Proceedings of the 1996 IEEE
International Conference on Robotics and Automation
Minneapolis, Minnesota - April 1996

The Curvature-Velocity Method for Local Obstacle Avoidance

Reid Simmons

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

We present a new method for local obstacle avoidance by indoor mobile robots that formulates the problem as one of constrained optimization in velocity space. Constraints that stem from physical limitations (velocities and accelerations) and the environment (the configuration of obstacles) are placed on the translational and rotational velocities of the robot. The robot chooses velocity commands that satisfy all the constraints and maximize an objective function that trades off speed, safety and goal-directedness. An efficient, real-time implementation of the method has been extensively tested, demonstrating reliable, smooth and speedy navigation in office environments. The obstacle avoidance method is used as the basis of more sophisticated navigation behaviors, ranging from simple wandering to map-based navigation.

1 Introduction

We address the problem of local obstacle avoidance for mobile robots operating in unknown, or partially known, environments. While this problem has been studied by several other researchers, we have a number of desiderata that necessitate a new approach to the problem. Several desiderata are common to most existing methods:

- The robot should navigate safely, even in the face of noisy sensors and dead-reckoning error.
- The robot should be goal-directed while trying to avoid obstacles.
- The method must be computationally efficient, to run in real-time on-board the robot.

In addition, we have a number of desiderata that are often not addressed by other methods:

- The dynamics of the robot should be taken into account, to enable it to travel at high speeds in crowded environments.
- The method should explicitly try to maximize forward progress of the robot.
- The method should simultaneously control both the direction and speed of the robot.

Our approach, the *Curvature-Velocity Method (CVM)*, addresses all these concerns. The main distinctions of the method is that it operates in the velocity space of the robot, rather than Cartesian or configuration space, and it chooses commands by maximizing an objective function that trades off vehicle safety, speed, and goal-directedness. The method presumes that the robot can control both translational and rotational velocities, but cannot turn instantaneously. Rather it travels along arcs of circles. This formulation includes synchro-drive robots, differentially steered robots, and many non-holonomic vehicles. While our formulation neglects the effects of accelerations, in practice it is a good approximation for indoor mobile robots traveling at walking speeds.

The curvature-velocity method works by adding constraints to the velocity space and choosing the point in that space that satisfies all constraints and maximizes an objective function. Constraints derive from physical limitations on the robot's velocities and accelerations, and from sensor data that indicate the presence of obstacles. The latter are used to represent, for each possible curvature, how far the robot can travel before hitting an obstacle.

To achieve real-time performance, the curvature distance to obstacles is approximated with a piecewise constant function. This approximation divides the velocity space into a discrete number of regions, each of which has constant distance to impact. The method finds the point in each region that maximizes the objective function. The overall maximal point is used to command the robot. Several simple extensions make the basic method more robust to sensor noise and reduce the possibility of the robot getting stuck. Tests on our indoor mobile robot demonstrate that it produces speedy, smooth and safe travel in peopled office environments.

2 Related Work

Several well-known local obstacle avoidance methods work by computing a direction for the robot to head in, but do not take vehicle dynamics into account. For example, Potential Field approaches [1, 8] use vector sums of repulsive and attractive features to compute a desired robot heading. Speed control is sometimes handled by choosing velocity proportional to the magnitude of the potential vector. The Vector Field Histogram method [2] improves on this approach by computing a one-dimensional polar histogram, which is then processed to detect open areas for the robot to travel through. Robot velocity, chosen after the direction has been selected, is proportional to the distance to obstacles ahead. While this method produces smoother travel and can handle both narrow and wide openings, it,

0-7803-2988-4/96 $4.00 © 1996 IEEE

like the Potential Field approach, does not account for the fact that when robots turn they typically move along arcs, rather than in straight lines. In cluttered environments, this neglect of vehicle dynamics can be critical.

While methods that take vehicle dynamics and non-holonomic constraints into account have been studied in the context of off-line path planning [4, 9], such methods are generally too computationally expensive for fast local obstacle avoidance.

Recently, however, several local obstacle avoidance methods have been reported that do incorporate vehicle dynamics, choosing steering commands rather than travel direction. The Steering Angle Field method [5], like our own, uses the curvatures tangent to obstacles to constrain a continuous space (in their case, the one-dimensional space of steering angles). The curvatures and associated arc distances are used to prohibit travel over ranges of steering angles. The method calculates constraints for several distance thresholds, and tries to travel along the freest dimension. Speed control is an iterative "negotiation" process between the pilot module and the local obstacle avoidance module, as opposed to our method, in which speed and turn rate are chosen simultaneously, as the point in velocity space that maximizes the objective function.

A similar method for high-speed indoor navigation that operates in velocity space was developed somewhat earlier, but independently [3]. This method looks at a discrete set of arcs, constrained by the vehicle dynamics, and chooses one that most closely heads in the goal direction, while ensuring that the robot does not hit an obstacle within a few seconds of travel. The original method used a two-step approach to pick curvatures and velocities; subsequently, they have adopted our one-step method for simultaneously choosing curvatures and velocities [6]. A similar approach has been developed for outdoor navigation [7]. Here, full vehicle dynamics are considered, so the path is not necessarily a circular arc, a traversability measure is calculated for each path, and the one with the best value is chosen. Both these methods have the problem that, in analyzing only a discrete set of arcs, good paths may "fall through the cracks" and not be considered.

3 The Curvature-Velocity Method

We formulate the local obstacle avoidance problem as one of *constrained optimization* in the *velocity space* of the robot. The *velocity space* of a robot is the set of controllable velocities. For synchro-drive robots, such as our own, the velocity space has the orthogonal dimensions of translational (tv) and rotational (rv) velocities.[1] By *constrained optimization,* we mean that the robot chooses the (tv, rv) pair that maximizes some objective function, while meeting all constraints on allowable velocities.

There are several advantages to this formulation of the local obstacle avoidance problem. First, by operating in velocity space we can simultaneously control the speed and heading of the robot, and can come up with solutions that correspond directly to commands to control the robot. By treating the problem as one of constrained optimization, we can easily incorporate constraints from the environment and robot dynamics, and can come up with formulations that, for instance, trade off speed for safety.

To begin with, we assume that the robot always travels along arcs of a circle, whose curvature is given by $c = rv/tv$, where a positive curvature denotes clockwise motion. Thus, each point in velocity space corresponds to motion of constant curvature in Cartesian space. While this is actually just an approximation, due to effects of acceleration, such effects are negligible given the relatively slow speeds and high accelerations of most indoor mobile robots [6].

The physical limitations of the robot impose two types of velocity-space constraints. One is that the robot has maximum rotational and translational velocities: $tv \le tv_{max}$, $tv \ge -tv_{max}$, $rv \le rv_{max}$, $rv \ge -rv_{max}$. In our formulation, we also add the constraint $tv \ge 0$ to prohibit backwards motion. Limitations on rotational and translational accelerations, ra_{max} and ta_{max}, supply other constraints. Given the robot's current velocities, rv_{cur} and tv_{cur}, and some time interval T_{accel} (which is chosen based on the cycle time of the CVM algorithm), we add three more constraints that give the achievable velocities in the next time step:

$$rv \ge rv_{cur} - (ra_{max} \times T_{accel})$$

$$rv \le rv_{cur} + (ra_{max} \times T_{accel})$$

$$tv \le tv_{cur} + (ta_{max} \times T_{accel})$$

The obvious other constraint on tv is not added for safety reasons: We want to ensure that $tv = 0$ is always a reachable part of the space.

An important source of constraints is imposed by the obstacles in the environment. We can transform Cartesian space obstacles into velocity space constraints as follows: First, convert the obstacle to configuration space and, for all curvatures c, calculate the distance $d_c(c, obs)$ that the point robot would travel before hitting the obstacle *obs*. Then, define the distance function for an obstacle in velocity space as:

$$d_v(tv, rv, obs) = \begin{cases} d_c(rv/tv, obs) & \text{if } (tv \neq 0) \\ \infty & \text{otherwise} \end{cases} \quad (1)$$

Given a set of obstacles *OBS*, the cumulative distance function is defined as:

1. For differentially steered vehicles, the velocity space of left (v_l) and right (v_r) velocities is easily transformed into the tv/rv space: $rv = (v_r - v_l)/w$; $tv = ((v_l - v_r)/(v_r - v_l)) \cdot w/2$, where w is the robot's lateral wheelbase.

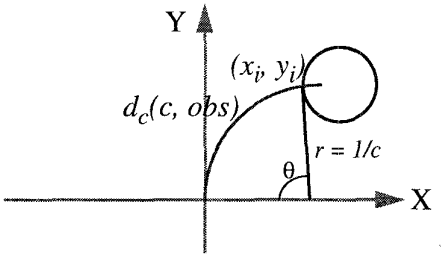


Figure 1: Obstacle Distance Calculations

$$D(tv, rv, OBS) = \min_{obs \in OBS} d_v(tv, rv, obs) \quad (2)$$

Finally, since sensor range is limited (and to avoid computing with infinite values), we clip the function D at some limiting distance L (three meters, in our implementation):

$$D_{limit}(tv, rv, OBS) = \min(L, D(tv, rv, OBS)) \quad (3)$$

In general, computing the obstacle distance function $d_c(c, obs)$ can be very complex for arbitrarily shaped obstacles. To address this, we approximate obstacles as circles. This is a reasonable approach given our sensor input: sonar and laser range readings. Since our robot is also circular, converting from Cartesian to configuration space obstacles merely involves increasing the radii of the obstacles by the radius of the robot. Now, computing d_c is straightforward. For a robot at the origin facing the positive y-axis, given that the curvature c intersects the obstacle at (x_i, y_i), we get (Figure 1):

$$\theta = \begin{cases} \text{atan}\,(y_i / (x_i - 1/c)) & c < 0 \\ \pi - \text{atan}\,(y_i / (x_i - 1/c)) & c > 0 \end{cases}$$

$$d_c(c, obs) = \begin{cases} y_i & c = 0 \\ |1/c| \cdot \theta & c \neq 0 \end{cases} \quad (4)$$

Given these physical and environmental constraints on the velocity space, commands for the robot are chosen by optimizing an objective function. From the desiderata in Section 1, it is clear that the objective function should prefer higher speeds, curvatures that travel longer before hitting obstacles, and should try to orient the robot to head in the desired goal direction. We represent these criteria with a simple linear objective function, in which each term is normalized between zero and one:

$$f(tv, rv) = \alpha_1 speed(tv) + \alpha_2 dist(tv, rv) + \alpha_3 head(rv)$$

$$speed(tv) = tv / tv_{max}$$

$$dist(tv, rv) = D_{limit}(tv, rv, OBS)/L \quad (5)$$

$$head(rv) = 1 - |\theta_g - rv \cdot T_c| / \pi$$

The *speed* term simply indicates a preference for traveling faster, all else being equal. The *dist* term indicates a preference for traveling longer distances along the given curvature (*rv/tv*) without hitting obstacles. The *head* term is the error in goal heading. It is defined to be the difference between the desired goal heading θ_g (in the robot's local reference frame) and the heading the robot will achieve if it turns at *rv* radians per second for some time constant T_c (our implementation sets T_c to one second).

The α values indicate the relative weight to be given to each term in the objective function. With this objective function the robot can exhibit various behaviors, depending on the α weights and the distribution of obstacles, ranging from slowing down to turn sharply and avoid an obstacle, to traveling at full speed, but turning earlier to avoid the same obstacle. Section 6 presents our experimental results on the sensitivity of the method to the choice of α values.

4 Real-Time Implementation

The curvature-velocity method, as presented in the previous section, meets all of our criteria for local obstacle avoidance, except one: it is not computationally efficient. It is hard to compute D_{limit}, even with the simplifying assumption of circular obstacles. Also, given a general formulation of D_{limit}, it is time consuming to optimize f, even with approximation techniques such as simulated annealing. In this section, we describe implementation details that address the computational concerns.

Real-time performance is achieved by approximating D_{limit} with a finite set of intervals, each of which has constant "distance to impact." This set of such *curvature intervals* is determined by using the curvatures tangent to obstacles to divide the velocity space into regions of constant distance, and by further dividing overlapping regions such that the distance associated with each interval is the minimum distance of all overlapping regions. Minimum and maximum velocity and acceleration constraints are added to the space and, for each curvature interval, the vertex points along the upper boundary of the constraints are evaluated (since the point that maximizes the objective function is guaranteed to lie along the upper boundary). The robot chooses the command that maximizes the objective function over all curvature intervals.

We compute D_{limit} (Eqn. 3) using an approximation of the velocity-distance function d_v (Eqn. 1). Note that, for a given obstacle *obs*, $d_c(c, obs)$ is infinite outside the curvatures tangent to the obstacle. Thus, we need consider only those curvatures between c_{min} and c_{max} (Figure 2):

$$c_{min} = 2(x_{obs} - y_{obs}) / \sqrt{x_{obs}^2 + y_{obs}^2 + r_{obs}^2}$$

$$c_{max} = 2(x_{obs} + y_{obs}) / \sqrt{x_{obs}^2 + y_{obs}^2 + r_{obs}^2} \quad (6)$$

The intersection points (needed to compute d_c, Eqn. 4) are:

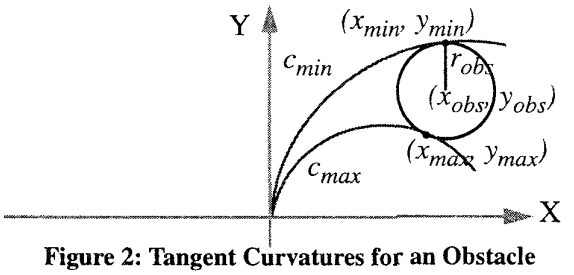


Figure 2: Tangent Curvatures for an Obstacle

$$
\begin{aligned}
x_{min} &= (x_{obs} - r_{obs}) / (1 - (c_{min} \cdot r_{obs})) \\
y_{min} &= y_{obs} / (1 - (c_{min} \cdot r_{obs})) \\
x_{max} &= (x_{obs} + y_{obs}) / (1 + (c_{max} \cdot r_{obs})) \\
y_{max} &= y_{obs} / (1 + (c_{max} \cdot r_{obs}))
\end{aligned} \tag{7}
$$

Note that Eqn. 6 is not defined if the obstacle circle overlaps the origin ($x_{obs}^2 + y_{obs}^2 \le r_{obs}^2$). Since, in reality, the obstacle and the robot cannot occupy the same space, this can only occur due to sensor noise or the circular approximations made in specifying obstacles. Either way, we deal with such an obstacle by defining its radius r_{obs} to be $x_{obs}^2 + y_{obs}^2 - \varepsilon$ (where ε is set to a centimeter). We can then use Eqn. 6 to calculate minimum and maximum curvatures for *all* obstacles.

Given the tangent curvatures, a first approximation (this will be refined subsequently) is to take d_v to be constant between the minimum and maximum curvatures:

$$
d_v(tv, rv, obs) = \begin{cases} \min(d_c(c_{min}, obs), d_c(c_{max}, obs)) \\ \quad \text{if} c_{min} \le rv/tv \le c_{max} \\ \infty \quad \text{otherwise} \end{cases} \tag{8}
$$

This produces a set of obstacle intervals, each of constant distance. To compute the piecewise constant approximation of D_{limit}, we find the min-union of these obstacle intervals by splitting overlapping obstacle intervals and eliminating the overlapping piece with the greater associated distance. An efficient algorithm for this uses the *curvature interval* data structure $(\langle c_1, c_2 \rangle, d_{1,2})$, where $c1$ and $c2$ are two curvatures ($c_1 \le c_2$), and $d_{1,2}$ is the constant distance value within that interval. Geometrically, each curvature interval defines a pair of lines in velocity space, with the distance value between the lines being constant (Figure 3).

The min-union algorithm begins with the curvature interval $(\langle -\infty, \infty \rangle, L)$. For each obstacle, the pair of tangent curvatures (Eqn. 6) and the associated distance (Eqn. 8) are computed to form a new curvature interval

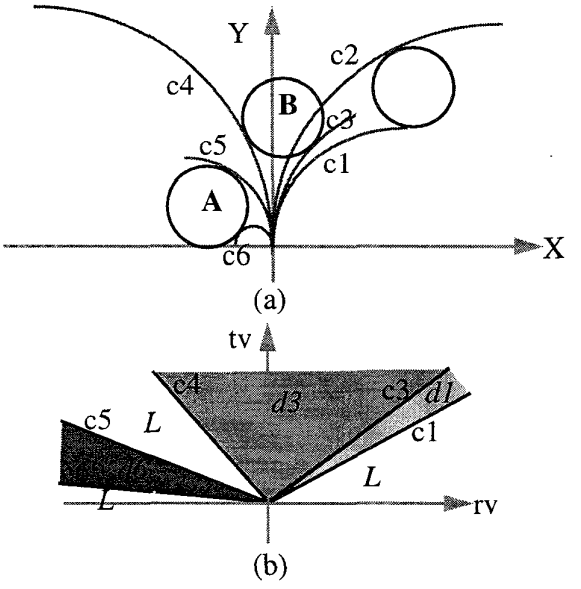


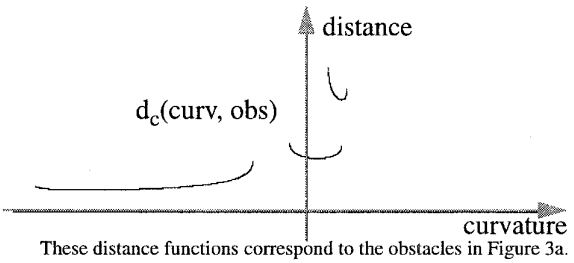
Figure 3: Curvature Interval Constraints

$(\langle c_{min}, c_{max} \rangle, d)$ (Figure 3a). An existing interval $(\langle c_1, c_2 \rangle, d_{1,2})$ is modified depending on its relationship with the new interval (Figure 3b):

- **Disjoint**: Do nothing
- **Contained by** (i.e., $c_{min} \le c_1$ and $c_2 \le c_{max}$): Set $d_{1,2} \leftarrow \min(d, d_{1,2})$
- **Contains** (i.e., $c_1 \le c_{min}$ and $c_{max} \le c_2$): If $d<d_{1,2}$, split the existing interval into three: $(\langle c_1, c_{min} \rangle, d_{1,2})$, $(\langle c_{min}, c_{max} \rangle, d)$ and $(\langle c_{max}, c_2 \rangle, d_{1,2})$, otherwise do nothing.
- **Overlapping**: If $d<d_{1,2}$, split the existing interval into two, with the result depending on which side of the intervals overlap occurs: either $(\langle c_1, c_{min} \rangle, d_{1,2})$ and $(\langle c_{min}, c_2 \rangle, d_{1,2})$ if $c_{max} > c_2$, otherwise $(\langle c_1, c_{max} \rangle, d)$ and $(\langle c_{max}, c_2 \rangle, d_{1,2})$.

Approximating the distance function as constant between the tangent curvatures is not very good, since d_v is really quite non-linear (Figure 4). In some situations the approximation is too conservative, especially where the distance values for the two tangent curvatures are very different (e.g., obstacle A in Figure 3a). More importantly, this approximation is often too liberal: the actual minimum may be less than either tangent curvature distance (e.g., obstacle B).

To remedy both these problems, we refine the approximation of d_v to itself be piecewise constant. The



These distance functions correspond to the obstacles in Figure 3a.

Figure 4: Representative Obstacle Distances

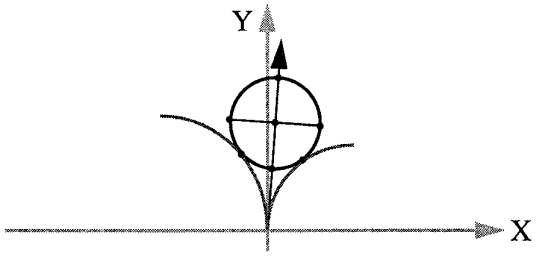


Figure 5: Dividing Obstacles Into Quadrants

idea is to break the interval $\langle c_{min}, c_{max} \rangle$ into a small number of intervals, calculate a constant distance value for each interval, and then use the min-union algorithm described above to approximate the overall D_{limit} function. Our approach is to pick the point on the obstacle circle that is closest to the origin (straight-line distance) and divide the obstacle into quadrants starting at that point (Figure 5). Curvature intervals are defined for adjacent points lying between the minimum and maximum tangent curvatures, where the distance value of each interval is assigned to be the minimum of the two endpoint intervals (calculated using Eqn. 4).

To maximize the objective function, we note that each curvature interval provides a pair of linear inequalities on the velocity space. The velocity and acceleration constraints described in the previous section are also linear inequalities. Thus, we have a set of linear inequalities and a linear objective function – a form easily solvable, in general. In our case, however, there is additional structure to the problem that can simplify calculations even further: since the distance value is constant between pairs of curvature lines and the objective function is monotonically increasing in *tv*, the optimal value of the function, for each curvature interval, lies along the top boundary of the constraint lines. This leads to a very efficient algorithm: for each curvature interval, compute the objective function at each vertex along the upper boundary of the relevant constraints and choose the overall best value for all the intervals (with one small extension: we also need to compute the objective function where the heading error $d\theta$ is zero, that is, where $rv = \theta_g / T_c$).

5 Extensions

The curvature-velocity method, as described in the previous two sections, has a few practical problems. First, due to sensor noise, obstacles may not be precisely where they are represented internally - thus, we want the robot to stay clear of obstacles when possible, and to slow down if it is unavoidable to travel near them. Second, although the objective function usually does a good job of trading off goal-directed and obstacle-avoiding behaviors, there are two situations where it does not work well: when all choices are equally bad, indicating the robot is stuck, and when the robot is far off the desired goal heading. In this section, we describe straightforward extensions to the basic curvature-velocity method that address each of these problems.

One simple extension, which helps compensate for sensor noise, is to grow the obstacles by a safety margin. We use a relatively small safety margin (5 cm), since the size of the safety margin is inversely proportional to how narrow an opening the robot can pass through. Too large a safety margin and the robot has trouble navigating through doorways and past people in crowded corridors.

Another extension, adapted from [3, 6], that helps keep the robot away from obstacles (or, at least, makes it act more cautiously when traveling near obstacles) is to add a constraint that makes the maximum allowable translational velocity proportional to the distance to obstacles. Specifically, for each curvature interval $(\langle c_1, c_2 \rangle, d)$, we constrain the maximum translational velocity $tv \leq d / T_{imp}$ so that the robot will be able to travel at least T_{imp} seconds before hitting an obstacle.

For some curvature intervals $(\langle c_1, c_2 \rangle, d)$, the distance d is greater than one, or both, of the extreme points $d_c(c_1, obs)$ and $d_c(c_2, obs)$, indicating that the robot will pass nearby an obstacle if it travels along that curvature extrema (e.g., d_1, the value of interval $\langle c_1, c_3 \rangle$ in Figure 3, is greater than $d_c(c_3, B)$). Although adding a safety margin helps, we further increase the safety factor by adding a constraint such that the robot travels at full speed only when it is at least distance S (e.g., one robot radius) away from all obstacles.

We determine this constraint by first calculating the curvature that will pass S away from the obstacle that is tangent to the curvature c at distance d:

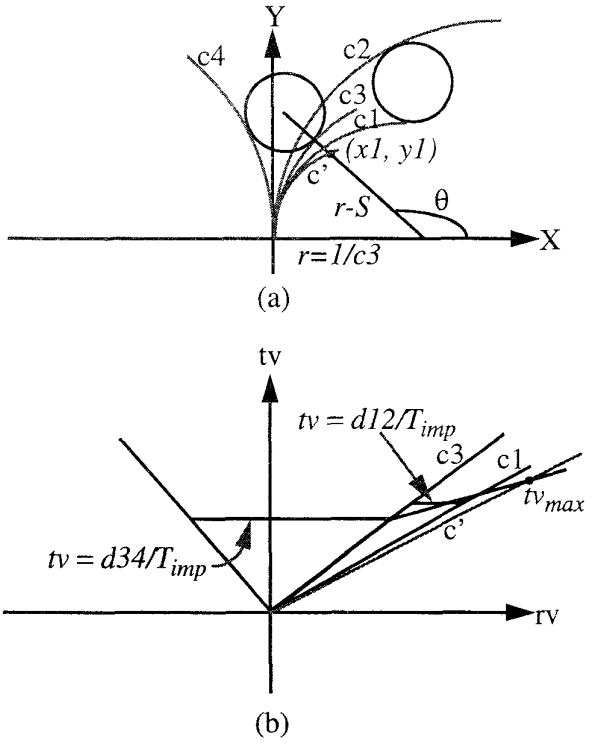


Figure 6: Obstacle Safety Constraints

$$\text{if } (c = 0)\ \{x = S; y = d\} \text{ else}$$

$$\{\theta = \begin{cases} -d \cdot c & c < 0 \\ \pi - d \cdot c & \text{otherwise} \end{cases}$$

$$r = 1/c \qquad (9)$$

$$x = (r - S)\cos\theta + r$$

$$y = (r - S)\sin\theta\}$$

$$c' = 2 \cdot x/(x^2 + y^2)$$

Then, we add the constraint that the translational velocity falls below the line formed by the points $(d \cdot T_{imp},\ c \cdot d \cdot T_{imp})$, $(tv_{max},\ c' \cdot tv_{max})$:

$$tv \le \frac{T_{impact} \cdot tv_{max} - d}{T_{impact} \cdot tv_{max} \cdot c' - c \cdot d}(rv - tv_{max} \cdot c') + tv_{max} \qquad (10)$$

This ensures that the robot will not travel at full speed (tv_{max}) unless it passes at least S away from the obstacle (Figure 5b). A similar constraint, using $(r + S)$ in Eqn. 9, is added for the lower bound of the curvature interval.

Occasionally, the best value of the objective function is to move very slowly, or not at all ($tv \approx 0,\ rv \approx 0$). This often occurs, for instance, when the robot is surrounded on three sides by obstacles and its goal heading is straight ahead, into the obstacle field. To handle such situations, we adopted the "rotate away" method of [6], in which the robot just stops and rotates in place when the best translational velocity command is nearly zero. This allows the robot to eventually "see" open space in front of it and move forward, hopefully enough to move it out of the local trap.

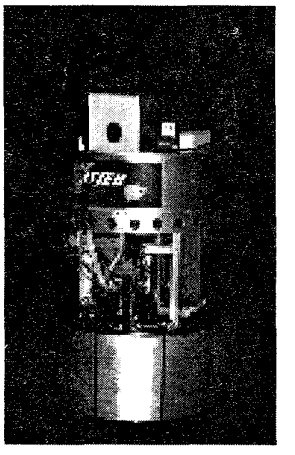


Figure 7: The Xavier Mobile Robot

The final extension is necessitated by our choice of weighting parameters for the objective function. The chosen weights work well when the robot close to facing the goal direction (within ±60 degrees), but performs poorly when it is facing in the opposite direction. In such cases, we want to "strongly encourage" the robot to start turning around to face the goal direction. We do this by increasing the weight of the goal-heading term proportional to how far the robot is from the goal direction. In particular, we replace α_3 in the objective function (Eqn. 5) with $\alpha_3(1 + \alpha_4(\theta_g/\pi)^2)$. The next section discusses the sensitivity of the method to this parameter value.

6 Results

The curvature-velocity method (CVM) has been implemented and tested extensively on the Xavier mobile robot (Figure 7). Xavier is built on a four-wheel synchro-drive base, produced by RWI, and has independent control over translational and rotational velocities. For obstacle detection, it uses a ring of 24 sonars (data rate of 1-2 Hz) and a 30 degree field of view front-pointing Nomadics laser range sensor (data rate of 5-10 Hz). The sonar and laser data are combined and filtered using a simple 20 cm resolution histogram grid [2].

The base provides Xavier with dead-reckoning information at 8 Hz, which is the rate at which the CVM algorithm is run. When a new report is received, each occupied grid cell is converted to ego-centric coordinates and processed to produce curvature constraints. There are typically between 15-30 obstacles, which yield about 10-15 distinct curvature

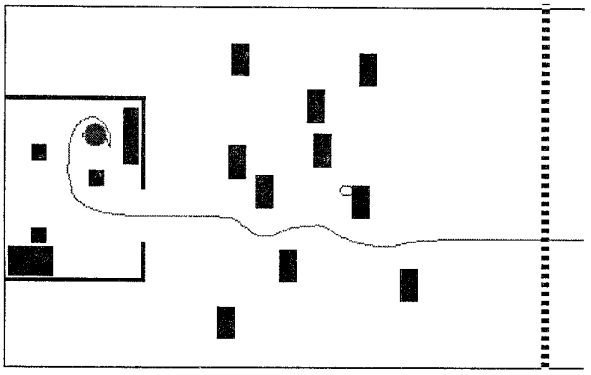


Figure 8: Testing Sensitivity of CVM Parameters

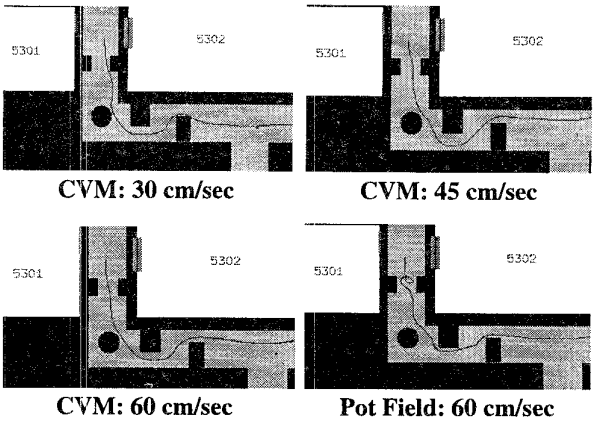


Figure 9: Traces of Xavier's Runs

intervals (the number of intervals is fewer than the number of obstacles due to overlapping intervals). The appropriate velocity constraints are added, and the (*tv, rv*) point with the best evaluation is sent to the base computer. The algorithm, including sensor processing, takes about 12 msecs, running on-board a 66 MHz 486 computer.

CVM depends on a number of parameters, in particular those of the objective function (Eqn. 5). To determine the sensitivity of the method to these parameters, we ran a series of tests in a simulated environment (Figure 8). The environment was set up to test the method's ability to both head out of local minima and to avoid obstacles while pursuing a goal direction. In each trial, the robot started in the position and orientation shown, and was told to head straight ahead; A trial was completed when the robot crossed the dotted line.

The parameters were systematically varied, with a number of trials run at each setting. Based on the average time to finish a trial, we conclude that the method is rather sensitive to the relative value of the goal heading weight (α_3), but is more or less insensitive to the relative values of the distance (α_2) and progress (α_1) weights. The method fared badly if α_3 was zero (aimless wandering) or if the ratio $\alpha_3/(\alpha_1 + \alpha_2)$ was greater than about 15% (the robot often got trapped in local minima). Within that range, the standard deviation within a trial was typically greater than the different between its mean and that of the best setting, although the average was slightly better with $\alpha_1 > \alpha_2$.

Similar trials were run varying α_4, the "extra" weighting that encourages the robot to turn when it is too far off heading. The results showed that CVM is relatively insensitive for values of α_4 greater than zero but less than about 3 (at which point the robot again tends to get trapped in local minima). The best setting for this environment is α_1=0.6, α_2=0.3 and α_3=0.1, with α_4=1.0 (average completion time: 67.5s, std: 10.2s), which is what we use for the results in Figure 8 and on Xavier, reported below.

Traces of various runs on Xavier are shown in Figure 9. To better accentuate the differences, in the figure the obstacles (two boxes, a round trash can, and a fire door) are grown by the robot's radius, and the robot is a point. In each case, the robot starts at the top of the figure, facing towards the bottom, and is commanded to head 90 degrees to its left. In each case, the robot must skirt the wall and navigate around three discrete obstacles. We compare CVM running at 30, 45, and 60 cm/sec, and an implementation of the potential field approach [1] running at 60 cm/sec. Note that, overall, CVM produces noticeably smoother paths than the potential field method. In addition, at higher speeds the robot stays further away from obstacles, and the paths are somewhat smoother. This is mainly due to the fact that the "near obstacle" constraints (Eqn. 9) become more important at higher speeds, which make the robot respond to objects earlier.

The local obstacle avoidance algorithm is used by several higher level behaviors. "Wandering" is achieved by always setting the local goal heading (θ_g) to zero, which biases the robot to continue along its current heading. A "head in direction" behavior is achieved by setting θ_g to the difference between the current robot heading and the desired (global) goal direction. Finally, a "go to goal" behavior is implemented by transforming the global goal location to the robot coordinate frame and setting θ_g to be the angle between the robot heading and the local goal point. For this last behavior, we need one additional extension: If the goal point falls within a curvature interval whose associated distance is less than the straight-line distance to the goal (i.e., there are no intervening obstacles), then the weight for the goal-heading term (α_3) is set very high to "strongly encourage" the robot to head towards the goal.

These behaviors, in turn, form the basis for map-based navigation schemes [10, 11]. The resulting system exhibits

reliable, and speedy, navigation in peopled office/corridor environments.

7 Conclusions

We have presented the curvature-velocity method for local obstacle avoidance, which treats the problem as one of constrained optimization in the velocity space of the robot. Advantages of this formulation include the ability to simultaneously control the speed and heading of the robot, the ease of incorporating constraints from both the environment and the robot dynamics, and the ability to handle trade offs between speed, safety, and goal-directedness.

CVM achieves real-time performance by approximating how far the robot could travel along a given curvature before hitting an obstacle. The approximation is a piecewise constant function, defined by the curvatures tangent to the obstacles. Additional velocity constraints are added based on the physical limits of the robot and the desire to keep it away from obstacles, or at least to travel slowly when passing nearby obstacles.

The method has been implemented and tested on Xavier, a synchro-drive robot (it is also applicable to differentially steered vehicles and to non-holonomic vehicles that travel along circular arcs). The implementation is quite efficient, and enables the robot to travel safely in peopled office environments at speeds up to 60 cm/sec. The limiting factor in speed appears to be the rate of the sonar sensors (the limited field of view of the laser prevents it from being the primary obstacle detection sensor). By improving the cycle time of our sensors, we expect to approach one meter per second travel.

Future work based on this method includes finding better approximations to the distance function and trying out more sophisticated objective functions. We would also like to investigate extending the curvature-based velocity space approach to multi-step navigation planning problems.

This work has shown that by taking vehicle dynamics into account, and by maximizing an objective function that trades off speed, safety and goal-directedness, we can create an efficient, real-time local obstacle avoidance algorithm that produces safe, smooth, speedy travel in obstacle-strewn environments.

8 Acknowledgments

Many thanks to Lonnie Chrisman, Richard Goodwin, Sven Koenig, and Sebastian Thrun for suggestions on the design and implementation of CVM. Thanks to Greg Armstrong for running experiments and keeping Xavier healthy. This research is sponsored by the Wright Laboratory, Aeronautical Systems Center, Air Force Materiel Command, USAF, and the Advanced Research Projects Agency (ARPA) under grant number F33615-93-1-1330.

9 References

[1] R. C. Arkin. Motor Schema-Based Mobile Robot Navigation. *International Journal of Robotics Research*, August 1989, pp. 92-112.

[2] J. Borenstein and Y. Koren. The Vector Field Histogram - Fast Obstacle Avoidance for Mobile Robots. *IEEE Transactions on Robotics and Automation*, **7:3**, 1991, pp. 278-288.

[3] J. Buhmann, W. Burgard, A. B. Cremers, D. Fox, T. Hofmann, F. Schneider, J. Strikos and S. Thrun. The Mobile Robot Rhino. *AI Magazine*, **16:2**, Summer 1995, pp. 31-38.

[4] P. Jacobs and J. Canny. Planning Smooth Paths for Mobile Robots. *In Proc. IEEE Intl. Conference on Robotics and Automation*, Scottsdale AZ, May 1989, pp. 2-7.

[5] W. Feiten, R. Bauer and G. Lawitzky. Robust Obstacle Avoidance in Unknown and Cramped Environments. *In Proc. IEEE Intl. Conference on Robotics and Automation*, San Diego, CA, May 1994, pp. 2412-2417.

[6] D. Fox, W. Burgard and S. Thrun. The Dynamic Window Approach to Collision Avoidance. Tech Report IAI-TR-95-13, CS Department, University of Bonn, 1995.

[7] A. Kelly. An Intelligent Predictive Control Approach to the High Speed Cross Country Autonomous Navigation Problem, Tech Report CMU-CS-TR-95-33, School of Computer Science, Carnegie Mellon University, 1995.

[8] O. Khatib. Real-time Obstacle Avoidance for Manipulators and Mobile Robots. In *Proc. IEEE Intl. Conference on Robotics and Automation*, St. Louis, MO, March 1985, pp. 500-505.

[9] J. C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[10] R. Simmons, Becoming Increasingly Reliable. In *Proc. of 2nd Intl. Conference on Artificial Intelligence Planning Systems*, Chicago, IL, June 1994.

[11] R. Simmons and S. Koenig. Probabilistic Navigation in Partially Observable Environments. In *Proc. Intl. Joint Conference on Artificial Intelligence*, Montreal, Canada, August 1995, pp. 1080-1087.