INTERNATIONAL JOURNAL OF INTELLIGENT CONTROL AND SYSTEMS
VOL. 10, NO. 3, SEPTEMBER 2005, 188-200

A Method For Mobile Robot Obstacle Negotiation

Cang YE

Abstract- **This paper presents an obstacle negotiation method for mobile robot navigation. The proposed method first transforms a local terrain map surrounding the robot's momentary position into a grid-type traversability map by extracting the slope and roughness of a terrain patch through least-squares plane-fitting. It then evaluates the overall traversal property along each direction. The traversal information is used to form a traversability field histogram, from which the steering and velocity commands for the robot are determined. The new concept of motion-context is proposed and used to guide the robot in such a way that the algorithm converges to the target with a finite-length path.**

***Index Terms*—obstacle negotiation, obstacle avoidance, terrain traversability analysis, traversability map, traversability field histogram, motion-context.**

1. INTRODUCTION

Autonomous navigation of mobile robots on non-flat terrain requires the ability to decide if the terrain and/or obstacle ahead are traversable or if they have to be circumnavigated. This, together with the ability to generate the appropriate steering and velocity commands for the robot, is termed Obstacle Negotiation (ON). An issue closely related to ON is terrain mapping.

Many of the existing algorithms for terrain mapping employ stereovision [1-5]. Stereovision has several disadvantages: (1) sensitivity to environmental conditions, (2) low range resolution and accuracy, and (3) range measurement errors that increase with the true range. To overcome the above problems, 3-D Laser Rangefinders (LRFs) have been used since the early nineties [6-8]. A 3-D LRF is usually costly and heavy, ant thus not suitable for small and/or expendable robots. Furthermore, the slow frame rate produced by 3-D LRFs can affect the robot's mobility if range data are acquired frame by frame.

The author recently developed a terrain mapping method [9,10] (briefly described in Section 2) using a much cheaper and lighter 2-D laser rangefinder for ON. It fully utilizes the accuracy of the laser range data for mapping but avoids the slow frame rate associated with 3-D LRFs. In this paper, an ON method is proposed to navigate the robot using the terrain mapping method.

An ON method has to address two issues: Terrain Traversability Analysis (TTA) and Path Planning. TTA is a process to evaluate the traversal property of a terrain map. Langer *et al.* [11] compute terrain elevation statistics (the minimum height, maximum height, and slope of a terrain patch) and classify terrain cells as traversable or untraversable ones by comparing the elevation statistics with threshold values. Gennery [12] proposes a method based on weighted least-squares fitting. His key idea is to find a least-squares plane to a terrain patch, and use the roll and pitch of the fitted plane, and the residual of the fit to estimate the slope and roughness of the terrain patch, respectively. In order to deal with the inaccuracy of the stereovision data, Gennery assigns a weight to each data point before searching for the least-squares plane. This method is computationally complex since the computation of the covariance matrix of each data point is required and the slope calculation is iterative.

A similar TTA algorithm without iterations is proposed by Singh [13]. In this algorithm, the roll, pitch, and roughness measures are normalized in the range [0, 1], and the lowest value among these three measures determines the overall goodness of a cell. A goodness map is created at each time instant by weighting a sequence of previous goodness maps. The weighting factors are proportional to the distance the robot travels. The traversability value of each cell is defined as the product of the goodness value and a so-called certainty value associated with that cell.

Seraji and Howard [5] propose a terrain traversability measure based on fuzzy logic. It is termed Traversability Index (TI) which represents the degree of ease for the robot to across the terrain. A TI is described by a number of fuzzy sets. Three terrain characteristics, namely, roughness, slope, and discontinuity, are extracted from the stereo image and described by fuzzy sets. A fuzzy inference system is then built to map the terrain characteristics to the TIs. The method is fast and allows real-time computation of TIs. The disadvantage is that the correctness and completeness of the fuzzy rules hinge on human expert who compiled the rules.

The existing path planning methods can be classified into two categories: global and local ones. A global method [14] is carried out in a completely known environment. It searches for an optimal path (i.e., with minimum cost) in the connected traversable space of the environment. A* algorithm [15] is a popular search method to find the optimal path. On the other hand, a local method, also called obstacle avoidance/negotiation, handles with unknown or partially known terrain where a robot has to explore the terrain using its sensor(s) and plan its motion on the fly [16]. For a real-world navigation problem, a local method is essential since complete terrain information is usually unavailable ahead of time or there may be unexpected change in the terrain even if it is known a priori.

Manuscript received October 30, 2005; revised December 28 2005. This work was funded by the U.S. Department of Energy under Award No. DE-FG04-86NE3796, by DARPA under Award. No. F007571. This paper is extended from "Obstacle avoidance for the segway robotic mobility platform" published at *American Nuclear Society International Conference on Robotic and Remote System for Hazardous Environment,* Orlando, FL, USA, April 2003.

C. Ye is with Department of Applied Science, University of Arkansas at Little Rock, 2801 S. University Ave, Little Rock, AR 72204, USA (e-mail: cxye@ualr.edu)

Local obstacle negotiation is often seen as a reactive behavior in behavior-based navigation methods [11, 5]. For instance, the system described in [5] comprises a Traverse-terrain Behavior (TB), an Avoid-obstacle Behavior (AB), a Seek-goal Behavior (SB), and a Behavior Integration Module (BIM). Each module uses fuzzy logic for decision-making. The BIM infers a weighting factor for each behavior module and the fuzzy commands recommended by the three modules are weighted and defuzzified to produce the motion command for a robot. The fuzzy inference system for TB and AB are layered. In the first layer, the terrain is partitioned into seven regions and the fuzzy inference system infers the three most traversable regions. These regions are preferred-left, front, and preferred-right, and they are selected according to the TIs of the seven regions. Then in the second layer, another fuzzy inference system determines the best sector among them. A similar method [17] with one layer is used for navigation on flat ground navigator. The main disadvantage of this kind of methods is that the completeness and correctness of the fuzzy rules are difficult to maintain and the effectiveness of the BIM cannot be quantified or proven. The layered structure of multiple fuzzy systems may add additional difficulties to these issues.

Some researchers employed a so-called "arcs approach" [4, 11, 13, 18]. In this approach the algorithm generates a number of candidate arcs and then either votes for the arc with the largest clearance [11] or calculates the cost along each arc and selects the one with the lowest cost [4, 13, 18]. The robot is then moved along the winning arc. The arcs approach is well suited for four-wheel robot whose movement can be approximated by arcs. Gennery proposes a probability method for path planning [12]. His method computes the cost of driving through a grid. The cost function comprised two components: the distance traveled and the probability that the slope or roughness may be too large to traverse. A path planner is then employed to find a path that minimizes the total cost. This method plans a path across a grid rather than along an arc. The probability approach takes into account the inaccuracy of stereovision data.

Motivated by the TangentBug algorithm proposed in [19], researchers at JPL develop the WedgeBug path planner [20] for planetary navigation. The algorithm modifies the omnidirectional vision requirement of the TangentBug to the more limited field of view of stereovision systems used on planetary rovers. Just as with the TangentBug method, the WedgeBug converges at the target with a finite-length path.

There are some attempts to apply potential field methods to terrain navigation [21]. However, the application of potential field methods to obstacle avoidance may cause the problem of oscillatory motion [22] or cause the robot to get trapped in a local minimum. These problems are alleviated by Vector Field Histogram (VFH) method [23].

This paper presents an ON method called Traversability Field Histogram that is an extension of the VFH algorithm. The paper is organized as follows: Section 2 provides a brief overview of the ON system. Sections 3 and 4 describe in greater details the TTA and path planning algorithms, respectively. Section 5 presents simulation and experimental results of the overall ON algorithm. Section 7 concludes the paper.

2. OVERVIEW OF THE TRAVERSABILITY FIELD HISTOGRAM METHOD

Figure 1 is a block diagram of the ON system. It consists of four main modules: Terrain Mapping, Terrain Traversability Analysis (TTA), Path Planning, and Motion Control. This section gives a brief overview of the system.

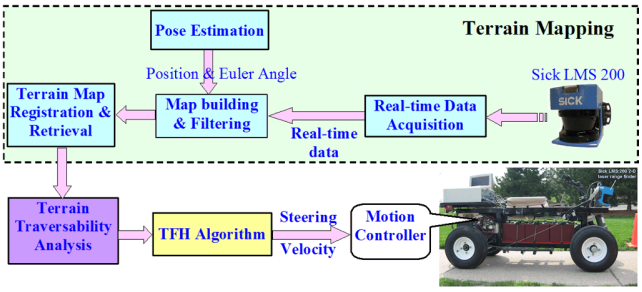


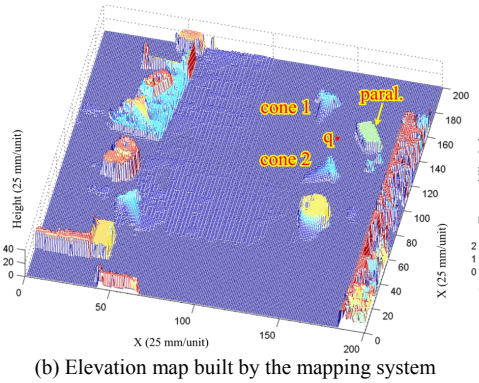
Fig. 1. Diagram of the obstacle negotiation system: the module within the dashed lines represents the Terrain Mapping module

The main vision sensor in the system is the SICK 2-D laser rangefinder (LRF) that is mounted on the front end of a mobile robot. The LRF looks forward and downward at the terrain. While the robot is in motion, the fanning laser beams sweep the terrain ahead of the robot and produce continuous range data of the terrain. The terrain data are then transformed into coordinate values in the world coordinate system using the robot pose information (roll, picth, yaw angle and XYZ coordinates) from the Pose Estimation (PE) system, and registered in a terrain map. The PE system is developed at the University of Michigan's Mobile Robotics Lab [24]. In the ON system, this terrain map is represented as a grid-type map in which each element or "cell" holds a value representing the height of the terrain at that cell. The details of our terrain mapping method are explained in [9].

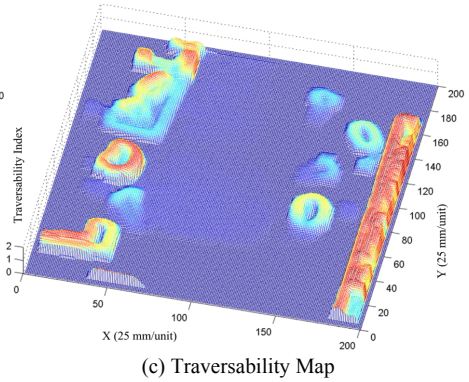
The task of the TTA module is to analyze the terrain map cell by cell and to generate a new grid-type map, called the "traversability map." Each cell in this map holds a value describing the degree of difficulty for the robot to move across the cell. This value is called the Traversability Index (TI).

The Path Planning module is a local path planner. It analyzes the traversability map and generates steering and velocity commands to move the robot along a direction with the overall lowest TI value. This paper introduces the TTA and the path planning method and it assumes the terrain map is available from the Terrain Mapping module.

(a) Obstacle course for mapping



(b) Elevation map built by the mapping system



(c) Traversability Map

Fig. 2. Traversability analysis on the elevation map built by the terrain mapping system

3. TERRAIN TRAVERSABILITY ANALYSIS (TTA)

The TTA module transforms a terrain map into a traversability map by assigning a TI value to each cell in the terrain map. This process is divided into two steps: estimating terrain slopes and roughness, and computing the TI value.

Suppose that a terrain map is denoted $\mathbf{E} = \{z_{i,j}\}$, where i and j are the row and column cell indices, respectively; and the Robot Geometric Center (RGC) is located at $(x_i, y_j, z_{i,j})$ in $\mathbf{E}$, where x_i and y_j are the coordinates corresponding to cells (i, j). A square terrain patch in $\mathbf{E}$ is defined as $P = \{z_{k,l} \mid_{k=i-L,\cdots,i+L;\, l=j-L,\cdots,j+L}\}$. P has a side length of $2L+1$ (in terms of the number of cells) and it is centered at cell (i, j). The terrain patch has to be so exact that it completely envelops the robot regardless of the robot's orientation. For example, if the robot is rectangular with side lengths A and B, then the side length, C, of the square patch should be $C = \sqrt{A^2 + B^2}$ and $L = \text{int}(C/2G)$, where G is the side length of the grid. The number of data points in the patch is $N=(2L+1)\times(2L+1)$. A least-squares plane to this terrain patch is found using the plane-fitting algorithm detailed in Appendix.

The slope of the terrain patch is then estimated by

$$\alpha = \cos^{-1}(\mathbf{n} \cdot \mathbf{b}) \tag{1}$$

where $\mathbf{n}$ is the normal to the least-squares plane and $\mathbf{b} = (0,0,1)^T$. The roughness of the terrain patch is approximated by

$$\sigma = |\mathbf{d}| = \sqrt{\sum_{i=1}^{i=N} d_i^2} \tag{2}$$

where $|\mathbf{d}|$ is the Euclidean norm of the residual of the fit $\mathbf{d}$ and d_i is the distance between the i^{th} data and the fitted plane (see the definition in Appendix). This paper concerns about terrain with hard surface. Therefore, the TI of cell (i, j) can be determined based on the slope and roughness of the terrain patch and it is computed by

$$\tau_{i,j} = F_1\alpha + F_2\sigma / N. \tag{3}$$

Apparently, F_1 and F_2 represent the contribution of the terrain slope and roughness to the TI value. In this study, $F_1=300$ and $F_2=6$ are used, which results in the slope having a slightly larger contribution than the roughness. This allows the robot to avoid high profile terrain features.

The traversability analysis is performed cell by cell in the terrain map. Once each cell is assigned a TI value, the terrain map $\mathbf{E}$ is converted to a traversability map $\mathbf{T} = \{\tau_{i,j}\}$. Fig. 2(b) depicts a terrain map built by our Terrain Mapping module using real sensor data when the robot traversed the obstacle course in Fig. 2(a). Figure 2(c) shows the traversability map obtained by the TTA algorithm (with a patch size: 9 cells × 9 cells), respectively.

In a grid-type terrain map, an obstacle's boundary comprises line segments in X, Y, or diagonal direction. The TTA process expands the boundary along X or Y direction by L cells and expands the diagonal boundary by $\sqrt{2}L$ cells. This automatically takes into account the robot's dimensions during navigation and it allows us to treat the robot as a point in the path-planning algorithm described in Section 4.

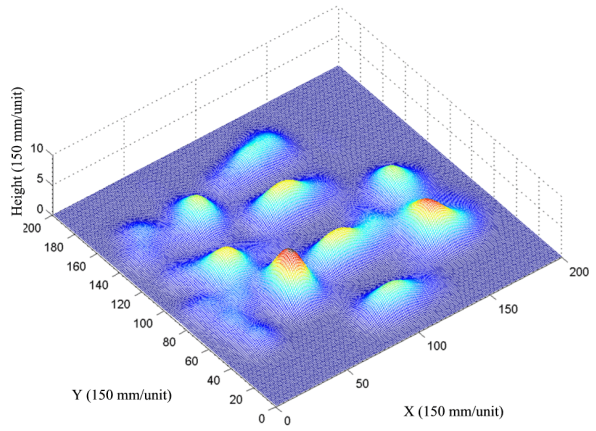
To test our ON method in the widest possible range of environments, we develop an ON simulator able to generate arbitrary terrain maps and run a simulated robot through those environments. Figure 3(a) shows the terrain map produced by the simulator while Figure 3(b) displays the traversability map computed by the TTA algorithm.

4. PATH PLANNING USING THE TRAVERSABILITY FIELD HISTOGRAM (TFH)

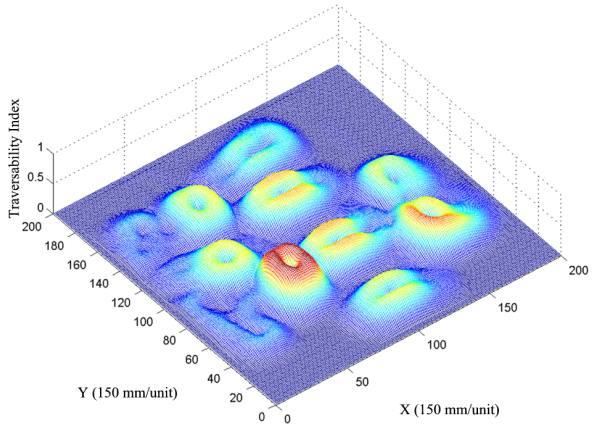
The proposed TFH algorithm first obtains a square-shaped local terrain map T^* from the current terrain map $\mathbf{E}$. T^* centers at the momentary position of the RGC and consists of $w_s \times w_s$ cells. In this paper, w_s=61 is used. As a result, T^* covers a physical region of 9.2 × 9.2 m.

Next, the TFH algorithm performs the terrain traversability analysis (described in the preceding section) over the cells in T^* and transforms the terrain map into a traversability map.

After the above preprocessing, the TFH algorithm computes the motion command based on T^*. The computation consists of two stages: (1) Transform T^* into a one-dimensional traversability filed histogram; and (2) compute motion commands based on the histogram.



(a) Elevation map generated by the simulator



(b) Traversability Map

Fig. 3. Traversability analysis on the elevation map generated by the simulator

4.1 Creating Traversability Field Histogram (TFH)

Figure 4 is the 2-D representation of part of the 2.5-D traversability map shown in Fig. 3(b). In Fig. 4, the TI value is rendered in color. The cells with nonzero TI values in the traversability map generate an imaginary vector field, which exerts virtual repulsive forces on the robot and repels it away from an untraversable region. We call this field "traversability field." A polar coordinate is used to compute the strength and direction of the traversability field. As shown in Fig. 4, T^* is divided into n sectors, each of which has an angular resolution of α (α is chosen such that $360/\alpha$ is an integer). Each sector k, for $k = 0, \cdots, n-1$, corresponds to a discrete angle $\rho = k\alpha$ in the polar coordinate.

The direction of the traversability field produced by cell (i, j) is calculated by

$$\beta_{i,j} = \tan^{-1}\frac{y_j - y_o}{x_i - x_o} \tag{4}$$

and the magnitude is

$$m_{i,j} = \tau_{i,j}^2 (a - b d_{i,j}) \tag{5}$$

where

x_i, y_j are the coordinates of cell (i, j),

x_o, y_o are the coordinates of the RGC,

$\tau_{i,j}$ is the TI value of cell (i, j),

a, b are positive constants, and

$d_{i,j}$ is the distance between cell (i, j) and the RGC.

In (5), $\tau_{i,j}$ is squared such that the impact of a cell with a small/large TI value is diminished/magnified. $m_{i,j}$ is proportional to $-d_{i,j}$. Therefore, cells with non-zero TI values result in larger vector magnitudes when they are closer to the robot and smaller ones when they are farther away. It should be noted that a and b are chosen such that $a - bd_{\max} = 0$, where $d_{\max} = \sqrt{2}(w_s - 1)/2$ is the distance between the farthest cell of T^* and the RGC. This arrangement guarantees $m_{i,j} = 0$ for the four farthest cells (the four vertices) of T^*. The cells (i, j) in T^* are assigned to the k^{th} sector according to

$$k = \text{int}(\beta_{i,j}/\alpha). \tag{6}$$

For sector k, the Magnitude of Traversability Field (MTF) produced by all of its cells is calculated by

$$h_k = \sum_{i,j} m_{i,j}. \tag{7}$$

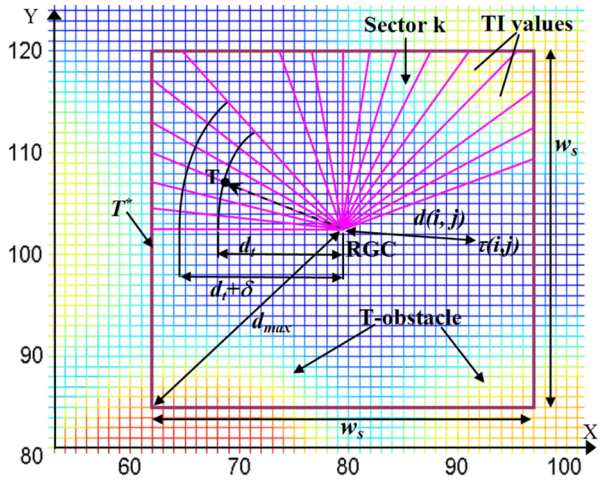


Fig. 4. Transformation of a traversability map into a histogram: X and Y axes represent the world coordinates. For simplicity, each sector is drawn in 10° and only some of the sectors are shown.

In this implementation α = 5°, therefore, there are 72 sectors. The MTF of each sector represents the overall difficulty moving in the corresponding direction and it is represented in the form of histogram.

Figure 5 shows the TFH for the momentary situation of Fig. 2(b) where the RGC is located at cell (150, 144) (labeled q). The parallelepiped object produces large MTFs in sectors 0-13 and sectors 65-71 (i.e., -35°–65°) because it is high in elevation. Cone 1 generates small MTFs in sectors 19-31 (i.e., 95°–155°) since its low-profile tip faces the robot, whereas cone 2 produces relatively larger MTFs in sector 50-57 (i.e., 250°–285°) because its base (with its higher elevation) faces the robot. This exemplifies how TFH correctly reflects the overall traversability of each sector in region T^*.

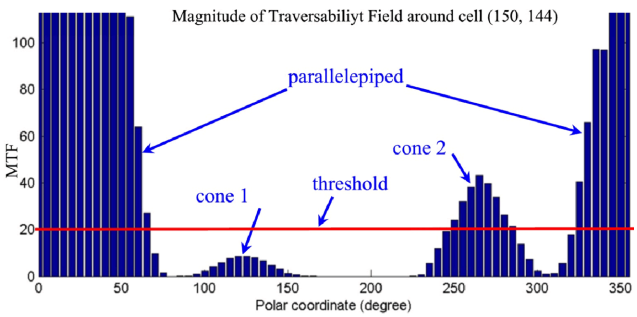


Fig. 5. The traversability field histogram when the RGC is at point q in Fig. 2(b)

4.2 Steering Control

When the robot travels across terrain, the TFH changes from instance to instance. The objective of the ON algorithm is to steer the robot into a direction with a lower MTF, while still maintaining a direction that is close to the target direction. As can be seen from Fig. 5, a TFH typically has "hills" (sectors with high MTFs) and "valleys" (sectors with low MTFs). A candidate valley is defined as a cluster of consecutive sectors with MTFs below a certain threshold, e.g., Fig. 5 has two candidate valleys. The candidate valley eventually selected to determine the robot's next heading direction is called the wining valley. In [23] the candidate valley closest to the target direction is selected to determine the robot moving direction to avoid obstacles in 2-D environment. This approach, called "the-closet-valley-wins" scheme in this paper, may result in trap situation for ON in a 3-D (more accurately, 2.5-D) environment.

To overcome this problem, the concept of "motion-concept" (detailed in Section 4.2.2) is proposed and used to decide the robot motion. Assuming that the robot's target is (x_t, y_t), the sector number k_t of the target vector is calculated by (4) and (6) (x_i and y_i are replaced by x_t and y_t in this case). We denote the i^{th} candidate valley by $v_i = [k_i^R, k_i^L]$ where k_i^R and k_i^L represent the sector number of the right and left border of candidate valley v_i, respectively. The width of the valley s_i is $s_i = k_i^L - k_i^R + 1$ sectors. A valley is a wide valley if $s \geq s_{\max}$ (in this paper, $s_{\max} = 12$), and otherwise, it is a narrow one. A winning valley is denoted v_w. The robot's next heading direction is calculated by

$$k_h = \begin{cases} k_w^R + \min(s_w, s_{\max})/2 & \text{if } |k_t - k_w^R| \leq |k_t - k_w^L| \\ k_w^L - \min(s_w, s_{\max})/2 & \text{otherwise} \end{cases} \quad (8)$$

where k_w^R and k_w^L represent the sector number of the right and left border of v_w, respectively. The border determining the next heading direction is called the winning border, and s_w is the width of v_w. According to (8), the robot moves along the center of the valley if the winning valley is narrow. Otherwise, it moves along a direction with $s_{max}/2$ sectors away from the winning border.

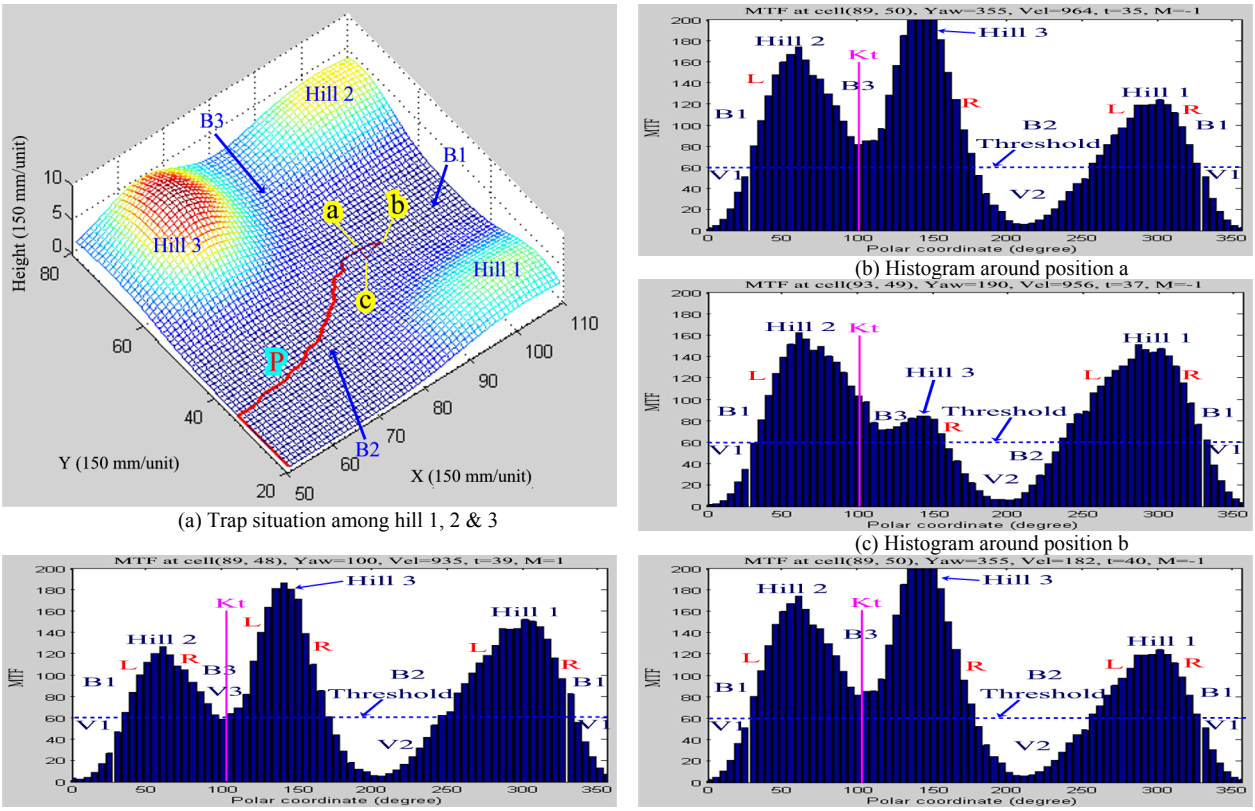
4.2.1. Trap conditions

A trap situation may be defined as a condition in which the robot loops around endlessly in the same area. There are two types of trap conditions: (1) environmental traps and (2) algorithmic traps. An environmental trap is a trap created by the terrain conditions, for example, a cul-de-sac. A local ON algorithms are inherently susceptible to such traps, unless combined with a global capabilities or heuristic recovery routines. An algorithmic trap is a shortcoming of the path planning algorithm: the robot fails to find way out of the trap, even though no environmental trap exists. Unless enhanced with the algorithm explained below, the-closest-valley-wins scheme can fall into algorithmic traps, as illustrated by the following example.

Figure 6 illustrates a typical trap scenario. The robot started at cell (50, 20) and navigated along trajectory P toward the target at cell (60, 180). At the 35th time step, it moved to position a in Fig. 6(a). The TFH at a is depicted in Fig. 6b. The three hills in Fig. 6a produced three peaks in Fig. 6b, while the three passages—B1, B2 and B3—generated three valleys, which are also denoted as B1, B2 and B3 in Fig. 6(b). Since the threshold was set at 60, only B1 and B2 created candidate valleys V1 and V2, of which V1 is the winning valley because it is closer to the target direction k_t. Therefore, the robot's next heading (yaw) is pointed at 355°. Consequently the robot moved in this heading direction until it reached position b at the 37th time step. The motion from a to b moves the robot away from Hill 3 and thus lowers the MTFs of Hill 3 as shown in Fig. 6(c). The changes in MTFs widen valley V2 at its right border and made V2 the closest one. Therefore, the winning valley is now V2 and the robot steers to a 190° heading (notice that in this case, the bottom of B3 is very close to the threshold). The robot travels at this direction and arrives at c after two time steps. This motion moved the robot away from Hill 2 and close to Hill 3. The MTFs of Hill 3 increase and that of Hill 2 decrease as shown in Fig. 6(d). At position c, passage B3 produces a MTF slightly below the threshold since the robot faces the passage directly. This creates a new candidate valley V3 which turns out to be the winning valley, since it is the closest to k_t. Therefore, the robot steers to a 100° heading and moves back to position a. Because the robot moves closer to Hills 2 and 3, their MTFs increase and valley V3 is removed from the list of candidate valleys. The robot steers back to a 355° heading. This way, the robot is trapped and continues to drive around in a tight loop.

4.2.2. Motion-context

In this paper, the concept of motion-context it is used to overcome algorithmic traps. The robot's motion-context at time step *t* is defined as



(a) Trap situation among hill 1, 2 & 3

(b) Histogram around position a

(c) Histogram around position b

(d) Histogram at position c

(e) Histogram around position d

Fig. 6. Trap situation caused by the-closest-valley-win scheme: Kt – target sector, L, R – left / right border of a candidate valley, M – motion-context, t – time step, Vel – velocity of the robot in 3D space, Yaw – yaw (heading) angle. Note: valley V1 comprises two parts, one within the 1st phase and the other one within the 4th phase. v_{max}=1 m/s.

$$\mu = sign(k_h - k_t), \qquad (9)$$

where k_t and k_h are the sector number of the target vector and the robot's heading vector at time step *t-1*, respectively. $\mu = 0$ means that the robot is moving straight toward the target while $\mu = -1$ and $\mu = 1$ indicate that the robot is deviating from the target direction because of an avoidance maneuver initiated by a right-turn or left-turn, respectively. The deviations of the i^{th} valley's left and right boarders from the target vector are defined as $D_i^L = | k_t - k_i^L |$ and $D_i^R = | k_t - k_i^R |$. Their minimum values are denoted by $D_{\min}^L$ and $D_{\min}^R$, respectively. There are several scenarios with different numbers of obstacle in T^*:

1) Multiple valleys (Fig. 7(a) and (b)): This case most likely produces narrow valleys. Figure 7(a) shows the scenario where the target vector lies inside a MTF peak. In this case, $D_{\min}^L$ and $D_{\min}^R$ correspond to the left border of valley *j* and the right border of valley *j+1*, respectively. Figure 7(b) illustrates the scenario where the target vector lies inside a valley. In this case, $D_{\min}^L$ and $D_{\min}^R$ are corresponding to the left and right borders of valley *j* (the winning valley), respectively.
2) Single valley (Fig. 7(c)): This condition usually generates a wide valley. In this case, $D_{\min}^L$ and $D_{\min}^R$ correspond to the left and right border of the wide valley. Under this condition, too, the target vector may be located within the peak or within the valley. However, we only show the former here for conciseness.

Table 1: Rules for determining the wining valley and border

If \ μ	μ=1	μ=0	μ=-1
$D_{\min}^L > D_{\min}^R$	k_{j+1}^R	k_j^L	k_j^L
$D_{\min}^L \le D_{\min}^R$	k_j^L	k_j^L	k_j^L

The subscripts in this table are applicable to Fig. 7(a) only. For the case in Fig. 7(b), all subscripts are *j*; while for Fig. 7(c), all subscripts are 1.

We use Table I to select the winning valley and the winning border. This decision table gives the robot a tendency to follow the contour of an obstacle at its left and may even cause it to circle the obstacle and thus miss the target. Therefore, we need an exit condition. Once the exit condition is met, the robot is forced to move straight toward the target direction and thus reset the motion-context. In order to do this, we define the target direction to be "free" if each sector of the cluster of sectors $[k_t - s_{\max}/2, k_t + s_{\max}/2]$ lies in a candidate valley, i.e., the MTF of each sector is smaller than the threshold value. The steering control algorithm first checks whether the target direction is free. If so, then the robot's next heading

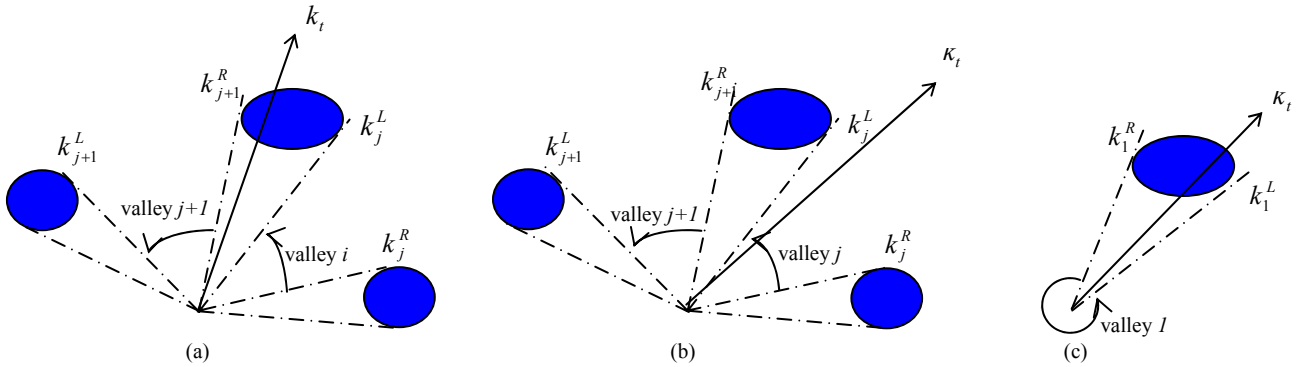


Fig. 7. Situations with a varying number of candidate valleys and different locations of the target vectors: as a peak in TFH corresponds to an obstacle at that direction, we use an obstacle to represent a peak, and the passage between obstacles to represent a candidate valley for the sake of intuitiveness. The straight lines represent the left and right borders of the valley. They are not necessarily tangents to the obstacle.

is pointed at the target direction. Otherwise, the scheme in Table I is applied to determine the winning border and thus the steering direction.

Close examination of Fig. 6 finds that the main reason for the trap situation is that the-closest-valley-wins scheme produces a steering angle of 190° from the histogram in Fig. 6(c) when the robot was at position b. Applying the rules in table 1 to the same navigation task, we may find that at position b—with the motion-context μ = -1—the robot chooses valley V1 as the winning valley and uses the left border to determine the steering even if valley V2 is closer to the target vector ($D^L_{\min} > D^R_{\min}$). This way, the robot continues to move in the same valley, hence maintained the continuity of motion and retains the motion-context. This prevents the robot from moving backward to position c and going into an endless loop just as the-closest-valley-wins scheme does. The full trajectory determined by our scheme using the motion-context is shown in Fig. (8).

4.3 Speed Control

The robot runs initially at its maximum speed v_{max}. It maintains this speed unless forced by the TFH algorithm to a lower value, which is determined at each time step as follows:

The MTF in the current direction of travel is denoted as $h_c^{'}$. A large value of $h_c^{'}$ indicates that a potential untraversable terrain lies ahead. Therefore, a reduction in speed is required. In this study, speed is reduced inversely proportional to the MTF value in the momentary direction of travel:

$$v^{'} = v_{\max}(1 - \frac{\min(h_c^{'}, h_m)}{h_m}) \tag{10}$$

where h_m is a constant, which is empirically determined to produce a sufficient reduction in speed. A reduction in speed is also required when the robot approaches the target. The robot speed taking into this consideration is computed by

$$v^{"} = \frac{v^{'} \min(d_t, d_m)}{d_m} \tag{11}$$

where d_t is the distance between the target and the RGC, and d_m is a constant (1.5 m in this work). Note that $v^{"}$ is actually the projection of the robot speed V on the X-Y plane. It produces a movement l_{xy} (in X-Y plane) along the robot's next heading direction which moves the RGC to $(x_o^{'}, y_o^{'})$. The robot's speed in 3-D space is calculated by

$$V = \frac{v^{"} l_{xyz}}{l_{xy}} \tag{12}$$

where l_{xyz} is the 3-D distance between the current RGC position at (x_o, y_o, z_o) and the next one at $(x_o^{'}, y_o^{'}, z_o^{'})$.

4.3 Convergence proof of the TFH algorithm

In summary, the TFH algorithm functions as follows (for simplicity, we describe the algorithm in the X-Y plane):

1) Start the robot with $v^{"} = v_{\max}$ and μ=0.
2) If the target is reached, then stop the robot; otherwise go to Step 3.
3) Calculate the TFH. If the target direction is free, then move the robot straight toward the target with v" Otherwise, go to Step 4.
4) Apply Table I to determine the winning valley and the winning border and thus the steering direction. Move the robot along that direction at speed v" . Go to 2.

The above algorithm globally converges at the target if the terrain is free of environmental traps. Such a terrain typically consists of convex obstacles. The convergence of the algorithm based on convex obstacles is proven in this section.

Lemma: the TFH algorithm causes the robot to follow an obstacle's boundary with a finite clearance and eventually guides the robot to the target.

Proof: first, the TFH method forces the robot to maintain a certain clearance from an obstacle. In case that the wining valley is a narrow one, the robot always moves in the center of the valley. As for a wide winning valley, if the robot is moving closer to an obstacle, then the number of sectors occupied by the obstacle grows and the MTFs of these sectors grow, too. In other words, the width of the winning valley decreases, and the borders of the winning valley move away

from the obstacle. According to (8), this pushes the robot away from the obstacle. On the other hand, if the robot is moving farther away from the obstacle, the borders of the winning valley move closer to the obstacle. This allows the robot to approach the obstacle more closely. These two contradictory effects maintain the robot on a path with a certain clearance from the obstacle.

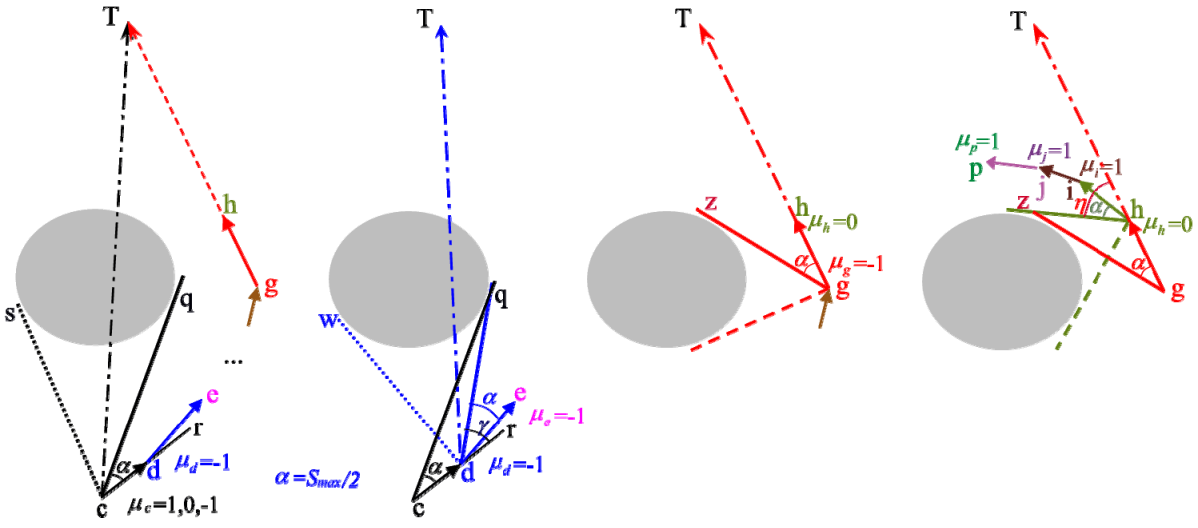
Second, the robot always moves straight toward the target unless a MTF peak appears in its path. When a MTF peak is 'seen', the robot steers to the left or right border of the winning valley with a certain clearance from the border. The winning valley may be a wide valley (Fig. (8)) or a narrow one (Fig. 7(a)).

(1) Wide valley (Fig. (8)): without losing generality, we assume that a peak occurs in the histogram when the robot is at c (c is called a divergence point). Since $D^L_{\min} \le D^R_{\min}$, the robot turns right (Fig. 8(a)) regardless of the motion-context μ. This results in μ=-1, thus the robot always uses the left border to derive its next heading until conditions μ=1 and $D^L_{\min} > D^R_{\min}$ are met. At point *c*, the robot's next heading is along $\overrightarrow{cd}$ which is offset α from the left border *cq* of the valley. This deviates the robot's movement vector from the target vector $\overrightarrow{cT}$ and it moves the robot to d. Since γ is the exterior angle of the triangle Δcdq (Fig. 8(b)), γ >α. This means the next movement vector $\overrightarrow{de}$ must stay inside $\angle qdr$. This rotates the robot's movement vector counter-clockwise around the obstacle (i.e., it tries to align the movement vector with the target vector). This counter-clockwise avoidance maneuver maintains $D^L_{\min} \le D^R_{\min}$ until point p (Fig. 8d) where the winning border switches from the left border to the right border (because conditions μ=1 and $D^L_{\min} > D^R_{\min}$ are met). We call point p a switch point. This means that the robot always follows the boundary of the obstacle counter-clockwise before it reaches the switch point.

(2) Narrow valleys: In this case (e.g., Fig. 7(a) and Fig. 7(b)), there is no substantial difference except that the heading of the robot now constantly aims at the center of the valley (i.e., the center of the passage between the obstacles). As the robot moves, the width of the valley grows. Eventually, the valley becomes a wide one and this brings the robot into the wide valley situation, as proved in (1).

Finally, as the robot's motion aligns the movement vector with the target vector, a point h (Fig. 8c) always exists on the robot's path from c to T such that $\mu = 0$. This is due to the continuity in space and motion. At the point immediately before h (i.e., point g), the cluster of sectors $[k_t - s_{\max}/2, k_t + s_{\max}/2]$ lies in a wide valley. This is because at g the target vector overlaps the movement vector; thus, it is also $s_{max}/2$ away from the winning border. This means that the target is free at g. Therefore, the robot abandons the boundary following at g and moves straightly toward the target. This resets the motion-context to zero.

Note that if the condition at c is $D^L_{\min} > D^R_{\min}$, then the



(a) (b) (c) (d)

Fig. 8. Situations with a wide candidate valley and $D^L_{\min} \le D^R_{\min}$: μ_x represents the motion-context at position *x*. The dotted line represents the right border of the valley and the solid line represents the left border of the valley. Since the TFH algorithm maintains the movement vector in the direction with an angle α away from the winning border, we may attach the movement vector to the wining border. The winning border and the movement vector form a "rigid body." The TFH algorithm rotates this rigid body counter-clockwise such that it gradually aligns the movement vector with the target vector. (a) shows the target vector $\overrightarrow{cT}$, left boarder *cq* and right boarders *cs* of the valley, and next movement vector $\overrightarrow{cd}$ with the robot at point a. The robot path determined by the TFH algorithm is shown as $c \to d \to e \to \ldots \to g \to h \to \ldots T$. (b)-(d) depict the target vector, left and right boarders, and next movement vector when the robot is at positions d, g, and h, respectively.

robot will turn left. The lemma can be proven in the similar way. However, it is omitted for conciseness. It is apparent that the robot may miss the target without checking the exit condition. This is depicted in Fig. 8(d) when the robot is at h. Since the exterior angle η of triangle Δghz is bigger than α, the robot's next heading direction is $\overrightarrow{hi}$ without checking the exit condition. In this way, the robot continues to follow the obstacle boundary as $g \to h \to i \to j \to p$ and miss the target; and it moves backwards at the switch point p.

Theorem: The TFH algorithm guarantees the convergence of an obstacle negotiation task. In other words, the obstacle negotiation algorithm always terminates with a finite-length path if such a path exists.

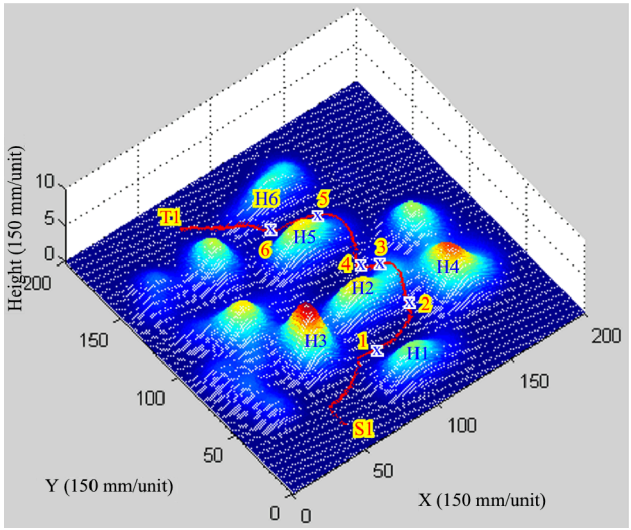
Proof: We first consider the wide valley case. If there never is a divergence point, the robot moves from starting point S straight toward the target T. The length of the robot's path $l(S,T)$ equals $d(S,T)$ –the straight-line distance from S to T; and it is finite. If c is a divergence point, the robot follows the obstacle boundary at a finite distance until the exit condition is met at g. According to the lemma, the length of the path from c to g $l(c,g)$ is finite. After the robot departs from the obstacle-hugging path at g and moves straight toward the target, the path length from g to T is $l(g,T) = d(g,T)$, which is finite. Therefore, the total path length $l(S,T) = l(S,c) + l(c,g) + l(g,T)$ is finite.

Next, we consider the narrow-valley case where the path comprises the portion $l(S,c)$ (from the start to the divergence point), $l(c,e)$ (from the divergence point to a point e where the valley changes to be a wide one), $l(e,g)$ (from e to the exit point g) and $l(g,T)$ (from g to the target). Obviously, $l(S,c)$ and $l(g,T)$ are the length of straight-line segments, hence they are finite. According to the lemma, $l(e,g)$ is also finite. During the movement from c to e, the robot always moves along the center of the passage between the obstacles, therefore, the length of path $l(c,e)$ is finite. Summing up all the cases, the TFH algorithm terminates with a finite-length path. The theorem is thus proven.

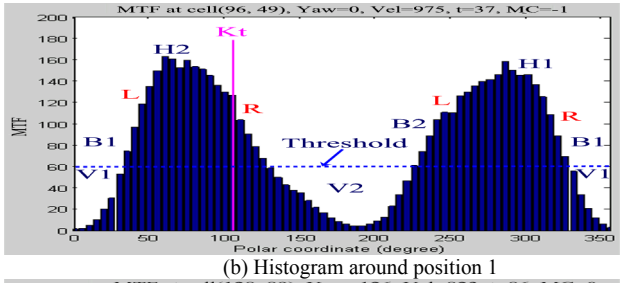
There might be a difficult case where the target is located very close to an obstacle. In this case, the target direction is never free—meaning the robot never reaches the target. In this paper, the concept of "virtual valley" is proposed to remedy this problem. As depicted in Fig. 4, the target T is very close to the hill on its left. The MTFs in sector k_t (corresponding to T) and the neighboring sectors is nonzero and they grow as the robot moves closer to T. This eventually keeps the robot from reaching T. To resolve this problem, a cluster of sectors $[k_t - n, k_t + n]$ (n=s_{max}/2) is constructed. The MTFs of all cells that are inside this cluster and whose distances to the RGC are larger than $d_t + \delta$ are ignored. Here, d_t is the distance between the target and RGC and δ is a safety factor that maintains a clearance between the robot and the obstacle when the robot reaches the target. This treatment then creates a wide valley in $[k_t - n, k_t + n]$ if the terrain inside this area is passable. We call this a "virtual valley." This virtual valley creates a free target direction ahead of the robot, and thus guides the robot toward the target.

5. Simulation and Experimental Results

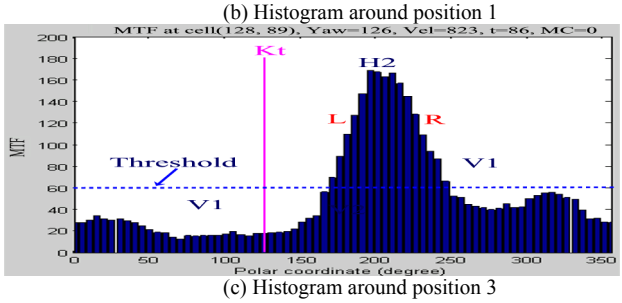
The ON simulator is used to verify the TFH algorithm. Figure 9(a) shows a typical run over a simulated terrain. The run started at S_1, terminated at target T_1, and is generated



(a) Terrain map and the robot trajectory



(b) Histogram around position 1



(c) Histogram around position 3

Fig. 9. A navigation task from S1 to T1: Kt – target sector; L/R – left / right border of a candidate valley; M – motion-context, t – time step; Vel – velocity of the robot in 3D space; Yaw – yaw (heading) angle. The histogram at position 2, 4 and 5 are omitted for conciseness. v_{max}=1 m/s.

in real-time. This run is as the same as that in Fig. 7 except that the motion-context is used here.

The robot first steers to the passage between hills H1 and H3 and moved to point 1. The TFH at this point is displayed in Fig. 9(b). Since the motion-context is -1, the left border of valley V1 is used to determine the heading (0° in this case). The robot maintaines its heading and goes through the passage between hills H1 and H2.Compared with the situation in Fig. 7, the use of motion-context resolves the trap problem. The robot traverses moderate terrain between hills H2 and H4 by following the boundary of hill H2. When the robot reaches point 3, the target direction is free. As depicted in Fig. 7(c), the cluster of sectors $[k_t - s_{\max}/2, k_t + s_{\max}/2]$ lies in a wide valley V1. Sice the exit condition is met, the robot leaves hill H2 and moves straight toward the target (thus M=0 in Fig. 7(c)). It maintains this heading and zero motion-context until it arrives at the divergence point at position 4, where hill H5 entered region T^* and creates a peak in the TFH. The robot follows the boundary of Hill H5 until position 6. At this position, the target direction was free again and the robot is forced to move straight toward the target. Finally, the robot comes to a full stop at T1.

A number of simulation runs are carried out on widely differing real and simulated terrain maps. In all cases, the robot achieves its targets with finite-length paths without being trapped. In all of these runs, the roll and pitch angles of the robot are within [-6°, 7°] and [-8°, 9°], respectively; This means that the TFH algorithm successfully guides the robot in traversing relatively moderate terrain.

The TFH algorithm is a local method. It may be susceptible to an environmental trap like other local methods. A typical environmental trap is a cul-de-sac as shown in Fig. 10. The condition that the robot may successfully escape from a trap is that the distance between each cell on the concave boundary and the RGC must be sufficiently small when the robot enters the cul-de-sac. If this condition is met, the entire concave boundary is inside T^* and appears to be a peak in the histogram such that the robot steers away from the cul-de-sac.

Simulation runs in the environment in Fig. 10 show that the robot avoids the obstacle without entering the U-shaped boundary if $\eta \leq 32$ cells (η is the depth of the concave boundary); and the robot went into U-shaped boundary but it escaped from the obstacle as long as $\eta < 50$ cells. These results reveal that the TFH algorithm can deal with environmental trap to some extent. The robot may be trap if $\eta \geq 50$. Using a larger T^* (i.e., a bigger value of w_s) will get the robot out of the trap. But this increases the computational cost. A viable solution is to use a dynamic w_s whose value is adjusted according to the environment condition.

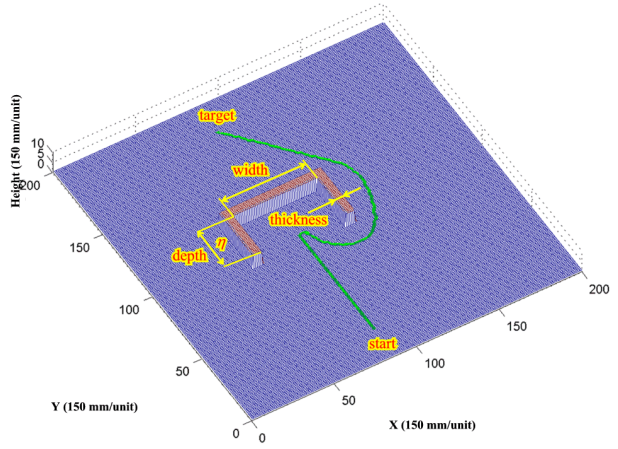


Fig. 10. A concave obstacle scenario: depth=49 cells, width=85 cells, thickness=9 cells. v_{max}=1 m/s.

The TFH algorithm has been tested by experiments using the Segway Robotic Mobility Platform (SRMP) on flat and non-flat terrain. The SRMP is depicted in Fig. 11. The LRF has a tilt down angle of -14° from the robot's equipment plate. The LRF's angular resolution is set at 1° and it sends out 181 range data (covering a 180° field of view) every 13.3 ms at 500k baud. The PE system is a self-contained unit which estimates the robot pose (roll, pitch, yaw angles and XYZ coordinates) in real-time. The onboard computer (VIA EPIA MiniITX) uses RTLinux as the operating system. A real-time thread acquires the laser and pose data and buffers these data. The terrain mapping module fetches the data in a batch (8 scans) and registers them in the terrain map. Then the TTA module analyzes the terrain; and the TFH algorithm computes the robot motion commands and sends the commands to the robot. The above process iterates every 106 ms until the target is reached. The terrain data are sent to a remote computer to visualize the mapping and navigation process in real-time. In the experiments, the maximum speed of the robot was set to 1 meter per second. A terrain patch of 5 cells ×5 cells is used for TTA.

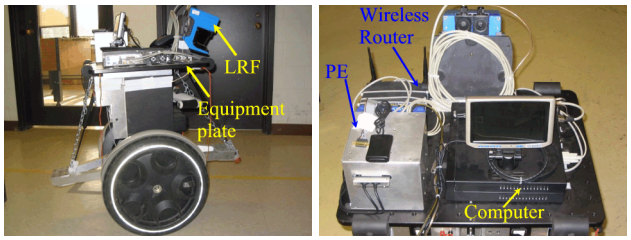
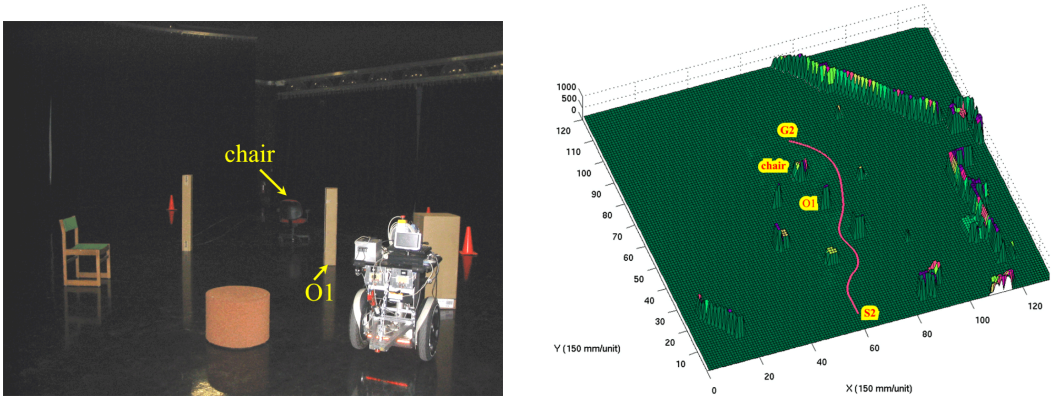


Fig. 11. The SRMP: The main components for terrain mapping are the PE and the LRF that are connected to the on-board computer. The wireless router is used for the access of the off-board computer to the mapping system.

Figure 12 shows one of the experiments on flat ground. Figure 12(a) is a picture of the obstacle course in an indoor environment. The terrain map and robot trajectory generated by the system are displayed in Fig. 12(b). It is clear that the robot maneuvers itself among obstacles in a smooth path. The result indicates that the TFH algorithm can handle conventional obstacle avoidance task successfully.

Figure 13 demonstrates an ON experiment on non-flat terrain that consists of man-made "ramps" and "curbs." Figure 13(a) is a snapshot of the video clip taken during the experimental run. It renders the scenario just after the robot climbs up the ramp. Figure 13(b) is the screen

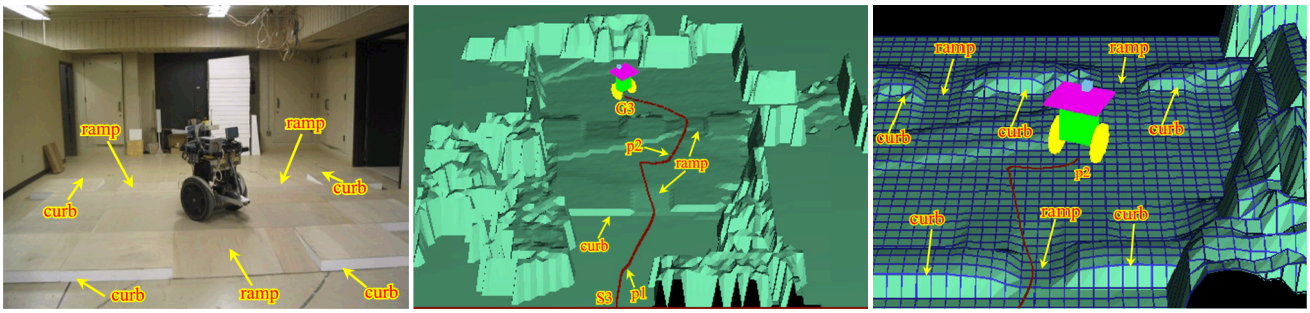


(a) Indoor obstacle course (b) Terrain map and robot trajectory

Fig. 12. Experimental run on flat ground (the root is also able to deal with the obstacle course when a maximum speed of 3 m/s is used)

snapshot of the terrain map and the robot trajectory in the remote display when the robot arrives at target G3; and Fig. 13(c) is the traversability map when the robot was at position p2. In Fig. 13(c), curbs appear to be obstacles in the traversability map as their cells have large TIs; while ramps looks like passage since their cells possess small TIs. The TFH algorithm guides the robot to the target G3 via the right ramp.

In conclusion, an obstacle negotiation method is proposed for mobile robot navigation on unknown terrain. The algorithm first employs an analytical terrain traversability analysis method to transform the local terrain map into a traversability map. It then transforms the traversability map into a traversability field histogram, from which the velocity and the steering commands are determined. The motion-context is proposed to overcome



(a) Non-flat terrain (b) Terrain map and the robot trajectory (c) Traversability map surrounding p2

Fig. 13. Experimental run on non-flat terrain

6. DISCUSSIONS AND CONCLUSIONS

The proposed TFH algorithm is computationally efficient. The operation count of the TTA algorithm for each cell is $C_{tta}=24N+113$. The total cost of transforming a local terrain map into a traversability map is $C_{tta}\times M^2$, where M is the side length (in terms of cells) of the local terrain map T^*. In order to transform the traversability map into a TFH, $10M^2-k$ operations are required, where $k=72$ is the number of sectors of the histogram. The total cost of the TFH algorithm is then $C_{tta}\times M^2+10M^2-k$. In fact, we only need to perform TTA on new cells as they enter region T^*. The worst case is that the robot move diagonally and the motion causes $2M-1$ cells enter region T^*. In this case, M^2 in the first term of the cost expression should be replaced by $2M-1$. Since M=61 and N=5 is used in the experiment, the TFH algorithm requires about 65,331 operations to determine the motion commands and it takes the onboard computer less than 2 ms in the experiment. This means that the TFH is suitable for real-time implementation.

trap situations with the-closest-valley-wins scheme. The algorithm guarantees to terminate the obstacle negotiation task at the target with a finite-length path for a terrain with convex obstacles, and is able to handle concave obstacles to some extent. It is computationally efficient and is practical for real-time implementation.

Appendix A Plane Fitting

The plan-fitting method is to find a least-squares plane to a set of data points $\mathbf{p}=\{\mathbf{u}_i\}$, $i=1,2,\cdots,N$. We define the following notation:

$\mathbf{u}=(x,y,z)^T$: A 3-D point in the plane

$|\bullet|$: The Euclidean norm, e.g., $|\mathbf{u}|=\sqrt{x^2+y^2+z^2}$

$\mathbf{u_i}=(x_i,y_i,z_i)^T$: The i^{th} data point

$\bar{\mathbf{u}}=(\bar{x},\bar{y},\bar{z})^T$: The centroid of the data,

$$\frac{1}{N}\left(\sum_{i=1}^{N}x_i,\sum_{i=1}^{N}y_i,\sum_{i=1}^{N}z_i\right)$$

$\mathbf{n} = (a,b,c)^T$: Direction cosines of the normal to the plane

P: The *N*×*3* matrix containing the *N* data points of the terrain patch: $\begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \vdots & \vdots & \vdots \\ x_N & y_N & z_N \end{bmatrix}$

∇_{χ} : The gradient of a scalar function with respect to $\boldsymbol{\chi}$, e.g., $\nabla_{\mathbf{u}} f(x,y,z) = (\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z})^T$

The plane containing $\mathbf{u_c} = (x_c, y_c, z_c)^T$ and the direction cosines of normal $\mathbf{n}$ is given by

$$\mathbf{n} \cdot (\mathbf{u} - \mathbf{u_c}) = 0 . \tag{13}$$

The distance between $\mathbf{u_i}$ and the plane is calculated by

$$d_i = d(\mathbf{u_i}) = d(\mathbf{u_i}, \mathbf{u_c}, \mathbf{n}) = \mathbf{n} \cdot (\mathbf{u_i} - \mathbf{u_c}) \tag{14}$$

The plane-fitting problem is to find a least-squares plane, i.e., to determine $\mathbf{u_c}$ and $\mathbf{n}$ such that the following objective function is minimized:

$$J(\mathbf{u_c}, \mathbf{n}) = \sum_{i=1}^{N} d_i^2 = \sum_{i=1}^{N} [\mathbf{n} \cdot (\mathbf{u_i} - \mathbf{u_c})]^2 \tag{15}$$

At the least-squares solution, $\nabla_{\mathbf{u_c}} J = 0$, which yields $\sum_{i=1}^{N} \mathbf{n} \cdot (\mathbf{u_i} - \mathbf{u_c}) = 0$. Multiplying by 1/N give

$$\frac{a}{N}\sum_{i=1}^{N}(x_i - x_c) + \frac{b}{N}\sum_{i=1}^{N}(y_i - y_c) + \frac{c}{N}\sum_{i=1}^{N}(z_i - z_c) = 0 \tag{16}$$

Distributing the summation produces

$$a(\bar{x} - x_c) + b(\bar{y} - y_c) + c(\bar{z} - z_c) = 0 \tag{17}$$

This means $d(\bar{\mathbf{u}}) = \mathbf{n} \cdot (\bar{\mathbf{u}} - \mathbf{u_c}) = 0$, i.e., the centroid $\bar{\mathbf{u}}$ lies on the least-squares plane. Therefore, letting $\mathbf{u_c} = \bar{\mathbf{u}}$, the fitted plane can be rewritten as $\mathbf{n} \cdot (\mathbf{u} - \bar{\mathbf{u}}) = 0$. The remaining problem is to determine the direction cosines $\mathbf{n}$, which can be found by solving the constrained minimization problem, namely, minimizing $J(\bar{\mathbf{u}}, \mathbf{n})$ subject to the constraint $|\mathbf{n}| = 1$. Letting $G = |\mathbf{n}|^2 - 1$, the problem is to minimize $J(\bar{\mathbf{u}}, \mathbf{n})$ subject to the constraint $G = 0$. According to the Lagrange Multiplier methods [25], the minimum occurs at a point where $\nabla_{\mathbf{n}} J = \lambda \nabla_{\mathbf{n}} G$ for some real numbers λ. Since $\nabla_{\mathbf{n}} G = 2\mathbf{n}$ and $\nabla_{\mathbf{n}} J = 2(\mathbf{Q}^T\mathbf{Q})\mathbf{n}$ where

$$\mathbf{Q} = \begin{bmatrix} x_1 - \bar{x} & y_1 - \bar{y} & z_1 - \bar{z} \\ x_2 - \bar{x} & y_2 - \bar{y} & z_2 - \bar{z} \\ \vdots & \vdots & \vdots \\ x_N - \bar{x} & y_N - \bar{y} & z_N - \bar{z} \end{bmatrix}. \tag{18}$$

This yields the eigen-problem given by

$$(\mathbf{Q}^T\mathbf{Q})\mathbf{n} = \lambda \mathbf{n} , \tag{19}$$

i.e., $\mathbf{n}$ and λ are the eigenvector and the eigenvalue of matrix $\mathbf{Q}^T\mathbf{Q}$, respectively. According to the Singular Value Decomposition (SVD) [26], $\mathbf{n}$ is also the singular vector of $\mathbf{Q}$ while λ is the square of the singular value of **Q.** This allows us to obtain $\mathbf{n}$ by the SVD.

Details for computing the singular values and the singular vectors are explained in [26]. As the SVD produces three singular vectors, we must select the correct one for $\mathbf{n}$. Equation (19) can be written as

$$\begin{aligned} (x_i - \bar{x})(\mathbf{n} \cdot \mathbf{u_i}) &= \lambda a \\ (y_i - \bar{y})(\mathbf{n} \cdot \mathbf{u_i}) &= \lambda b \\ (z_i - \bar{z})(\mathbf{n} \cdot \mathbf{u_i}) &= \lambda c \end{aligned} \tag{20}$$

Multiplying these three equations by *a*, *b* and *c*, respectively, and summing up the resulting equations yield $\sum_{i=1}^{N} [\mathbf{n} \cdot (\mathbf{u_i} - \bar{\mathbf{u}})]^2 = \lambda$. According to (15), the left value is just the objective function, $J(\bar{\mathbf{u}}, \mathbf{n})$ which equals $|\mathbf{d}|^2$, where $\mathbf{d} = (d_1, d_2, \cdots, d_n)^T$ is the plane-fit residual and $|\mathbf{d}|$ is its Euclidean norm. Therefore, the correct eigenvector (i.e., the normal to the fitted plan $\mathbf{n}$) for the least squares solution is the one corresponding to the smallest eigenvalue. When using the SVD, the singular vector corresponding to the smallest singular value is chosen for $\mathbf{n}$. Since $\mathbf{u_c}$ and $\mathbf{n}$ in (13) have been determined, the fitted plane is then found.

In summary the process of finding the least-squares plane comprises these steps:

1) Calculate the centroid $\bar{\mathbf{u}}$ of the data points in the terrain patch **p** and form the matrix **Q** by (18).
2) Calculate the eigenvalues λ_i of the matrix $\mathbf{Q}^T\mathbf{Q}$ and select the minimum eigenvalue, $min(\lambda_i)$.
3) Calculate the eigenvector that corresponds to $min(\lambda_i)$ and assign this eigenvector to $\mathbf{n}$.
4) The least-squares plane is then given by $\mathbf{n} \cdot (\mathbf{u} - \bar{\mathbf{u}}) = 0$ and the Euclidean norm of **d** equals the square root of $min(\lambda_i)$.

In the C++ implementation of the above process, the Householder Method [27] is used to reduce matrix $\mathbf{Q}^T\mathbf{Q}$ to tridiagonal form and the Tridiagonal QL Implicit Algorithm [27] is used to find the eigenvalues and the eigenvectors. The operation count of the former is $4K^3/3$ while that of the later is about $3K^3$, where K=3 is the dimension of matrix $\mathbf{Q}^T\mathbf{Q}$. The overall complexity in solving the eigen problem is $13K^3/3$. The preprocessing to form the matrix requires $6N+(2N-1)K^2$ operation, where N is the number of data points. The total workload of the plane fitting algorithm is thus $13K^3/3+6N+(2N-1)K^2=24N+108$ operations.

ACKNOWLEDGEMENT

The author would like to thank the anonymous reviewers and editor for their comments, which help him

improve this paper significantly. He would also like to thank Dr. Johann Borenstein for his support on this work.

Reference

1 S. Betgé-Brezetz, P. Hebert, R. Chatila, and M. Devy, "Uncertain map making in natural environments," *Proc. IEEE International Conference on Robotics and Automation*, Minneapolis, MN, 1996, pp. 1048-1053.

2 D. S. Apostolopoulos, M. D. Wagner, B. Shamah, L. Pedersen, K. Shillcutt, and W. L. Whittaker, "Technology and field demonstration of robotic search for Antarctic meteorites," *International Journal of Robotics Research*, vol. 19, no. 11, pp. 1015-1032, 2000.

3 K. Fregene, R. Madhavan, and L. E. Parker, "Incremental multi-agent robotic mapping of outdoor terrain," *Proc. IEEE International Conference on Robotics and Automation*, Washington, DC, 2002, pp. 1339-1346.

4 R. Simmons, *et al.*, "Experience with rover navigation for lunar-like terrains," *Proc. IEEE/RSJ International Conference on Intelligent Robots and System*, Pittsburgh, PA, 1995, pp. 441-446.

5 H. Seraji and A. Howard, "Behavior-based robot navigation on challenging terrain: a fuzzy logic approach," *IEEE Trans. on Robotics and Automation*, vol. 18, no. 3, 2002.

6 I. S. Kweon and Takeo Kanade, "High-resolution terrain map from multiple sensor data," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 278-292, 1992.

7 E. Krotkov and R. Hoffman, "Terrain mapping for a walking planetary rover," *IEEE Trans. on Robotics and Automation*, vol. 10, no. 6, pp. 728-738, 1994.

8 C. M. Shoemaker and J. A. Bornstein, "The demo III UGV program: a testbed for autonomous navigation research," *Proc. IEEE ISIS/CIRA/ISAS Joint Conference*, Gaithersburg, MD, 1998, pp. 644-651.

9 C. Ye and J. Borenstein, "Obstacle avoidance for the segway robotic mobility platform," *Proc. the American Nuclear Society International Conference on Robotics and Remote System for Hazardous Environment*, Gainesville, FL, 2004, pp. 107-114.

10 C. Ye and J. Borenstein, "A novel filter for terrain mapping with laser rangefinders," *IEEE Trans. on Robotics*, vol. 20, no. 5, pp. 913-921, 2004.

11 D. Langer, J. K. Rosenblatt, and M. Hebert, "A Behavior-based system for off-road navigation," *IEEE Trans. on Robotics and Automation*, vol. 10, no. 6, pp. 776-783, 1994

12 D. B. Gennery, "Traversability analysis and path planning for a planetary rover," *Autonomous Robots*, vol. 6, no. 2, pp. 131-146, 1999.

13 S. Singh, *et al.*, "Recent progress in local and global traversability for planetary rovers," *Proc. IEEE International Conference on Robotics and Automation*, San Francisco, CA, 2000, pp. 1194-1200.

14 J. A. Janét, R. C. Luo, and M. G. Kay, "Autonomous Mobile Robot Global Motion Planning and Geometric Beacon Collection Using Traversability Vectors", *IEEE Trans. on Robotics and Automation*, vol. 13, no. 1, pp. 132-140, 1997.

15 N. J. Nilsson, *Principles of Artificial Intelligence,* Tioga Publishing Company, 1980.

16 A. Stentz and M. Hebert, "A complete navigation system for goal acquisition in unknown environments," *Autonomous Robots*, vol. 2, no. 2, 1995.

17 C. Ye and D. W. Wang, "A novel navigation method for autonomous mobile robots," *Journal of Intelligent and Robotic Systems*, vol. 32, no. 4, pp. 361-388, 2001.

18 S. Lacroix, *et al.*, "Autonomous Rover Navigation on Unknown Terrains: Functions and Integration," *International Journal of Robotics Research*, vol. 21, no. 10-11, pp. 917-942, 2002.

19 I. Kammon, E. Rivlin, and E. Rimon, "A new range-sensor based globally convergent navigation algorithm for mobile robots," *Proc. IEEE International Conference on Robotics and Automation*, Minneapolis, MN, 1996, pp. 429-435.

20 S. L. laubach and J. W. Burdick, "An autonomous Sensor-based path-planner for planetary micorovers," *Proc. IEEE International Conference on Robotics and Automation*, Detroit, MI, 1999, pp. 347-354.

21 H. Haddad, M. Khatib, S. Lacroix, and R. Chatila, "Reactive navigation in outdoor environments using potential fields," *Proc. IEEE International Conference on Robotics and Automation*, 1998, pp. 1232-1237.

22 Y. Koren and J. Borenstein, "Potential Field Methods and Their Inherent Limitations for Mobile Robot Navigation," *Proc. IEEE International Conference on Robotics and Automation*, Sacramento, CA, 1991, pp. 1398-1404.

23 J. Borenstein and Y. Koren, "The Vector Field Histogram–Fast Obstacle-Avoidance for Mobile Robots," *IEEE Trans. on Robotics and Automation*, vol. 7, no. 3, pp. 278-288, 1991.

24 L. Ojeda, M. Raju and J. Borenstein, "FLEXnav: A Fuzzy Logic Expert Dead-reckoning System for the Segway RMP." *Proc. of the Defense and Security Symposium (was Aerosense) Unmanned Ground Vehicle Technology VI (OR54)*, Orlando, FL, 2004.

25 D. P. Bertsekas, Constrained Optimization and Lagrange Multiplier Methods, Academic Press, 1982.

26 D. H. Golub and C. F. Van Loan, *Matrix Computations*, Prentice-Hall, 1986.

27 W. H. Press, et al, *Numerical Recipes in C++*, Cambridge University Press, 2002.

Cang Ye received B.E. and M.E. degree from the University of Science and Technology of China, P. R. China in 1988 and 1991, respectively. He received his Ph.D. degree from the University of Hong Kong in 1999.

He is currently an Assistant Professor in Department of Applied Science, University of Arkansas at Little Rock, AR. He was a research investigator at University of Michigan, Ann Arbor, MI from 2003 to 2005. His research interests are in mobile robot terrain mapping and navigation, fuzzy systems and reinforcement learning.

Dr. Ye is a senior member of IEEE and a member of Technical Committee on Robotics and Manufacturing Automation, IEEE Systems, Man, and Cybernetics Society.