectively. The z-axis is vertical to the ground.
2. The camera coordinate system (CCS): denoted as u–v–w. The camera is associated with the camera coordinate system whose origin C is attached to the camera lens center. The v-axis is coincident with the optical axis and the u–w plane is parallel to the image plane.
3. The image coordinate system (ICS): denoted as u–w. The image plane of the image coordinate system is coincident with the u–w plane of the CCS and its origin I is the image plane center.
4. The global coordinate system (GCS): denoted as x'–y'–z'. The origin G of the global coordinate system is located at a certain fixed position. The x'-axis and y'-axis are defined to lie on the ground. In this study, the environment features, line segments on the top-view map, are all assumed to be parallel to x'-axis or y'-axis.

These four coordinate systems are shown in Fig. 11. Since the origins of the ICS, CCS, and VCS are attached to some points on the ALV, the ICS, CCS, and VCS are moving with the vehicle during navigation. On the contrary, the GCS is fixed and is defined to be coincident with the VCS when the ALV is at the starting position in the initial model learning stage.

The transformations between these four coordinate systems can be found in [18]. Note that since the ALV

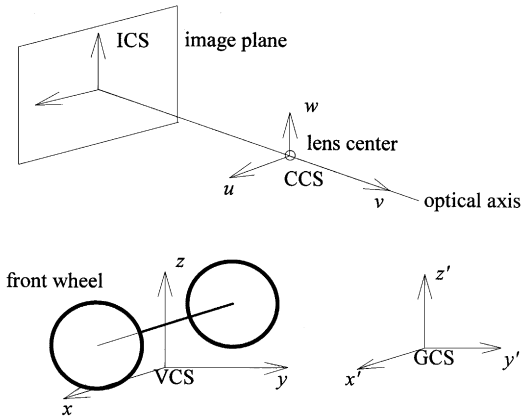

Fig. 11. The four coordinate systems ICS, CCS, VCS and GCS.

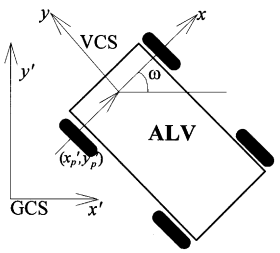

Fig. 12. The relation between VCS and GCS.

always navigates on the ground, the relation between the two 2D coordinate systems x–y and x'–y' is sufficient to determine the position and orientation of the vehicle. In other words, the translation vector (x'_p, y'_p) and the rotation angle ω of the ALV in the x'–y' coordinate system as shown in Fig. 12 determine the position and the direction of the vehicle in the GCS, respectively. The transformation between the GCS and the VCS can be written as

$$(x', y', 1) = (x, y, 1)\begin{bmatrix} \cos\omega & \sin\omega & 0 \\ -\sin\omega & \cos\omega & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x'_p & y'_p & 1 \end{bmatrix}. \quad (9)$$

References

[1] Lebègue X, Aggarwal JK. Extraction and interpretation of semantically significant line segments for a mobile robot. Proceedings of 1992 IEEE International Conference on Robotics and Automation, Nice, France, May 1992. p. 1778–85.

[2] Lebègue X, Aggarwal JK. Generation of architectural CAD models using a mobile robot. Proceedings of 1994 IEEE International Conference on Robotics and Automation, San Diego, California, USA, Vol. 1. May 1994. p. 711–7.

[3] Nashashibi F, Devy M, Fillatreau P. Indoor scene terrain modeling using multiple range images for autonomous mobile robots. Proceedings of 1992 IEEE International Conference on Robotics and Automation, Nice, France, Vol. 1. May 1992. p. 40–6.

[4] Nashashibi F, Devy M. 3D incremental modeling and robot localization in a structured environment using a laser range finder. Proceedings of 1993 IEEE International Conference on Robotics and Automation, Vol. 1. May 1993. p. 20–7.

[5] Ishiguro H, Maeda T, Miyashita T, Tsuji S. A strategy for acquiring an environmental model with panoramic sensing by a mobile robot. Proceedings of 1994 IEEE International Conference on Robotics and Automation, San Diego, California, USA, Vol. 1. May 1994. p. 724–9.

[6] Kurz A. Constructing maps for mobile robot navigation based on ultrasonic range data. IEEE Trans System Man Cybernet — Part B: Cybernet 1996;26(2):233–42.

[7] Dean T, Angluin D, Basye K, Engelson S, Kaelbling L, Kokkevis E, Maron O. Inferring finite automata with stochastic output functions and an application to map learning. Machine Learning 1995;18:81–108.

[8] Pan FM, Tsai WH. Automatic environment learning and path generation for indoor autonomous land vehicle guidance using computer vision techniques. Proceedings of 1993 National Computer Symposium, Chia-Yi, Taiwan, Republic of China, 1993. p. 311-21.

[9] Chen GY, Tsai WH. An incremental-learning-by-navigation approach to vision-based autonomous land vehicle guidance in indoor environments using vertical line information and multi-weighted generalized hough transform technique. Proceedings of 1996 Conference on Computer Vision, Graphics, and Image Processing, Taichung, Taiwan, Republic of China, August 1996. p. 151–58.

[10] Chen GY, Tsai WH. Unsupervised learning of unexplored environment by pushdown transducer for autonomous land vehicle navigation. Proceedings of 1997 Conference on Computer Vision, Graphics, and Image Processing, Taichung, Taiwan, Republic of China, August 1997. p. 335–42.

[11] Dorigo M. Introduction to the special issue on learning autonomous robots. IEEE Trans System. Man Cybernet — Part B: Cybernet 1996;26(3):361–4.

[12] Donnart JY, Meyer JA. Learning reactive and planning rules in a motivationally autonomous animat. IEEE Trans System Man Cybernet — Part B: Cybernet 1996;26(3):381–95.

[13] Yamauchi B, Beer R. Spatial learning for navigation in dynamic environments. IEEE Trans System Man Cybernet — Part B: Cybernet 1996;26(3):496–504.

[14] Qiao L, Sato M, Takeda H. Learning algorithm of environmental recognition in driving vehicle. IEEE Trans System Man Cybernet 1995;25(6):917–25.

[15] Haralick RM, Shapiro LG. Computer and robot vision, Vol 2. Reading, MA, USA: Addison-Wesley, 1993.

[16] Ballard DH. Generalizing the Hough transform to detect arbitrary shapes. Pattern Recognition 1981;13(2):111–22.

[17] Hopcroft JE, Ullman JD. Introduction to automata theory, languages, and computation. Reading, MA, USA: Addison-Wesley, 1979.

[18] Foley JD, Dam AV. Fundamentals of interactive computer graphics. Reading, MA, USA: Addison-Wesley, 1982.