Journal of Mechanical Design

Technical Brief

A Geometric Path Planner for Car-like Robots

Shiang-Fong Chen
Assistant Professor

Jiansong Deng[1]
Research Associate

Department of Mechanical and Aerospace Engineering, Tri-State University, 1 University Avenue, Angola, IN 46703

This technical brief presents a refined slabbing method, originally used for free-flying robots, for finding efficient paths for nonholonomic robots. Our method takes kinematic constraints and reversal maneuvers into account. We create orientation levels for each orientation configuration of the robot. The slopes of slabbing lines in each orientation level match the orientation of a robot in that level. The resulting slabbing lines act as ''rails'' to guide the robot. Thus, a robot, if it keeps moving in a given orientation level, can only translate straight forward or straight backward along a given slabbing line. Limiting robot movement to straight forward or straight backward along a slabbing line prevents the robot from violating kinematic constraints, by moving sideways to another slabbing line. [S1050-0472(00)01403-3]

1 Introduction

There are two classes of robots. One class, ''free-flying'' robots (also called *holonomic* robots), can rotate and translate freely in their workspace. The other class, ''car-like'' robots (also called *nonholonomic* robots), move under given kinematic constraints, such as minimum turning radius. Most holonomic robot path planners successfully use the configuration space (C-space) approach to find acceptable solution paths [1]. However, an acceptable path for a free-flying robot may not be a feasible path for a car-like robot.

Barraquand and Latombe [2] find a nonholonomic path by decomposing the C-space into an array of small rectangloid cells and considering kinematic constraints between each pair of adjacent cells. Jiang et al. [3] and Svestka and Overmars [4] find a nonholonomic path directly in the workspace.

Other nonholonomic path planners first generate a non-feasible collision-free path by ignoring kinematic constraints. Then, they transform that path into a feasible one by using a local planner, e.g., Reeds and Shepp curves [5]. Finally, such planners optimize the resulting path [6]. Path optimization usually involves a significant amount of complicated control theory and differential geometry.

Our proposed method solves nonholonomic robot path planning by using a simple refined slabbing method. We refine slabbing methods originally developed for free-flying robots [7,8]. In contrast to the horizontal slabbing lines used for free-flying robots, in our approach, for car-like robots, we use slabbing lines with slopes which are different for different orientation levels. Using our approach, kinematic constraints can also be satisfied.

Section 2 briefly describes the slabbing method for free-flying robots [8]. Section 3 then presents our refined slabbing method for car-like robots.

2 Slabbing Method for Holonomic Path Planning

Step 1. Building Multiple Levels of C-Spaces. Successive configuration spaces are computed for every δ radians of angular rotation spanning from $-\pi/2$ to $\pi/2$. Each δ is referred to as a *rotation interval*, and successive configuration spaces are referred to as *rotation* (*or orientation*) levels.

Step 2. Building a Passage Network for Each Level. For each orientation level, the Chen et al. [8] algorithm finds an associated passage network by slabbing collision-free space. Slabbing lines, for all orientation levels, are *horizontal* lines in the Cartesian (x-y) world coordinate system; slabbing lines slab the environment, in non-increasing y-coordinate order, at all C-space obstacle vertices. Each slabbing line creates one or more gates, horizontal line segments connecting a C-space obstacle vertex with: another C-space obstacle vertex, a C-space obstacle boundary, or a C-space boundary. The passage network is built by connecting midpoints of adjacent gates. The bold lines in Fig. 1 show an example passage network. The passage network in each orientation level is the collision free translation locus for a robot in that orientation.

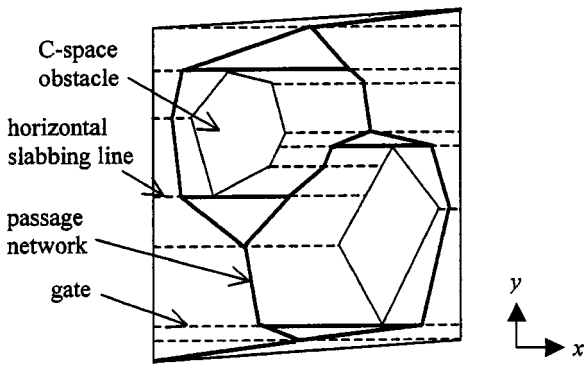

Fig. 1 Passage network for one orientation

[1]On leave from the Department of Mathematics, University of Science and Technology of China

Contributed by the Mechanisms Committee for publication in the JOURNAL OF MECHANICAL DESIGN Manuscript received May 1999. Associate Technical Editor: C. Innocenti.

 Copyright © 2000 by ASME

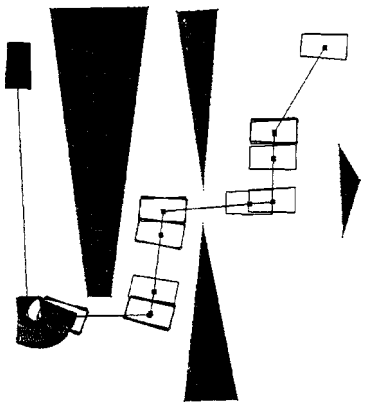

Fig. 2 Moving a free flying robot by slabbing method

Step 3. Building a 3D Network. Individual configuration spaces are then connected to one another using ''proper links,'' that maintain spatial coherence. The passage network in each level is connected, by proper links, to adjacent levels for constructing a 3D network. Using proper links guarantees that a robot will not ''jump'' spatially from position to position when moving from orientation level to orientation level. Robot rotation takes place when path searching moves from one orientation level to another orientation level. Dijkstra's algorithm is used to search for the shortest path in the 3D network. Finally, the searched path is projected onto a plane to show the final path locus. Figure 2 shows one example path.

3 Slabbing Method in Nonholonomic Path Planning

Observation. Suppose the minimum turning radius of a car-like robot is R. If a car makes a turn of $d\theta$ radians from P_1 to P_2, with minimum turning radius R, the car will travel a distance $R(d\theta)$. If a car moves straight from P_1 to P_2, car travel distance is $2R\cdot\sin(d\theta/2)$, which is the chord length between P_1 and P_2. We know that when $d\theta$ is very small, $2R\cdot\sin(d\theta/2)\cong R(d\theta)$. Thus, we can approximate a circle by a $[2\pi/(d\theta)]$-sided polygon, with sides having equal length $2R\cdot\sin(d\theta/2)$. Thus, if a robot moves around a circle with radius R, we can approximate the continuous rotational motion by discrete translation motions moving along the edges of a $[2\pi/(d\theta)]$-sided polygon, with the robot changing orientation at each vertex of the polygon. This observation gives us a clue for applying the slabbing method to car-like robot path planning.

3.1 Slabbing Algorithm

Step 1. Building Multiple Levels of C-Spaces. Since we allow both forward and backward motion, we compute configuration spaces for every δ radians of angular rotation spanning from $-\pi$ to π. Here, we assume a car-like robot with a rectangular shape.

Step 2. Revised Slabbing Lines for Each Orientation Level Instead of using horizontal slabbing lines, each C-space orientation level is slabbed using lines with the same orientation as the robot main axis. For a free-flying robot, the Chen et al. [8] algorithm finds a passage network which defines the orientation-level robot translation locus. However, for a car-like robot, we do not construct a passage network. Instead, for a car-like robot, we consider orientation-level slabbing lines as candidate translation loci. The slabbing lines, then, act as ''rails'' for robot translation motion.

In each orientation level, slabbing lines are equally spaced. Given two adjacent orientation levels, level (θ) and level $(\theta+d\theta)$, slabbing lines in level (θ) are represented by $L_{(\theta)}: y=\tan\theta\cdot x+c_i$, and slabbing lines in level $(\theta+d\theta)$ are represented by $L_{(\theta+d\theta)}: y=\tan(\theta+d\theta)\cdot x+c_j$ in the x-y world coordinate system (see Fig. 3); c_i and c_j establish the offset distance between adjacent parallel slabbing lines. Slabbing lines do not necessarily pass through the vertices of C-space obstacles. In order to approximate a reasonable path, the angular difference $d\theta$ between two adjacent levels should be as small as possible. We might select $d\theta$ for the robot and obstacle environment at hand.

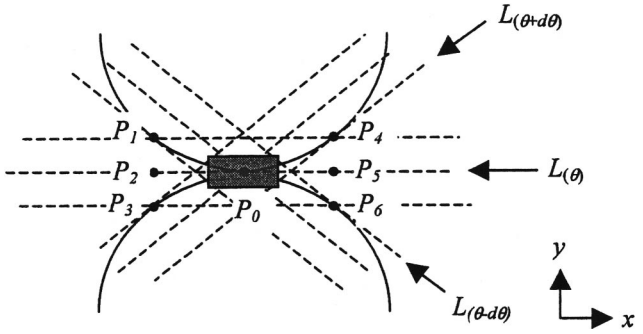

Fig. 3 Revised slabbing lines

To find c_i and c_j, we define a local coordinate system (u, v), located at the center of the car. The u-axis is perpendicular to the main axis of the car, and the v-axis is in the same direction as the main axis. Once rotation interval $d\theta$ and minimum turning radius R are given, step sizes along the u- and v-directions can be determined:

$$du = R(1-\cos d\theta)$$

$$dv = R\sin d\theta.$$

Figure 4 shows step sizes, du and dv. Thus, spacing between slabbing lines will be du for every level. We determine that dv is the unit translation step size for translation motion and $d\theta$ is the unit rotation step size for rotation motion. The unit step size dv for translation motion produces ''grids'' on the slabbing lines (e.g., P_0, P_2, and P_5 in Fig. 3).

When the robot translates forward or backward, it only moves along the same straight slabbing line and stays on the same rotation level. In other words, the robot can only move on the same ''rail'' (slabbing line); the robot cannot slide sideways onto other rails, even though the other rails are in the same rotation level. Our rule limiting robot translation to only forward or backward along the same slabbing line prevents the robot from violating kinematic constraints by moving sideways. For example, in Fig. 3, the current position for the robot is P_0, and, if the robot moves by translation only, it can only move along the line passing through grids P_2 and P_5 at level $L_{(\theta)}$. If the robot wants to keep the same orientation θ, but shift sideways (e.g., the line passing through grids P_1 and P_4), it should perform one reversal maneuver: rotate to another level (e.g., $L_{(\theta+d\theta)}$), translate along a new orientation rail, rotate back to level $L_{(\theta)}$ again, and then translate to the desired position.

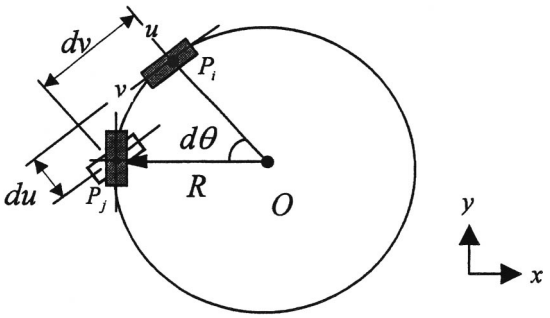

Fig. 4 Step sizes for the robot

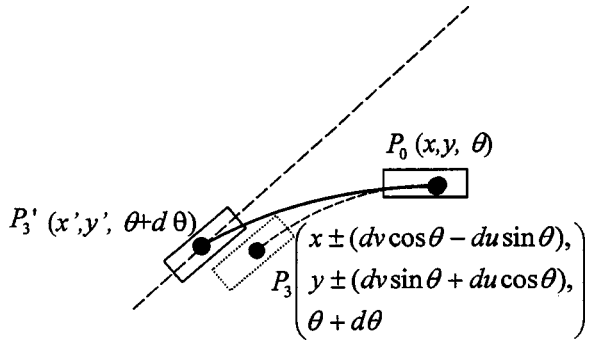

Fig. 5 Choose the closest grid point

Before we begin to search for a path, we first need to setup grid points for each orientation level. Since orientation-level grid points define ''stops'' for the robot, after applying any path-searching algorithm, the resulting path is composed of visited grid points.

We first setup grid points for level $\theta = 0$, then we rotate them to get the grid points for other orientation levels. Suppose G_{ij}^0 are grid points for $\theta = 0$, then the grid points for any θ level are

$$G_{ij}^{\theta} = G_{ij}^0 \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

Step 3. Searching for a feasible path. When the robot moves, it has six possible positions for its next step: two from the same orientation level (θ), (e.g., grid points P_2 and P_5 in Fig. 3), two from level ($\theta + d\theta$), (e.g., grid points P_3 and P_4 in Fig. 3), and two from level ($\theta - d\theta$), (e.g., grid points P_1 and P_6 in Fig. 3). Each step is assigned a cost. The next-step grid point chosen by the robot should be in free space. Any path-searching algorithm can be used to find the final minimum-cost path. When rotation interval $d\theta$ is small enough, such as 10 deg, we just need to connect grid points in order, and the resulting polyline will be the path.

3.2 Discussion. During the path-searching process, when the robot makes a turn, the new position $(x \pm (dv\cos\theta - du\sin\theta), y \pm (dv\sin\theta + du\cos\theta), \theta + d\theta)$ or $(x \pm (dv\cos\theta + du\sin\theta), y \pm (dv\sin\theta - du\cos\theta), \theta - d\theta)$ might not be a grid point on levels $\theta \pm d\theta$. If the next step is not a grid point on a neighboring orientation level, we need to find the grid point which is closest to it on level $\theta \pm d\theta$. The grid point selected must meet the kinematic constraints of the robot, e.g., lie at a point outside the robot minimum turning radius (see Fig. 5).

In order to get a smooth path for the robot, $d\theta$ should be as small as possible. For an $l \times l$ C-space, the total number of grid points for all levels will be $O((l/du)(l/dv)(2\pi/d\theta))$; that is $O(l^2(1/d\theta)^4)$. Thus, if $d\theta$ is very small, required memory and run time increases dramatically. Two methods can help reduce memory use and run time. First, we can use a large $d\theta$ to find a ''coarse'' path, and then refine the path by searching regions near the coarse path using a smaller $d\theta$. However, since a large $d\theta$ reduces robot maneuverability and results in a coarse path which occupies larger space, using a large $d\theta$ increases our risk that the robot will not find a path, even when a solution path does exist. Another method for reducing memory use and run time is to use any holonomic path planner to find a path, and then to convert the holonomic path to a nonholonomic path using our proposed algorithm.

4 Implementation

Our revised slabbing nonholonomic path-planning algorithm has been implemented in C++, with OpenGL for visualization, on a Pentium II 450MHz PC. In order to speed up computation time, first, we use $d\theta = 90$ deg to find a coarse path. Then, we re-grid the configuration space, using $d\theta = 30$ deg, and search areas near the coarse path to refine the path. After re-gridding, each orientation level contains 100×100 grid points, and, in our implementation, we use 12 orientation levels, $\{[(\pi - (-\pi)]/d\theta\} = 12$. Figure 6 presents four examples to demonstrate algorithm performance. Figure 6(*a*) shows a robot moving in an environment without any obstacles. Our chosen robot start position orientation is facing downward and our chosen robot goal position orientation is facing upward. Thus, the robot needs one reversal maneuver. Run time for example 6(*a*) is 8 seconds. In Fig. 6(*b*) we use the same start and goal positions as in Fig. 6(*a*), but one obstacle exists between the start and goal positions. Run time for example 6(*b*) is 13 seconds. Figure 6(*c*) has the same environment as Fig. 2, but shows results when using our revised slabbing nonholonomic path planning algorithm (as opposed to the holonomic path planning algorithm used for the free-flying robot in Fig. 2). Run time for example 6(*c*) is 35 seconds. Figure 6(*d*) uses a more complicated environment, and its run time is 120 seconds.

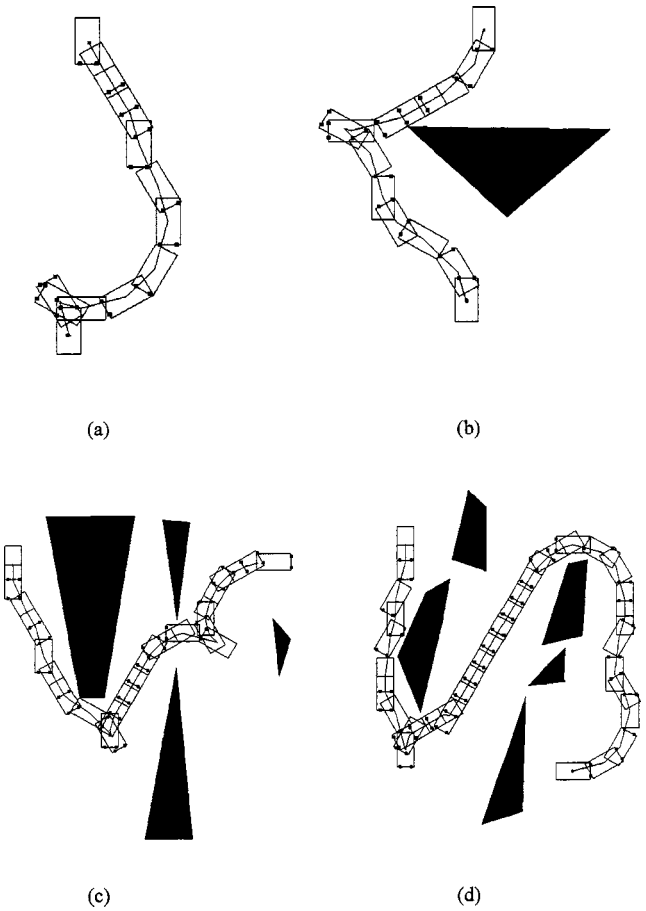

Fig. 6 Implementation examples

5 Conclusions

This technical brief presents a geometric approach for finding a nonholonomic path for a mobile robot. A slabbing method is successfully applied to path planning for a car-like robot. Kinematic constraints are taken into account. Several reversal maneuvers are allowed. Our proposed algorithm is simple and easy to implement, but algorithm run time strongly depends on orientation interval $d\theta$. Algorithm run time is $O(l^2(1/d\theta)^4)$, where l^2 is the area of the workspace. Thus, to reduce computation time, either a coarse path or a holonomic path can be found first, then, to get a smoother nonholonomic path, the coarse or holonomic path can be further refined using our proposed revised slabbing method.

References

[1] Lozano-Perez, T., 1983, ''Spatial Planning: A Configuration Space Approach,'' IEEE Trans. Comput., **C-32**, No 2,.

[2] Barraquand, J., and Latombe, J. C., 1990, ''Controllability of Mobile Robots with Kinematic Constraints,'' Technical Report No. STAN-CS-90-1317, Dept. of Computer Science, Stanford University.

[3] Jiang, K., Seneviratne, L. D., and Earles, S. W. E., 1996, ''Motion Planning with Reversal Manoeuvres for a Non-Holonomic Constrained Robot,'' Proc. ImechE J. Eng. Manuf., **210**, pp. 487–497.

[4] Svestka, P., and Overmars, M. H., 1998, ''Probabilistic Path Planning,'' *Robot Motion Planning and Control*, Laumond, J-P, ed., Springer, New York.

[5] Reeds, J. A., and Shepp, L. A., 1990, ''Optimal Paths for a Car that Goes Both Forwards and Backwards,'' Pac. J. Math., **145**, No. 2, pp. 367–393.

[6] Laumond, J-P., Jacobs, P., Taix, M., and Murray, R. M., 1994, ''A Motion Planner for Nonholonomic Mobile Robots,'' IEEE Trans. Rob. Autom. , **10**, No. 5, October pp. 577–593.

[7] Ahrikencheikh, C., Seireg, A. A., and Ravani, B., 1994, ''Optimal and Conforming Motion of a Point in a Constrained Plane,'' ASME J. Mech. Des., **116**, No. 2, pp. 474–479.

[8] Chen, S. F., Oliver, J., and Fernandez-Baca, D., 1998, ''A Fast Algorithm for Planning Collision-Free Paths with Rotation,'' ASME J. Mech. Des., **120**, pp. 52–57.