Augmented Reality-based In-vehicle Tour Guidance in Park Areas by Vertical-line Features in Omni-images

[1]*Yen-Cheng Wei*, [2]*Bao-Shuh P. Lin and* [3]*Wen-Hsiang Tsai*

[1]Institute of Multimedia Engineering
[2, 3] Department of Computer Science
National Chiao Tung University, Hsinchu, Taiwan
Emails: nick0301n@hotmail.com, whtsai@cis.nctu.edu.tw

ABSTRACT

A new tour guidance system for use in a vehicle driven in park areas based on augmented reality and omni-vision techniques is proposed. A passenger in the vehicle can get tour guidance information from a passenger-view image generated by the system and displayed on a passenger-held mobile device. The passenger-view image simulates what is seen by the passenger through the front vehicle window. Nearby building information is augmented on the passenger-view image for convenient inspection. To implement the system, a method for vehicle localization are proposed, by which line features are detected and analyzed to compute the vehicle position by the longest common subsequence algorithm. Also proposed is a method for generating the passenger-view image from acquired omni-images and augmenting building information on correct positions in the passenger-view image using the building positions yielded by the vehicle localization process. Good experimental results showing the feasibility of the proposed system are also included.

Keywords: **omni-image; augmented reality; tour guidance; longest common sequence algorithm**

I. INTRODUCTION

Today, video cameras are now used widely, bringing convenience to people. A new possible application is to affix a video camera on top of a vehicle and display the acquired images of the scenes in front of the vehicle on the screen of a mobile device held by an in-vehicle passenger, on which the information of along-path buildings can be "augmented" properly for tour guidance in a park area. This study aims to develop a system of this new type of *augmented reality (AR) based tour guidance* for uses by in-vehicle passengers with hand-held mobile devices. An illustration of such a type of tour guidance is shown in Fig. 1.

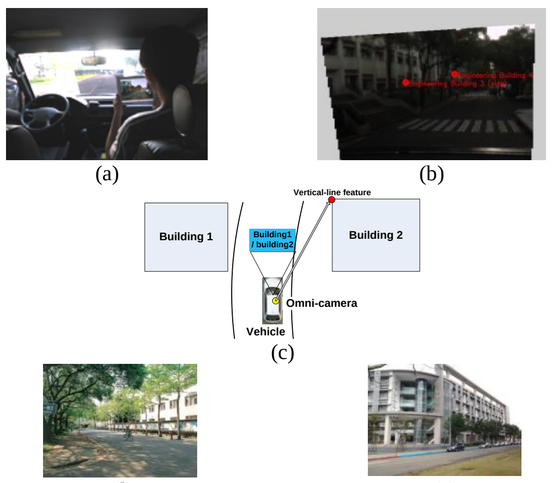

Fig. 1. Proposed AR-based tour guidance system in a moving vehicle. (a) An in-vehicle passenger holding a mobile device. (b) An "augmented" passenger-view image displayed on mobile-device screen. (c) Illustration of proposed system seen from top. (d) & (e) Examples of scenes with vertical lines on objects (buildings, light poles, etc.)

To implement such a type of AR-based guidance system, the core technique is to estimate the ^{v}ehicle location in every navigation cycle during the tour guidance process, which we call *vehicle localization* in this study (equivalent to the terms of *camera-pose estimation* and *user positioning* used in other studies). The uses of the Google Map and the GPS [1, 2] seems applicable for this purpose, but the imprecision in the GPS positioning results with errors ranging from 3 to 15 meters causes unacceptable AR effects on the mobile-device screen — the augmented guidance information will appear at incorrect positions on the screen, especially when the vehicle is driven through narrow roads with buildings close by.

[1] This work was supported by NSC project No. 101-2221-E-009-047-.
[3] Wen-Hsiang Tsai is also with the Department of Information Communication, Asia University, Taichung, Taiwan 41354.

Instead, an *omni-vision* approach is adopted in this study for vehicle localization, i.e., an omni-camera is affixed on top of the vehicle for taking images of the vehicle-surround to locate the vehicle. The abundant features appearing in the acquired environment images can be made use for more accurate vehicle localization. Specifically, the numerous "natural" *vertical-line* features, which are long edges appearing on along-path objects and perpendicular to the ground, like buildings and light poles such as those shown in Figs. 1(d) and 1(e), are utilized in this study for this purpose. This contrasts with the technique proposed by Chen and Tsai [3] in a similar study using "artificial" circular-shaped landmarks laid on sidewalks, which are less easy to deploy and apt to be destructed.

About other related vision-based localization methods for AR applications, Johansson and Cipolla [4] proposed a camera-pose estimation method from a single image of a building which is modeled as a number of parallel planes. Robertson and Cipolla [5] proposed a method for matching a single image against a database of facade views of buildings with known positions. Coors et al. [6] proposed a localization system using a 3D city model rendered from different viewing angles. Lee et al. [7] conducted tracking of the omni-camera pose in outdoor environments according to 5-degree-of-freedom motion estimates between two reference images and an input image. Reitmayr and Drummond [8] proposed a model-based hybrid tracking system combining the uses of edge-based tracking, gyroscope, gravity, magnetic field measurement, and a database of reference frames. Wagner et al. [9] combined two techniques, namely, SIFT and Ferns, for tracking natural features from planar targets in realtime on mobile phones. Uchiyama and Marchand [10] proposed an approach for detecting and tracking various types of planar objects with geometric features by using traditional keypoint detectors with the locally likely arrangement hashing technique.

The remainder of this paper is organized as follows. In Sec. 2, the system configuration and processes are introduced. In Sec. 3, the proposed vehicle localization idea is described. In Sec. 4, the process of environment learning is presented. In Sec. 5, the details of the vehicle localization process are described. In Sec. 6, the proposed method for AR-based tour guidance is presented. In Sec. 7, some experimental results are included, followed by conclusions in Sec. 8.

II. System Configuration and Processes

The configuration of the proposed system is shown in Fig. 2(a), which is of a client-server structure. A *client-side system* is set up on a laptop computer in the vehicle. The system controls an omni-camera affixed on the top of the vehicle to acquire an omni-image in every navigation cycle and transforms the omni-image into a *passenger-view image*, which "simulates" what is seen through the vehicle window by a passenger sitting aside the driver in the vehicle (see Figs. 1(a) and 1(b) for an illustration). The passenger-view image and a reduced version of the originally-acquired omni-image are then sent to a remote *server-side system* through a 4G/LTE network.

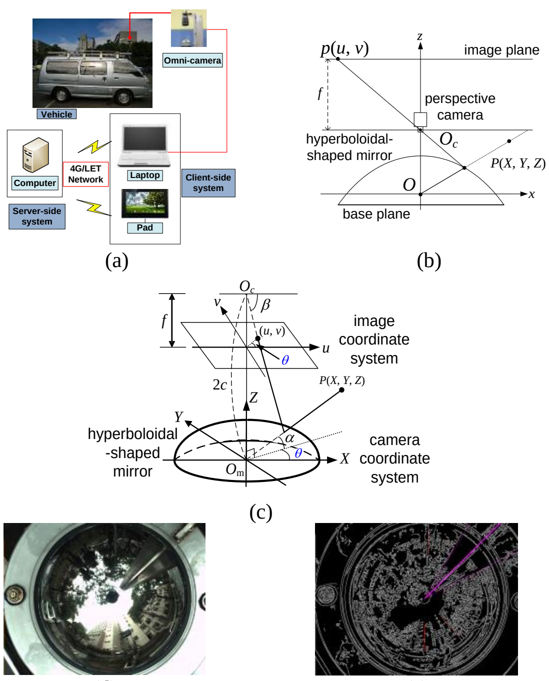

Fig. 2. Proposed system. (a) Configuration. (b) Side view of used omni-camera. (c) Details of (b) with image and camera coordinate systems shown. (d) Multiple vertical lines coming from a light pole and two building edges. (e) Extracted vertical lines from (a).

In the *navigation stage* in which tour guidance is conducted, the server-side system, after receiving the image data, analyzes them by the following steps: 1) detect vertical lines in the reduced omni-image; 2) match the vertical lines against a previously-learned vertical-line database to compute the location of the vehicle; 3) calculate the positions and orientations of the close-by buildings seen along the path (see Fig. 1(c) for an illustration) with respect to the vehicle's location and orientation; 4) unwrap the acquired omni-image and generate a passenger-view image from the result; 5) compute the positions of the closed-by buildings on the passenger-view image using the data obtained in Step 3); 6) augment the names and related information of the closed-by buildings on the passenger-view image by imposing them on the positions computed in Step 5); 7) send the augmented passenger-view image to the client-side system which then displays it promptly on the mobile-device screen.

III. Vehicle Localization by Vertical lines

A. Proposed Vehicle Localization Idea

The core of the proposed system is the vehicle localization technique by which AR-based tour guidance can be conducted. The main idea of the proposed vehicle localization technique is to match the vertical lines of objects appearing in the acquired omni-image against a pre-learned vertical-line database. This idea is based on two properties of the omni-image acquired by the omni-camera [11], which are reviewed in the following (refer to Figs. 2(b) and 2(c) for illustrations of the omni-camera structure and the involved coordinate systems, respectively, and also to Figs. 2(d) and 2(e) for a real omni-image and a processed version of it with edges extracted, respectively): 1) the *radial-line property*: a vertical line L in the real-world space will appear to be a radial line l going through the center of the omni-image if the base plane of the mirror of the omni-camera is parallel to the ground, as can be seen from the images of Figs. 2(d) and 2(e); 2) the *rotational-invariance property*: a real-world point P will appear to be an image point p in the omni-image with its orientation θ in the image coordinate system (ICS) identical to that of P with respect to the camera coordinate system (CCS) as long as the u-axis direction of the ICS is taken to be identical to the X-axis direction of the CCS, as illustrated by Fig. 2(c) where the angle θ is marked in blue.

From the above two properties, a 3rd property of the omni-image can be figured out: 3) the *vertical-line property*: the orientation (of the *base point* P) of a vertical line L in the real world with respect to the CCS is identical to that of the corresponding radial line l in the omni-image with respect to the ICS, as shown in Figs. 3(a) and 3(b).

According to the vertical-line property mentioned above, at *each specific spot* along the tour path, since each vertical line L on the along-path objects "seen" from the omni-camera is *fixed* with its orientation unchanged all time, the radial line l corresponding to L in the omni-image are *fixed as well* with the same orientation. Therefore, the multiple radial lines in the omni-image may be regarded as *stable features* and can be utilized for vehicle localization by a way of *vertical-line matching*, as illustrated in Figs. 3(c) through 3(e) in which three vertical lines coming from a light pole and two building edges are utilized.

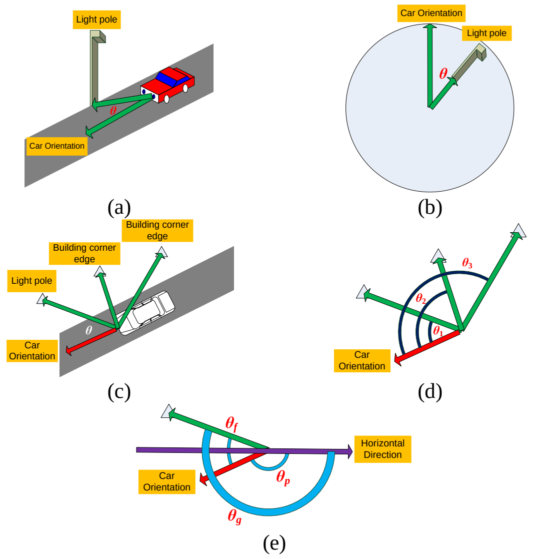

Fig. 3. Matching vertical lines for vehicle localization. (a) The vehicle passing a light pole with orientation θ. (b) An omni-image with the light pole detected with same orientation θ in omni-image. (c) Illustration of (a) from top. (d) Orientation angles of vertical-line features. (e) Computing orientation angle θ_f of vertical line.

Subsequently, the radial line l coming from the projection of a vertical line L onto the omni-image will be said to be a *vertical-line feature*. It is noted by the way that the orientation of each vertical line in the real-world space, computed with respect to the X-axis direction of the CCS, is also the direction of the vehicle movement, or equivalently, the direction of the currently-visited path segment, as illustrated in Figs. 2(c) and 3(e).

B. Vehicle Localization by Line Features

For the system to match multiple vertical-line features for vehicle localization during the navigation stage, a database of along-path vertical-line features has to be "learned" in advance in a *learning stage*. The details of such a feature learning process will be described later. The more vertical lines are used in the feature matching process, the more accurately vehicle localization can be conducted.

However, even if in the learning stage the multiple vertical-line features are learned completely, in the navigation stage some of such features might be *missed* or *extra* features might be detected due to various error sources in the acquired omni-image. That is, *feature insertions* and *deletions* should be considered in the vertical-line matching process in addition to *normal* feature matchings called *substitutions*. To handle these three types of feature matching, the *longest common subsequence (LCS) algorithm* is appropriate, which we propose for use in this study.

The LCS algorithm aims at finding the longest subsequence common to two or more symbol sequences. For our study here, each *symbol* is a vertical-line feature in the omni-image, specified mainly as the *orientation angle* of the vertical line in the real-world space, or equivalently, as that of the vertical-line feature in the omni-image, according to the previously-mentioned vertical-line property of the omni-image. If the orientation angles of two vertical-line features are distinct, the corresponding symbols are regarded as different as well. Moreover, there are two input symbol sequences only for our case here, one being that of the vertical-line features found in the omni-image acquired of the current-visited spot, and the other that of the vertical-line features kept in the learned database. The LCS algorithm is as follows

Algorithm 1. Finding the LCS.

Input: two sequences of symbols $S_x = < x_1, x_2, \ldots, x_n>$ and $S_y = < y_1, y_2, \ldots, y_m>$.

Output: the length L of the LCS of S_x and S_y.

Steps.

1. If $n = 0$ or $m = 0$, then set $L = 0$.
2. (*Symbol substitution*) If $x_n = y_m$, then do the following steps: 2.1) with $S_x' = < x_1, x_2, \ldots, x_{n-1}>$ and $S_y' = < y_1, y_2, \ldots, y_{m-1}>$ as inputs, perform this algorithm (Algorithm 1) recursively to compute the length L' of the LCS of S_x' and S_y'; 2.2) set $L = L' + 1$.
3. If $x_n \neq y_m$, then do the following steps: 3.1) (*Symbol deletion*) With $S_x = < x_1, x_2, \ldots, x_n>$ and $S_y' = < y_1, y_2, \ldots, y_{m-1}>$ as inputs, perform this algorithm (Algorithm 1) recursively to compute the length L_n of the LCS of S_x and S_y'; 3.2) (*Symbol insertion*) With $S_x' = < x_1, x_2, \ldots, x_{n-1}>$ and $S_y = < y_1, y_2, \ldots, y_m>$, perform this algorithm (Algorithm 1) recursively to compute the length L_m of the LCS of S_x' and S_y; 3.3) If $L_n > L_m$, set $L = L_n$; else, set $L = L_m$.

To use the above algorithm for the purpose of vertical-line matching, we have to modify the conditions $x_n = y_m$ and $x_n \neq y_m$ used in Steps 2 and 3, respectively, because the orientation angles of the vertical-line features detected by the system have in general *small variations* due to various error sources. That is, we set a tolerant range d for determining whether $x_m = x_n$ or not: if $|x_m - x_n| < d$, then yes; else, no. Then, we take d as an additional input to Algorithm 1 and use it in Steps 2 and 3 as said. The algorithm we propose for vehicle localization is as follows.

Algorithm 2. Vehicle localization.

Input: 1) a sequence S_x of the multiple vertical-line features detected by the system at a path spot; 2) a threshold d for judging the equality of two orientation angles; 3) a threshold D for judging if a matching is successful, and 4) a set S_e of learned data which include a series of along-path vehicle positions P_i and the associated sequences S_i of the vertical line features "seen" at each P_i.

Output: the decided vehicle position P_v or a message of "fail to localize the vehicle."

Steps.

4. Set $L = 0$.
5. Take sequentially an unprocessed sequence S_i with length L_i from the learned data set S_e.
6. With S_x, S_i, and d as inputs, perform Algorithm 1 to compute the length L' of the LCS of S_i and S_x.
7. If L' is larger than L and the ratio $L'/L_i > D$, then set $L = L'$ and take the vehicle position P_v to be the position P_i corresponding to S_i, meaning that the vehicle is located to be at position P_i so far.
8. If S_e is not exhausted, then repeat Steps 2 through 4; else, carry out either of the following two steps: 1) if $L \neq 0$, then take the final P_v as the desired vehicle localization result; 2) if $L = 0$, then output the message "fail to localize the vehicle."

When the above algorithm fails to localize the vehicle, the "old" vehicle location is not modified but taken to be that "predicted" using the data of the previous navigation cycle.

IV. Learning of Environments

A. Learning of Environment Map

According to the proposed vehicle localization technique described above, an *environment learning* process is proposed for use in the learning stage. The major task is to construct an *environment map* which includes the information of: 1) a real-world map, 2) a set of pre-selected paths, 3) the vertical line features seen along each path, and 4) the "to-be-augmented" names and other information of the buildings along each path.

In this process, at first a real-world map of the navigation environment, which is an image, is downloaded from the application system

"OpenStreetMap" on a website. Then, all desirable navigation paths are selected on the map, with each path drawn as a sequence of piecewise line segments and including the positions of the end points, (x_1, y_1) and (x_2, y_2), of each line segment p in the path, and the length L_p and the orientation angle θ_p of p computed respectively by

$$L_p = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}\ ; \qquad (1)$$

$$\theta_p = \tan^{-1}[(x_1 - x_2)/(y_1 - y_2)], \qquad (2)$$

where θ_p is defined with respect to the horizontal direction of the above-mentioned real-world map.

Next, the vertical-line features seen along each line segment p of each navigation path are "learned." The learned data for each vertical-line F include: 1) the position of F on the real-world map when viewed as a single point from the top; 2) the geometric relation of F with respect to p (described in detail later); and 3) the orientation angle θ_f of F with respect to the direction of p, as illustrated in Fig. 3(e).

More specifically about data item 2) above, while the vehicle is moving on path line segment p, it is possible that only on *a part of p* can the vertical line feature F be seen by the omni-camera. Therefore, for each detected vertical line F, we save the direction angles θ_{first} and θ_{last} from which the system in the vehicle can detect F for the first time and for the last time, respectively, in the omni-image by a radial-line detection scheme proposed in this study. In other words, we "learn" the range $\theta_{first} \sim \theta_{last}$ of the direction angles (in degrees) in which the vertical line F can be seen by the omni-camera. As to data item 3) above, a process is designed in this study which we will describe in the next section.

Finally, AR information for each building seen along the path is "learned," which includes: 1) the geometry of the building drawn as line segments; 2) the name of the building; and 3) the geometric relation between the building and the path. Specifically, to learn such building information, we drive the vehicle on the path and check visually if the buildings appear in the created passenger-view image: if yes, we stop the vehicle, mark each "visible" building corner on the real-world map. Then, we connect every two building corners by a line segment to form a side of the building on the real-world map. Only visible building sides from the passenger viewpoint are kept; those invisible ones are removed. Finally, we save the building name into the environment map. These steps are repeated until the end of the path is reached.

After learning the above three types of map data, environment map construction is completed. Accordingly, the proposed system can calculate the position of a nearby building with respect to any line segment of each path on the map.

B. Learning of Vertical-line Features

The task of learning the data of each along-path vertical line includes two parts: learning of a single vertical-line feature *at a time* and learning of multiple line features *simultaneously*. The aim is to calculate the orientation angles of the vertical-line features observable from each line segment of the path for use in vehicle localization.

(a) Learning of single vertical-line feature

In the 1st part of the learning task, we drive the vehicle along the path and detect one by one vertical lines appearing in the acquired omni-image. Each time a vertical line F is detected along a line segment p of the path, we stop the vehicle, and carry out the works of 1) measuring manually the position P of F; 2) computing orientation angle θ_f of F; 3) associating F with p; and 4) marking F as a point on the map. We repeat this process until the end of the path is reached. Here, the orientation angle θ_f of F, as illustrated in Fig. 3(c), is computed by:

$$\theta_f = \theta_g - \theta_p \qquad (3)$$

where θ_p is computed according to (1) and θ_g is the orientation angle of P of the vertical-line feature F with respect to the horizontal direction of the map. Note that though θ_g is the real-world orientation angle of the vertical line, it can be obtained from the orientation angle of the vertical line in the omni-image according to the derived vertical-line property mentioned previously.

(b) Learning of multiple vertical-line features

In the 2nd part of the vertical-line learning task, multiple features are learned simultaneously *automatically by simulation* at each spot on each line segment along the path, differently from the first part which is conducted by car driving and manual measurements. Specifically, for each line segment p_i of the path in the map, and for each discrete point P_k on p_i, we conduct the following steps: 1) from the data learned in the first part of vertical-line learning task as described in Sec. 4.2(a), retrieve as a set S_i those vertical lines F_j which are associated with p_i; 2) pick out from S_i those vertical lines F_j which are *observable* from point P_k where *observableness from* P_k means that the direction of the line from P_k to F_j falls into the range $\theta_{first} \sim \theta_{last}$ between the first-time and last-time direction angles of vertical line F_j on p_i; 3) calculate the orientation angle θ_f of each retrieved vertical line F_j by the following steps: 3a) use the "learned" position of F_j and that of P_k on the map to compute the orientation angle θ_g of F_j with respect to the horizontal direction of the map; 3b) compute the angle θ_p of the vehicle's movement direction as that of path line segment p_i with respect to the horizontal direction of the map using the map data; 3c) compute the orientation angle θ_f for F_j according to Eq. (3); 4) save θ_f into a table T_i for p_i for uses in the navigation stage.

C. Detection of Vertical Lines in Omni-images

To detect vertical lines in an acquired RGB omni-image, we transform the image into a YUV version and apply edge detection to the Y-channel frame to extract the edge points. Then, based on the vertical-line property of the omni-image and starting from the omni-image center, unbroken radial lines are extracted as desired vertical lines by searching lines in all radial directions. A result of this process is shown in Figs. 3(a) and 3(b).

V. Proposed AR-based Tour Guidance

A. Vehicle Localization without Vertical Lines

At path spots where vertical-line features are not available for vehicle localization, we estimate the vehicle speed and predict accordingly the vehicle location in the next navigation cycle. Vehicle speed estimation is accomplished by the use of motion vectors computed from the acquired omni-image sequence. Because the vehicle is driven just in one direction, i.e., forward, on the path, we use vectors of that direction to simplify motion vector computation. Also, not all the motion vectors in the omni-image are useful for vehicle speed estimation; specifically, we cut out the road part in the omni-image for motion vector computation, as illustrated in Fig. 4. The computed directions of the motion vectors in this part are all the same as the movement direction of the vehicle. The motion vectors so computed are averaged finally for use as the desired vehicle speed.

B. Use of Prior Knowledge for Tour Analysis

During vehicle localization, some environment knowledge is utilized to speed up the process. The idea is just to detect the right feature at the right position. Firstly, we divide the omni-image into two parts, the left side and the right, as illustrated in Fig. 5, and vertical-line features in the two sides are detected respectively. Secondly, the vehicle is driven just forward, so the detected vertical-line features must move backward in the image sequence. Finally, by the estimated vehicle speed and the pre-learned database, the next positions of the vertical-line features can be predicted.

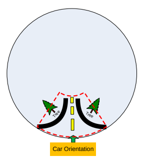

Fig. 4. Illustration of cutting a part of the omni-image (the area enclosed by red lines) to speed up motion vector computation.

Car Orientation
Car Orientation

(a) (b)

Fig. 5. Illustration of using knowledge of omni-image. (a) Omni-image which divided two parts. (b) Two parts of image.

B. Proposed Algorithm for Vehicle Localization

Based on the previous discussions and the mentioned prior knowledge, an algorithm for estimating the vehicle speed and another for implementing the idea of proposed vehicle localization are described in this section. Briefly speaking, in most areas along the path, the system uses vertical-line features to locate the vehicle; and at spots where appropriate vertical-line features are unavailable, the system uses the estimated vehicle speed to predict the next vehicle position.

Algorithm 3. Estimation of the vehicle speed.

Input: an image I_r of the last cycle, the image I_c taken of the visited spot in the current cycle, a $k\times l$ search window W, the vehicle direction $D(d_x, d_y)$, and two thresholds T_u and T_d.

Output: a value s of the current vehicle speed.

Steps.

1. Segment out the road parts in I_r and I_c as I_r' and I_c', respectively, as illustrated in Fig. 4.
2. Use I_r', I_c', and the search window W to compute motion vectors by the following steps, and save the result into a set S: 2.1) divide I_c' and I_r' into blocks B_i of size $z\times z$ with $z < l$ and k; 2.2) for each block B_i in I_c', conduct the following three operations to compute a motion vector V_i: a) let coordinates (i_b, j_b) specify the center of B_i, and (i_r, j_r) specify that of the block R_i in I_r' corresponding to B_i; b) search within the window W in the reference image I_r' to find a block R_i' centered at (i_r', j_r') which *matches* B_i *best* according to the measure of the average pixel-value difference between blocks B_i and R_i; c) compute the motion vector of B_i to be $V_i(v_{ix},$

v_{iy}) with $v_{ix} = i_b - i_r'$; $v_{iy} = j_b - j_r'$.

3. Delete those motion vectors V_i in S which are not in the vehicle direction $D(d_x, d_y)$.
4. Delete those motion vectors V_i in S with lengths larger than T_u or smaller than T_d.
5. Compute the mean of the lengths $v_i = (v_{ix}^2 + v_{iy}^2)^{1/2}$ of all the remaining V_i in S by $s = (\Sigma v_i)/N$ where N is the number of the remaining motion vectors; and take s as the output result.

Algorithm 4. Vehicle localization.

Input: the environment map *Map*, the last vehicle position $P_l(x_l, y_l)$, the image I_r taken in the last cycle, the image I_c taken of the visited spot in the current cycle, a $k\times l$ search window W, the vehicle direction $D(d_x, d_y)$, two thresholds T_u and T_d, and a tolerance d for judging angle equality.

Output: the current vehicle position P_v.

Steps.

1. Predict the vehicle position $P(x_c, y_c)$ by using the last vehicle position $P_l(x_l, y_l)$ and estimating the vehicle speed by the following steps: 1.1) with I_r, I_c, W, $D(d_x, d_y)$ and the two threshold values T_u and T_d as inputs, perform Algorithm 3 to estimate the speed s of the vehicle; 1.2) use s and $P_l(x_l, y_l)$ to compute the next vehicle position $P(x_c, y_c)$ with $X_c = x_l + s\times\cos\theta$; $y_c = y_l + s\times\sin\theta$, where θ is the direction angle of the currently-traversed line segment p_i of the path obtained from the environment map *Map*.
2. Use the predicted position $P(x_c, y_c)$ to retrieve the vertical-line feature table T_i of the line segment p_i of the path around $P(x_c, y_c)$ kept in *Map* as mentioned in Sec. 4.2(b).
3. If T_i is empty (meaning no vertical line is available for the current spot), then set the desired vehicle position P_v as $P(x_c, y_c)$ and exit; else, continue.
4. Detect the vertical lines in the current omni-image I_c and compute their orientation angles to form a sequence $S_x = <x_1, x_2, \ldots, x_n>$ according to (3), using a vertical-line detection process similar to that described in Sec. 4.3.
5. For each discrete point P_k in p_i, fetch from T_i the set S_f of the vertical lines corresponding P_k; and with S_x, S_f, and the threshold value d as inputs, perform Algorithm 2 to compute the position P_v of the vehicle as output.

VI. Proposed AR-based Tour Guidance

A. Construction of Passenger-view Images

In this study, we use the passenger-view image to generate the desired AR image. The passenger-view image is created to simulate what is seen by the passenger sitting aside the driver through the front window. Firstly, we assume that the viewpoint originates from the right-side passenger seat as illustrated in Fig. 6. Next, we use the method by Jeng and Tsai [11] to "unwarp" the acquired omni-image to be a *perspective-view image* as illustrated by Figs. 7(a) and 7(b). Then, we measure the positions of the four corners of the vehicle window and project them onto the perspective-view image from the passenger's view to cut a part of the image as the result, as illustrated by Fig. 7(c).

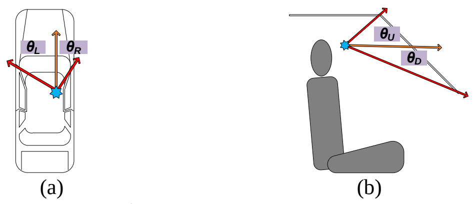

Fig. 6. Illustration of viewpoint in vehicle. (a) Top-view of vehicle where blue star is the passenger's viewpoint. (b) Side-view of vehicle.

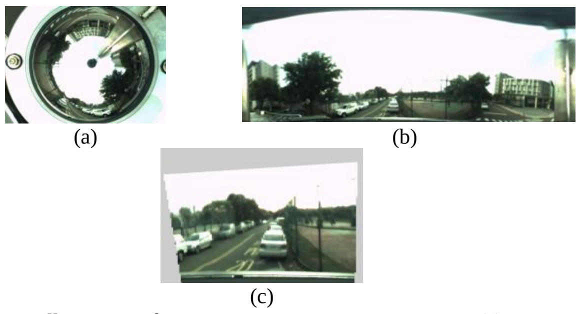

Figure 7. Illustration of generating a passenger-view image. (a) Omni-image acquired with the omni-camera. (b) Perspective-view image generated from (a). (c) Passenger-view image transformed from (b).

B. Augmenting Building Names on Images

After localizing the vehicle, we can obtain the position of the buildings observable by the passenger according to the environment map. Then, we can project the building positions onto the passenger-view image, on which we can finally *augment* the learned information of the building, including its name, to obtain the desired AR image for display on the mobile device. An example of the results is shown in Fig. 1(b).

VII. Experimental Results

A result of environment learning, which is an environment map, is shown in Fig. 8. In the learning stage, we drove a vehicle as shown in Fig. 9(a) on the path shown in Fig. 8. The system detected vertical lines like the example in Figs. 9(b) and 9(c), which were saved into a database.

In AR-based tour guidance, the system detects vertical line features and conducts vehicle localization along the path. Two examples of the vertical-line detection results are shown in Fig. 10. After locating the vehicle, the system used the result to generate an AR image on the use-held mobile device like the result shown in Fig. 11.

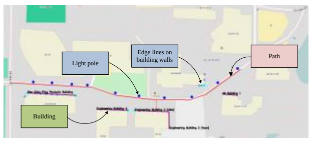

Fig. 8. The environment map learned by the proposed system.

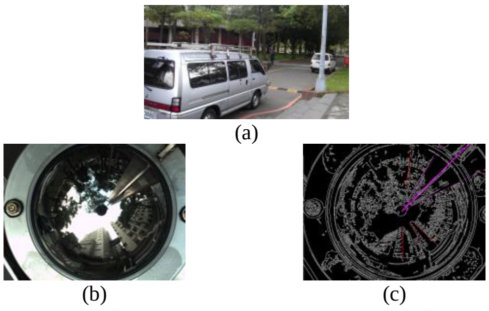

Fig. 9. A result of the learning stage. (a) An image of the vehicle driven on the path. (b) An omni-image acquired from the omni-camera. (c) Vertical-line features detected by the system.

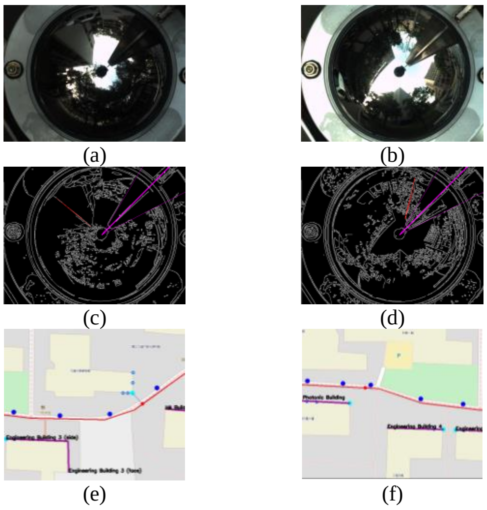

Fig. 10. Two results of detecting vertical lines and vehicle localization. (a) & (b) Acquired omni-images. (c) Line features detected from (a) & (b). (e) & (f) Vehicle locations computed by the system (indicated by red point) using (c) & (d), respectively.

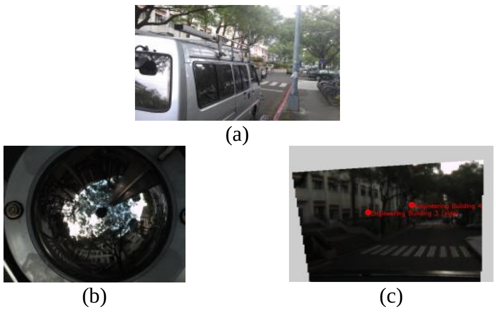

Fig. 11. AR-based navigation. (a) An image of the vehicle on a path spot. (b) An omni-image acquired. (c) Passenger-view image generated from (b) with name of nearby building augmented.

VIII. CONCLUSIONS

AR and omni-vision based tour guidance for outdoor environments has been proposed, by which an augmented passenger-view image which simulates what is seen through the vehicle window by a right-side passenger is generated. The image is displayed on a mobile-device screen held by the passenger. To implement this, several techniques have been proposed: 1) learning of the environment map; 2) detection of vertical lines in omni-images; 3) vehicle localization using vertical-line features; 4) generation of passenger-view images. The experimental results have revealed the feasibility of the proposed system.

REFERENCES

[1] "Google Maps/Google Earth APIs Terms of Service," June 2013. Available at http://www.google.com /apis/maps/terms.

[2] International GPS Service Central Bureau Data & products: the products, 2001. http://igscb.jpl.nasa.gov /components/prods.html.

[3] B. C. Chen and W. H. Tsai, "A Study on Tour Guidance by Vehicle Driving in Park Areas Using Augmented Reality and Omni-vision Techniques," *Proc. of IPPR Conf. on CVGIP*, Nantou, Taiwan, Aug. 2012.

[4] B. Johansson and R. Cipolla, "A system for automatic pose estimation from a single image in a city scene," *Proc. of IASTED Int'l. Conf. on Signal Process., Pattern Recog. & Applications*, Crete, Greece, June 2002.

[5] D. P. Robertson and R. Cipolla, "An image-based system for urban navigation," *Proc. 15th British Machine Vision Conf.*, Kingston-upon-Thames, UK, pp. 819-828, Sept. 2004.

[6] V. Coors, T. Huch and U. Kretschmer, "Matching buildings: Pose estimation in an urban environment," *Proc. of IEEE & ACM Int'l Symp. on Augmented Reality (ISAR 2000)*, pp. 89-92, Munich, Germany, October 2000.

[7] J. W. Lee, S. You and U. Neumann, "Tracking with Omni-Directional Vision for Outdoor AR Systems," *Proceedings of IEEE ACM Int'l Symposium on Mixed and Augmented Reality (ISMAR 2002), Darrnstadtt, Gkrmany*, October 2002.

[8] G. Reitmayr and T. W. Drummond, "Going out: Robust model based tracking for outdoor augmented reality," *Proc. IEEE Int'l Symp. Mixed and Augmented Reality* (*ISMAR'06*), Santa Barbara, California, USA, pp. 109–118, 2006.

[9] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond and D. Schmalstieg, "Pose tracking from natural features on mobile phones," *Proc. of IEEE Int'l Symp. Mixed and Augmented Reality (ISMAR'08)*, Cambridge, UK, pp. 125–134, 2008.

[10] H. Uchiyama and E. Marchand, "Toward augmenting everything: detecting and tracking geometrical features on planar objects," *Proc. of IEEE Int. Symp. on Mixed and Augmented Reality, (ISMAR'11)*, Basel, Switzerland, 2011, pp. 17-25.

[11] S. W. Jeng and W. H. Tsai, "Using pano-mapping tables for unwarping of omni-images into panoramic and perspective-view images," *Journal of IET Image Processing*, Vol. 1, No. 2, pp. 149-155, June 2007.