Proceedings of the IEEE International Conference on Robotics and Automation, San Francisco, CA, April 2000, pp. 2505-2511.

VFH*: Local Obstacle Avoidance with Look-Ahead Verification

Iwan Ulrich and Johann Borenstein

The University of Michigan
Dept. of Mechanical Engineering and Applied Mechanics
iwan@ri.cmu.edu, johannb@umich.edu

ABSTRACT

This paper presents an enhancement to the earlier developed Vector Field Histogram (VFH) method for mobile robot obstacle avoidance. The enhanced method, called VFH, successfully deals with situations that are problematic for purely local obstacle avoidance algorithms. The VFH* method verifies that a particular candidate direction guides the robot around an obstacle. The verification is performed by using the A* search algorithm and appropriate cost and heuristic functions.*

1. INTRODUCTION

Many mobile robot systems combine a global path-planning module with a local obstacle avoidance module to perform navigation. While the global path planner determines a suitable path based on a map of the environment, the obstacle avoidance algorithm determines a suitable direction of motion based on recent sensor data. Obstacle avoidance is performed locally in order to ensure that real-time constraints are satisfied. A fast update rate of the obstacle avoidance algorithm is required to allow the robot to safely travel at high speeds.

In 1985, Khatib presented the concept of *artificial potential fields*, which was the first real-time obstacle avoidance algorithm for mobile robots as well as for manipulators [6]. Around the same time, Moravec and Elfes pioneered the concept of certainty grids, a widely popular map representation that is well suited for sensor data accumulation and sensor fusion [10]. By integrating the concept of potential fields with the concept of certainty grids, Borenstein and Koren developed the *Virtual Force Field* method [1]. However, based on their experiments, they discovered and analyzed substantial shortcomings that are inherent to the concept of potential fields [7]. In order to overcome these shortcomings, they then developed the *Vector Field Histogram* (VFH), a method that looks for gaps in locally constructed polar histograms [2]. VFH was less likely to get trapped in local minima and allowed robots to travel at faster speeds without becoming unstable [9].

Due to its advantages, VFH became a popular obstacle avoidance method. However, its wide use resulted in the discovery of shortcomings. Several researchers, some of them inspired by their experiments with VFH, then developed algorithms based on the novel concept of the *Steer Angle Field* approach [4,5,11]. We experienced similar problems with the original VFH method when we implemented it in the GuideCane, a special type of mobile robot for the guidance of the blind [3]. Instead of changing the underlying concept of VFH, we overcame its problems with four incremental improvements, which led to VFH+ [12]. First, VFH+ takes into account the width of the mobile robot by using an implicit configuration space approach without explicitly building the C-space [8]. Second, VFH+ takes into account the robot trajectory. This improvement is particularly useful at high speed when the minimum steering radius can no longer be neglected. Third, VFH+ results in a trajectory that is less oscillatory due to a threshold hysteresis. Fourth, VFH+ is able to commit to a direction due to an improved direction selection, which is based on a cost function.

Although we were satisfied with the performance of VFH+ in general, the algorithm sometimes made undesirable choices. Because these situations occurred rarely, the underlying problem was only discovered after extensive testing of the GuideCane. In the next section we will give a detailed description of such a situation that is problematic for a purely local obstacle avoidance algorithm like VFH+. Section 3 outlines the VFH* algorithm, which overcomes this problem with look-ahead verification. Section 4 explains the purpose of the search parameters, while Section 5 describes the VFH* algorithm in detail. Section 6 presents experimental results, and the paper ends with a conclusion in Section 7.

2. PURELY LOCAL OBSTACLE AVOIDANCE

VFH+ sometimes fails to choose the most appropriate direction because of its purely local nature. The inherent problem of purely local obstacle avoidance algorithms can best be explained with an example. Figure 1 shows a situation where a mobile robot travels down a corridor and encounters two obstacles in its path. Obstacles are shown in black, while the configuration space is shown in gray. Although VFH+ uses the concept of configuration space only implicitly, the configuration space is drawn explicitly for better visualization.

The large circle drawn in a dashed line shows the approximate distance at which an obstacle triggers an avoidance maneuver. At the position shown in the example, VFH+ detects two openings. While the opening to the left results in the undesirable trajectory *A*, the opening to the right results in trajectory *B*, which at point *p* eventually turns into the desirable trajectory *C*. Unfortunately, both trajectories *A* and *B* appear equally appropriate to VFH+. In problematic situations like this, VFH+ would thus select the appropriate direction on average only 50% of the time.

It is important to note that we expect the same problem to occur with other purely local obstacle avoidance algorithms, such as those based on the steer angle field approach. The underlying problem is that purely local systems only consider the immediate effects of their selected trajectory without verifying its consequence.

It is also important to note that a larger trigger distance would not eliminate the problem. The same problem would simply occur earlier in the process. Unless the circle was extremely large, the mobile robot would still have two choices when it encounters the obstacle. A large trigger distance is also not practical, because obstacle avoidance maneuvers would start unnecessarily early. Moreover, a large trigger distance might lead the system to no longer detect existing openings and falsely report a trap situation.

The VFH* algorithm described in this paper overcomes problematic situations like this one most of the time by combining VFH+ with the A* search algorithm. VFH* does so by projecting the trajectory of the robot several steps ahead and evaluating the consequences. While VFH* is no longer purely local, it is still a local algorithm and thus performs in real-time.

3. THE VFH* ALGORITHM

The VFH+ method builds a polar histogram around the robot's current position, looks for openings in the histogram, and then determines between one and three suitable directions for each opening. VFH+ also assigns a cost value to each of these *primary candidate directions*. VFH+ then selects the primary candidate direction with the lowest cost as its new direction of motion.

In contrast, VFH* analyzes the consequences of heading towards each primary candidate direction before making a final choice for the new direction of motion. For each primary candidate direction, VFH* computes the new position and orientation that the robot would have after moving for a projected step distance d_s. At every projected position, VFH+ is again used to construct a new polar histogram based on the map information. This histogram is then analyzed for candidate directions, called

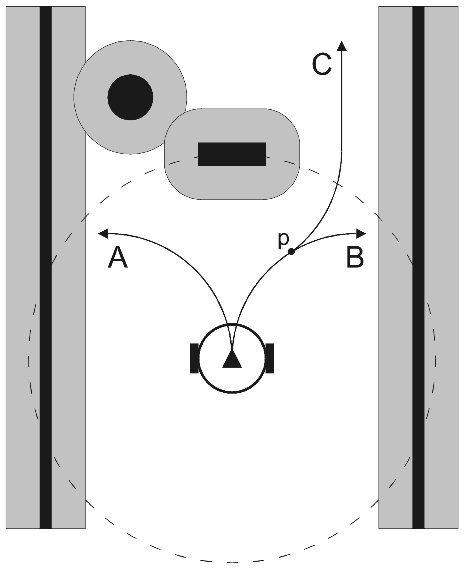

Figure 1: Problematic situation for purely local obstacle avoidance algorithms

projected candidate directions. By repeating this process n_g times, we build a search tree of depth n_g, where the end nodes (goals) correspond to a total projected distance of $d_t = n_g \cdot d_s$.

The goal of this search process is to find a suitable projected trajectory of distance d_t. Nodes in the search tree represent the projected positions and orientations of the mobile robot. Arcs represent the candidate directions leading from one position to another.

For every candidate direction c, a cost $g(c)$ is calculated similar to the cost function used in VFH+. The cost associated with a node is simply the sum of the costs of the branches leading back to the start node. The primary candidate direction that leads to the end node with the smallest total cost is then selected as the new direction of heading φ_d.

For an easier understanding of the concept, we have described VFH* as if it was based on the *breadth-first search* (BFS) algorithm. Actually, VFH* employs the *A** search method and uses a heuristic function $h(c)$ that is similar to the cost function of VFH+. The priority value for each node is then defined by $f(c) = g(c) + h(c)$. Because it takes much less time to compute the heuristic function than to expand a node, A* is much faster than BFS for our application. Like BFS, A* is optimal and complete, because our heuristic function never overestimates the cost to reach the goal state.

4. SEARCH PARAMETERS

The key parameters of VFH* are the total projected distance d_t, the projected step distance d_s, and the goal depth n_g. These parameters are related to each other through: $d_t = n_g \cdot d_s$.

The goal depth is proportional to the total projected distance d_t. The higher d_t is selected, the larger the total look-ahead, and the better the results of VFH* are. However, if this parameter is selected too high, the obstacle avoidance algorithm is slowed down substantially. It is not recommended to choose a value for the total projected distance that exceeds the range of the robot's sensors, unless the robot has an accurate map of a mainly static environment. The selection of d_t is thus a trade-off between the speed and the quality of the algorithm. If possible, this parameter should be set close to the range of the robot's sensors.

The goal depth is inversely proportional to the projected step distance d_s. If d_s is selected too large, the new position might be incorrectly projected through or right into an obstacle. If d_s is selected too small, the effect of a single projection is too small, resulting in a high value for n_g. This would result in an unnecessary deep search tree, which would substantially slow down the obstacle avoidance algorithm. The selection of d_s is thus a trade-off between the speed and the validity of the algorithm. Based on our experiments, we recommend setting this parameter equal to the diameter of the robot.

It is important to note that the VFH+ method is a special case of the VFH* method, where n_g is set equal to one, such that d_t is equal to d_s.

5. THE EXPANSION STEP

The expansion of a node n_i consists of building the polar histogram at the node's projected position (x_i,y_i), determining the corresponding candidate directions, calculating the projected position and orientation of the following nodes, determining the cost of reaching these nodes, and determining their heuristic values.

The first two steps, building the polar histogram and determining the corresponding candidate directions, are performed in the same way as in the VFH+ algorithm. The three remaining steps are described in detail in the following sections.

5.1 Projection of Position and Orientation

The computation of the projected position (x_{i+1},y_{i+1}) and orientation θ_{i+1} for a candidate direction can be done in several ways. In our approach, the projected robot trajectory is approximated by arcs of a circle and straight lines. This is the same model that we used in VFH+ to take into account the robot trajectory. This approximation is a suitable trade-off between algorithm accuracy and speed. Example trajectories are shown in Figure 2:

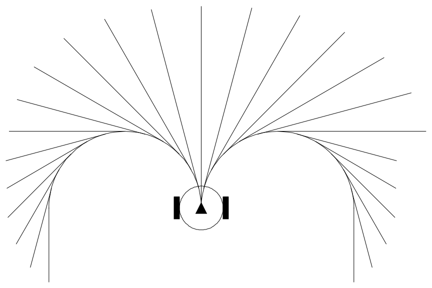

Figure 2: Trajectory approximation.

The parameters of this trajectory model are the projected step distance d_s and the minimum steering radii r_r and r_l for right and left turns respectively. For a given candidate direction, the algorithm first determines if the robot can reach this orientation during the projected step distance. If not, the projected trajectory is simply approximated by a curve with constant curvature. It is also necessary to distinguish between candidate directions to the right and to the left of the robot. Thus, we end up with four sets of equations. The maximum directions to the right and to the left are defined by:

$$\theta_r = \theta_i - \frac{d_s}{r_r} \qquad \theta_l = \theta_i + \frac{d_s}{r_l} \tag{1}$$

5.2 The Cost Function

The cost function for a *primary* candidate direction c_0 leading from the root node at depth zero to a successor node is identical to the one used in the VFH+ method:

$$g_0(c_0) = \mu_1 \cdot \Delta(c_0, k_t) + \mu_2 \cdot \Delta\left(c_0, \frac{\theta_n}{\alpha}\right) + \mu_3 \cdot \Delta(c_0, k_{d,n-1}) \tag{2}$$

with:

$$\Delta(c_1,c_2) = \min\left\{|c_1 - c_2|, \left|c_1 - c_2 - \frac{360^0}{\alpha}\right|, \left|c_1 - c_2 + \frac{360^0}{\alpha}\right|\right\} \tag{3}$$

where:

α:	angular resolution of histogram
θ_n:	current orientation
k_t:	target direction divided by α
$k_{d,n-1}$:	previously selected direction of motion / α

The three terms of the cost function retain the same purpose as in the VFH+ method. While the first term is responsible for the goal-oriented behavior, the two other terms make the robot commit to a direction. For a goal-oriented robot, we still have the following condition:

$$\mu_1 > \mu_2 + \mu_3 \qquad \text{[condition 1]}$$

For a *projected* candidate direction c_i of a node at depth i larger than zero, we propose a slightly modified cost function as follows:

$$g_i(c_i) = \lambda^i \cdot \Big[\mu_1' \cdot \max\{\Delta(c_i, k_t), \Delta(k_e, k_t)\} + \mu_2' \cdot \Delta\left(c_i, \frac{\theta_i}{\alpha}\right) + \mu_3' \cdot \Delta(c_i, c_{i-1})\Big] \qquad (4)$$

with: $$k_e = -\arctan\left(\frac{y_{i+1} - y_i}{x_{i+1} - x_i}\right) \qquad (5)$$

and: $$0 < \lambda \le 1 \qquad (6)$$

The first term of (4) again represents the cost associated with the deviation from the target direction, resulting in the goal-oriented behavior. However, this term is slightly different for a projected candidate direction than it is for a primary candidate direction. In the case of a projected candidate direction, this term also considers the effective direction of motion k_e, or in other words, the forward progress of a trajectory. There is an important difference between a candidate direction c_i and the corresponding effective direction of motion k_e. Ideally, we want them both to be in the same direction as the target direction. However, depending on the robot's current orientation and situation, it is possible for one of them to be equal to the target direction while the other one largely deviates from it. If we did not consider the effective direction of motion, a part of the projected trajectory could have very low cost even though it does not make any forward progress at all. Figure 3 shows an example, where the cost associated with the first term would be zero. By including the effective direction of motion in the first term, the cost of this trajectory becomes high, as it provides no forward progress at all.

It is important to note that for a primary candidate direction the first term of the cost function does not include the effective direction of motion. As the robot has no control over the current orientation, there is no reason to associate a cost with the effective direction of motion of a primary candidate direction. In contrast, the robot has some control over its projected trajectory. Therefore, it makes sense to associate a cost with the effective direction of motion of a projected candidate direction.

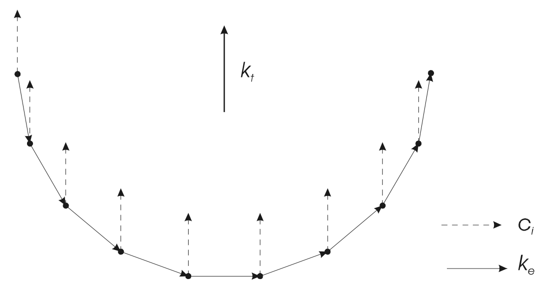

Figure 3: Trajectory part with zero "target direction" cost but high "effective direction" cost.

The second and third term have a different meaning for a projected candidate direction than for a primary candidate direction. In the case of a primary candidate direction, these terms represent a short-term memory effect that makes the robot commit to a direction. In the case of a projected candidate direction, the second and third term represent a cost associated with the smoothness of a projected trajectory.

The higher μ_1' is, the more goal oriented the robot's behavior is. The higher μ_2' and μ_3' are, the more the robot tries to find a smooth path. Only the relative values of these parameters are important, not their absolute values. For a goal-oriented robot, the following condition must be satisfied:

$$\mu_1' > \mu_2' + \mu_3' \qquad \text{[condition 2]}$$

To emphasize the importance of a primary candidate direction over a projected candidate direction, the following condition must also be satisfied:

$$\mu_1 \ge \mu_1' \qquad \text{[condition 3]}$$

Experiments with the GuideCane and simulations have shown that a good set of parameters for a goal-oriented mobile robot is:

$$\begin{array}{lll} \mu_1 = 5 & \mu_2 = 2 & \mu_3 = 2 \\ \mu_1' = 5 & \mu_2' = 1 & \mu_3' = 1 \end{array} \qquad (7)$$

Another important parameter is the *discount factor* λ, which we set to 0.8 in our experiments. Instead of giving equal weight to all candidate directions independent of their depth i, they are weighted by a factor λ^i. There are three reasons for the introduction of this factor.

First, it decreases the problem of a fixed goal depth n_g with a sharp cut-off. Without λ, all branches would have the same weight and the obstacle avoidance algorithm would not always behave as desired. An example of such a case with n_g set to 7 is shown in Figure 4, where the goal direction is indicated by k_t. Without λ, the total cost of trajectory *B* would be cheaper than the one of trajectory *A*. As a result, the robot keeps moving towards the right without making any forward progress towards the goal direction. Without λ, the obstacle avoidance algorithm has the undesired tendency to find trajectories that stop shortly before being influenced by an obstacle. If n_g was increased by one, trajectory *B* would become much more costly. However, even if trajectory *B* became more costly than trajectory *A*, the obstacle avoidance algorithm would pick trajectory *C* as its cheapest trajectory, again trying to stop shortly before being influenced by an obstacle. By introducing λ, the cut-off at the last branch is less sharp, as the weight of the last branch becomes relatively small. Consequently, trajectory *A* of our example becomes cheaper than trajectory *B*.

Second, the discount factor λ compensates somewhat for the uncertainty of the map information. Due to its sensors and the stochastic map building process, the mobile robot is more certain about its immediate surroundings. The further the projected position is away from the current position, the more uncertain the content of the map is at that position.

Finally, not using a discount factor is identical to using a discount factor with a value equal to one. With λ, we have more control over the weights of the branches at different depths, and we gain more control over the search algorithm's behavior.

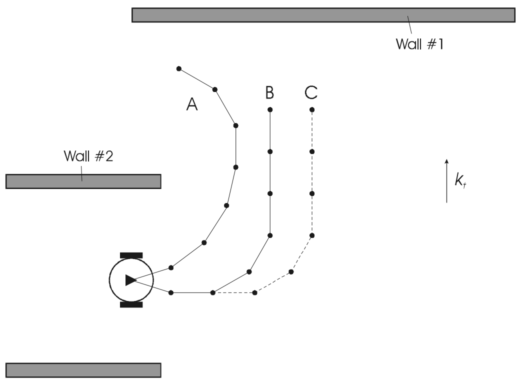

Figure 4: Situation that requires the discount factor λ.

5.3 The Heuristic Function

The heuristic function $h(c)$ is the estimated cost of the cheapest path from a node n_i to the goal state. A function is an admissible heuristic if it never overestimates the cost to reach the goal. With condition 2 satisfied, the cost is minimal if the robot can head towards the target direction k_t at every following node. We can thus get a simple admissible heuristic by replacing c_i in the cost function with k_t:

$$h_i(n_i) = \lambda^i \cdot \left[\mu_2' \cdot \Delta\left(k_t, \frac{\theta_i}{\alpha}\right) + \mu_3' \cdot \Delta(k_t, c_{i-1}) \right] \tag{8}$$

This heuristic only considers the cost associated with the next branch. It does not consider the cost associated with the effective direction of motion. Therefore, this heuristic is not optimal as it underestimates the minimum cost to reach a goal node. However, this heuristic is admissible and computationally very efficient.

A better admissible heuristic that considers the cost associated with the effective direction of motion is:

$$h_i(n_i) = \lambda^i \cdot \left[\mu_1' \cdot \Delta(k_e, k_t) + \mu_2' \cdot \Delta\left(k_t, \frac{\theta_i}{\alpha}\right) + \mu_3' \cdot \Delta(k_t, c_{i-1}) \right] \tag{9}$$

$$\text{with:} \quad k_e = -\arctan\left(\frac{y_{i+1} - y_i}{x_{i+1} - x_i}\right) \text{ based on } c_i = k_t \tag{10}$$

This heuristic is better but computationally more expensive than the previous one. It is still not optimal because it only considers the minimum cost associated with the next branch. To compute an optimal heuristic value, one could simply *expand* (without building the polar histogram and determining the corresponding candidate directions) the current node until reaching the goal depth by using the target direction k_t as the candidate direction at each node. By summing up the corresponding costs, we could get the optimal heuristic value. Yet, this heuristic requires more computational power.

The difference between these heuristics is a trade-off between quality and speed of the heuristic function. However, the heuristic has no influence on the resulting direction of heading. The first heuristic is currently implemented in the GuideCane.

5.4 The Branching Factor

By reducing the branching factor *b* of the search tree, the search algorithm becomes faster and requires less memory. The branching factor can be reduced by eliminating redundant nodes.

Each node has a number of successor nodes equal to the number of its candidate directions. Because the projected step distance d_s is usually small, several projected positions and orientations of successor nodes can be identical. All candidate directions that exceed θ_r have the same projected position and orientation. Similarly, all candidate directions that exceed θ_l have common values for the projected position and orientation. To reduce the branching factor, all but the cheapest candidate direction for each side can be eliminated. The explanation for the validity of this approach is simple. Consider a node with several candidate directions that exceed θ_l. As their projected positions and orientations are identical, their polar histograms will also be identical. Therefore, the remaining search trees will be identical for all these candidate directions. Only the third term of the cost function associated with these branches would be different. As long as condition 2 is satisfied, the costs associated with the nodes of the search tree that starts at the candidate direction with the lowest cost are always lower than the corresponding nodes of the other candidate directions. Due to this node elimination method, the branching factor b is rarely larger than three.

Another speed improvement was achieved by expanding the search tree only if there is more than one primary candidate direction. With only one primary candidate direction, there is no need to expand the search tree, as the robot has only one choice anyway.

6. EXPERIMENTAL RESULTS

The advantage of VFH* over VFH+ depends mainly on the goal depth n_g, which is proportional to the distance of look-ahead verification. Although we have tested VFH* with the GuideCane, we prefer to show results from a simulated obstacle course for a better comparison. In tests with the GuideCane, the VFH* method performed equally well as long as the obstacles were detected by the robot's ultrasonic sensors. Figure 5 shows the trajectories of VFH* with four different goal depth values. Figure 6 shows the search trees at critical or interesting positions. In all our experiments, we used the first heuristic.

Figure 5a shows the trajectory of VFH+. As explained previously, VFH* with a goal depth of one is identical to VFH+. Shortly after avoiding the first two obstacles, VFH+ encounters a problematic situation that is similar to the one described in Section 2. When the horizontal wall triggers an obstacle avoidance maneuver, VFH+ makes an unfortunate choice and heads to the left. At the position shown in Figure 5a, VFH+ slows down or stops the robot, as its minimum turning radius is too large for avoiding the corner on either side. In the case of a holonomic or a differential-drive configuration, the robot needs to substantially slow down, thus reducing its minimum turning radius. In the case of a car-like configuration with a non-zero minimum turning radius, the robot needs to stop, back up, and then proceed to the right. With either configuration, it would have been better to turn to the right earlier.

Figure 5b shows the trajectory of VFH* with a goal depth of two. Due to the look-ahead verification, VFH* makes the correct decision and turns to the right. Figure 6a shows the search tree at the time when this critical decision happens. After driving around the horizontal wall, VFH* guides the robot into a shallow dead-end. At the position shown in Figure 5b, VFH* detects that the robot entered a dead-end. Thus, the robot needs to back up or turn if its minimum turning radius permits it.

Figure 5c shows the trajectory with a goal depth of five. In this case, the goal depth is large enough to even avoid the dead-end. Figure 6b shows the search tree at the position when VFH* detects the dead-end and decides to avoid it by turning left. The depth of the dead-ends that can be avoided with VFH* is proportional to the value of the goal depth. However, it is clear that dead-ends can only be avoided if the range of the robot's sensors is farther than the depth of the dead-end, or if the dead-end is represented in a given map.

Figure 5d shows the trajectory with a goal depth of ten. In this case, VFH* already detects at the very beginning that it is better to turn right than left. In addition, the dead-end avoidance maneuver is initiated much earlier. The search trees at the two critical positions are illustrated in Figure 6c.

Figure 5 shows that the higher n_g is selected, the better VFH* performs. However, this improvement is at the expense of computational time. Table 1 shows an execution time comparison based on the GuideCane's embedded computer, a PC 486 running at 66 MHz. The second

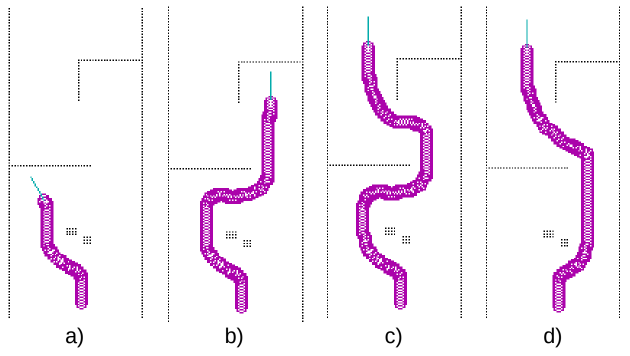

Figure 5: VFH* trajectory with: a) n_g = 1, b) n_g = 2, c) n_g = 5, and d) n_g = 10.

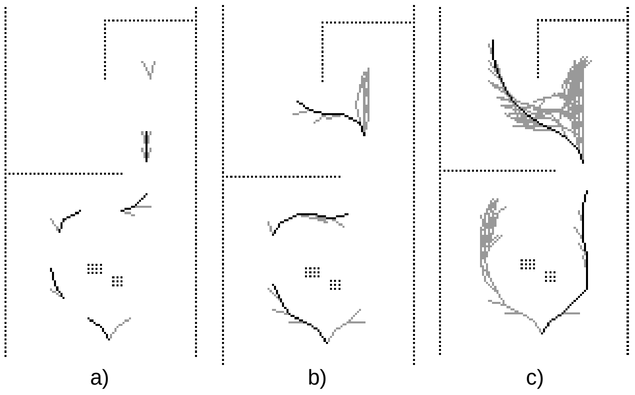

Figure 6: VFH* search trees with: a) n_g = 2, b) n_g = 5, c) n_g = 10. The black curve indicates the projected trajectory of the selected primary candidate direction. The gray lines show the expanded tree branches of the trajectories with higher total costs than the trajectory in black.

column averages the computation time over a densely cluttered obstacle course, excluding the times when there was only one primary candidate direction. The third column shows the maximum observed required computation time, ranging from 6 to 242 ms. In summary, the VFH* method is fast as long as the goal depth n_g is kept small. The average execution time is very short even with a goal depth as high as ten. However, the critical value is the maximum execution time, which limits the goal depth for an application. The maximum required time could be decreased by applying additional branching factor reduction techniques and by optimizing the software.

n_g	$T_{average}$	$T_{maximum}$
1	3 ms	6 ms
2	5 ms	11 ms
3	8 ms	22 ms
4	10 ms	39 ms
5	12 ms	82 ms
10	30 ms	242 ms

Table 1: VFH* execution time.

Tests with both the simulated and the real GuideCane have shown that a goal depth of just two is often sufficient to deal with problematic situations. As the GuideCane's on-board computer is clocked at only 66 MHz, we selected a value of two for the goal depth in the current implementation. With the range of the GuideCane's sonars, a goal depth between 3 and 5 would be optimal. However, the corresponding worst-case execution time is too slow with the current on-board computer to allow safe travel at high speed.

Similar to VFH+, the VFH* method is not very sensitive to its parameter values, and only little time is usually required for parameter tuning. As long as conditions 1 to 3 are satisfied and the parameter values are selected reasonably, the VFH* method performs well.

7. CONCLUSION

This paper presented VFH*, a local obstacle avoidance algorithm that uses look-ahead verification to consider more than the robot's immediate surroundings. While VFH* has the same obstacle avoidance performance as VFH+ for regular obstacles, VFH* is capable of dealing with problematic situations that would require the robot to substantially slow down or even stop. Experimental tests with the real GuideCane and simulations have proved to be very successful.

ACKNOWLEDGMENT

This research was funded by the Whitaker Foundation.

REFERENCES

[1] Borenstein, J., and Koren, Y., "Real-time Obstacle Avoidance for Fast Mobile Robots", *IEEE Transactions on Systems, Man, and Cybernetics,* Vol. 19, No. 5, Sept./Oct. 1989, pp. 1179-1187.

[2] Borenstein, J., and Koren, Y., "The Vector Field Histogram - Fast Obstacle Avoidance for Mobile Robots", *IEEE Journal of Robotics and Automation,* Vol. 7, No. 3, June1991, pp. 278-288.

[3] Borenstein, J., and Ulrich, I., "The GuideCane - A Computerized Travel Aid for the Active Guidance of Blind Pedestrians", *IEEE Int. Conf. on Robotics and Automation*, April 1997, pp. 1283-1288.

[4] Feiten, W., Bauer, R., and Lawitzky, G., "Robust Obstacle Avoidance in Unknown and Cramped Environments", *IEEE Int. Conf. on Robotics and Automation*, May 1994, pp. 2412-2417.

[5] Fox, D., Burgard, W., and Thrun, S., "The dynamic window approach to collision avoidance", *IEEE Robotics and Automation Magazine,* Vol. 4, No. 1, March 1997, pp. 23-33.

[6] Khatib, O., "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots", *IEEE Int. Conf. on Robotics and Automation,* March 1985, pp. 500-505.

[7] Koren, Y., and Borenstein, J., "Potential Field Methods and Their Inherent Limitations for Mobile Robot Navigation", *IEEE Int. Conf. on Robotics and Automation*, April 1991, pp. 1398-1404.

[8] Lozano-Pérez, T., "Spatial Planning: A Configuration Space Approach", *IEEE Transactions on Computers*, Vol. C-32, No. 2, 1983, pp. 108-120.

[9] Manz, A., Liscano, R., and Green, D.A., "A Comparison of Realtime Obstacle Avoidance Methods for Mobile Robots", *Experimental Robotics*, Toulouse, France, June 1991.

[10] Moravec, H.P., and Elfes, A., "High resolution maps from wide angle sonar", *IEEE Int. Conf. on Robotics and Automation*, March 1985, pp. 116-121.

[11] Simmons, R., "The Curvature-Velocity Method for Local Obstacle Avoidance", *IEEE Int. Conf. on Robotics and Automation*, April 1996, pp. 3375-3382.

[12] Ulrich, I., and Borenstein, J., "VFH+: Reliable Obstacle Avoidance for Fast Mobile Robots", *IEEE Int. Conf. on Robotics and Automation*, May 1998, pp. 1572-1577.