2009 IEEE International Conference on Robotics and Automation
Kobe International Conference Center
Kobe, Japan, May 12-17, 2009

On-line Planning of Nonholonomic Trajectories in Crowded and Geometrically Unknown Environments*

Yanbo Li and Jing Xiao[§], *Senior Member, IEEE*

***Abstract* — Navigation of a car-like robot in environments with unknowns requires effective on-line planning of nonholonomic trajectories. We propose a set of basic maneuver patterns based on Bezier curves that allow either forward or backward motion as building blocks to create nonholonomic trajectories quickly, given a sequence of knot positions/points (e.g., from some GPS navigator). These maneuver patterns are particularly useful for generating feasible trajectories in crowded environments with many narrow passages. We embed the above techniques in a new planner suitable for on-line planning of nonholonomic and collision-free trajectories, called the *ON planner*. Our ON planner enables that, given a sequence of rough knot points, a car-like robot can simultaneously plan and move in a geometrically unknown, crowded environment with local sensing towards a goal. Simulation results demonstrate the planner's nice capabilities.**

I. Introduction

Nonholonomic vehicles have attracted much study from the robotics research community in the last two decades. Existing research can be classified below:

- *Steering method to make a nonholonomic vehicle reach a pre-determined configuration*

Here the emphasis is to compute a nonholonomic path between two given configurations without taking into account obstacles (e.g., [1, 2, 3, 4]), with [4] also addressing optimal control to execute such a path. However, no algorithm can guarantee a solution to the problem [5].

- *Planning nonholonomic trajectories directly*

The main difficulty here arises from the differential constraints of the vehicle model [5]. Such constraints mean that there are fewer nonholonomic trajectory solutions than holonomic ones in the solution space. A nice survey of earlier work on planning nonholonomic paths can be found in [6]. RRT-based kinodynamic planners [6-8] are most common for searching nonholonomic trajectories in static or structured environments. However, such tree-based planners often cannot handle efficiently very crowded environments with many narrow passages, where the maneuver space for a vehicle is severely limited. A recent RRT-variant [9] addresses maneuvers but cannot deterministically control the vehicle orientation at the goal. There are also approaches for nonholonomic trajectory search based on evolutionary computation or genetic algorithms [10-12]. These planners use smooth operators or driving methods [10] to achieve nonholonomic solutions. However, they usually focus on finding an entire path/trajectory satisfying the differential constraints globally, which is time-consuming and not suitable for on-line planning.

- *Converting holonomic paths to nonholonomic trajectories*

In [13], an approach was introduced to convert a holonomic path to a nonholonomic one through multiple levels involving local search. However, the approach is for off-line conversion of an entire path and is not efficient for on-line operation. It also assumes known environments.

A major limitation of the existing nonholonomic motion planners is the assumption of known and mostly static environments. In addition, most of the approaches only assume the Dubin's model [14] without considering backward driving of a nonholonomic vehicle.

The work in this paper is inspired by the problem that a robotic vehicle moves autonomously in an unknown environment guided by a GPS navigator (which is already common for automobiles) and local sensing. The GPS navigator system usually provides only rough information of the environment in the form of a map, which does not indicate the actual geometry of the road as well as temporary (i.e., changeable or moving) or small obstacles. Thus, the robotic vehicle has to rely on local sensing to gain the geometric information, based which, it has to conduct on-line planning to produce collision-free nonholonomic trajectories for the robot to follow.

We introduce an on-line nonholonomic planning and navigation system that allows the robot to do the above with on-line sensing of limited range. The main idea of our method is to introduce a set of parameterized, basic maneuver patterns that the planner can use to build nonholonomic trajectories with arbitrary turns quickly in a piece-wise fashion. The maneuver patterns use Bezier curves, which allow both forward and backward driving of a vehicle with greater flexibility than using Reeds and Shepp curves [15]. It is easy to specify and modify Bezier curves analytically via control points for on-line planning, which is an advantage over clothoid [16]. Note that allowing backward driving is not only necessary for making sharp turns (as in parallel parking) but also desirable in some cases for efficiency. After all, backward driving is only awkward for humans (because we have no eyes on our back) but should not be so for a robot, which can easily move forward (i.e., following the traffic) while facing back. Simulation results demonstrated nice performance of the novel planner.

*This work was supported in part by the U.S. National Science Foundation under Grant IIS-0742610.

§The work was conducted at the IMI Lab, Department of Computer Science, University of North Carolina – Charlotte, Charlotte, NC 28223, USA. E-mails: yli54@uncc.edu, xiao@uncc.edu.

II. Constructing Nonholonomic Trajectories

In this section, we describe the issues and the key ideas of our approach to generate trajectories with nonholonomic constraints. We first review a nonholonomic vehicle model and then present the key idea: use a set of basic maneuver patterns consisting of straight-line and Bezier curve segments and allow both forward and backward motions to enable efficient on-line generation of nonholonomic trajectories in crowded environments.

A. Nonholonomic Vehicle Model

A mobile robot is nonholonomic if it has fewer controllable degrees of freedom than the dimensions of its configuration space. For a nonholonomic car-like vehicle, shown in Fig. 1, its reference position (x,y) between the rear wheels and orientation θ satisfy the following constraint:

$$-\dot{x}\sin\theta + \dot{y}\cos\theta = 0 \tag{1}$$

From the geometry and bounded linear velocity of the vehicle, we can derive the lower bound of the turning radius R of the vehicle.

The maximum turning angle φ_{max} of the front wheels determines a minimum geometry-based turning radius:

$$R^{g}_{\min} = \frac{L_d}{\tan(\varphi_{\max})} \tag{2}$$

The minimum velocity-based turning radious is determined by the current centripetal velocity v and the maximum acceleration a_{max}:

$$R^{v}_{\min} = \frac{v^2}{a_{\max}} \tag{3}$$

where a_{max} is determined by friction between the ground, the vehicle tires and the acceleration weight in the normal direction.

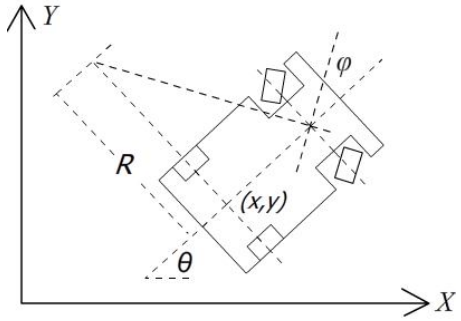

Figure 1. Nonholonomic vehicle model

Thus, the turning radius R for the robot at any point on a trajectory must satisfy:

$$R \geq \max(R^{g}_{\min}, R^{v}_{\min}) \tag{4}$$

B. Use of Bezier curves

The main problem here is how to obtain a nonholonomic trajectory in real-time, given a sequence of intermediate *knot points* that characterize a rough homotopic path segment. The key issue is how to construct a trajectory that each point on the trajectory satisfies the inequality (4). We propose the combination of linear and Bezier curve segments.

A Bezier curve is a smooth curve. In the parametric form, a 2nd order Bezier curve is defined as:

$$\left\{\begin{array}{l} f(t) = (1-t)^2 X_0 + 2t(1-t)X_1 + t^2 X_2 \\ g(t) = (1-t)^2 Y_0 + 2t(1-t)Y_1 + t^2 Y_2 \end{array}\right\} \tag{5}$$

where $t \in [0,1]$ is the parameter, and (X_0,Y_0), (X_1,Y_1), (X_2,Y_2) are control points of the Bezier curve, as shown in Figure 2.

The first order derivative is in the form:

$$\left\{\begin{array}{l} f' = At + C \\ g' = Bt + D \end{array}\right\}$$

where $A=2(X_0-2X_1+X_2)$, $B=2(Y_0-2Y_1+Y_2)$, $C=2(X_1-X_0)$, $D=2(Y_1-Y_0)$

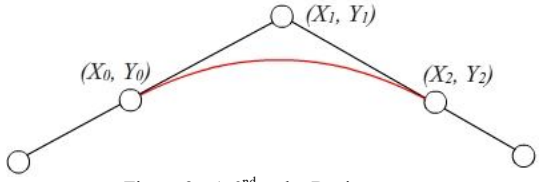

Figure 2. A 2nd order Bezier curve

The minimum radius along the curve happens at

$$t_{\min} = -\frac{AC + BD}{A^2 + B^2} \tag{6}$$

and can be expressed as:

$$R_{\min} = \sqrt{\frac{[(A^2+B^2)t^2_{\min} + 2(AC+BD)t_{\min} + (C^2+D^2)]^3}{(BC-AD)^2}} \tag{7}$$

Note that the radius of each point along a curve constrains the speed of a nonholonomic vehicle. The smaller the radius, the slower the speed has to be. A Reeds and Shepp curve that consists of line segments and circular arcs [14] has a constant speed limit on the arcs. On the contrast, for a Bezier curve, the minimum radius only happens at t_{min} around the middle of the curve. Thus, a properly parameterized Bezier curve (by adjusting the control points) has the advantage of allowing for higher entrance and exit speeds and more flexibility for the robot to adjust speeds. Hence, a trajectory of linear segments and Bezier curves allows higher vehicle speeds than that from a Reeds and Shepp curve.

C. Basic Maneuver Patterns

We propose 5 basic maneuver patterns as building blocks to enable nonholonomic turns. As shown in Figure 3, the three points P_{n-1}, P_n, and P_{n+1} indicate a turn formed by two line segments. The figure also shows four types of transition curves, which are Bezier curves with P_n as the common control point.

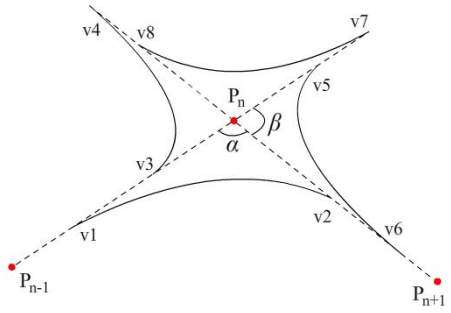

Figure 3. Four transition curve segments

We now describe how to use these four types of Bezier curves to construct maneuver patterns. For an arbitrary angle α formed by P_{n-1}, P_n, and P_{n+1}, as shown in Fig. 4, if α is sufficiently large, the straightforward turning curve v_1v_2 can be followed by a car-like robot to make the turn while satisfying inequality (4) under its velocity at P_{n-1}. Thus, the curve v_1v_2 is a maneuver pattern, called *maneuver 1*. Now if there is an obstacle to block the maneuver along v_1v_2, a different maneuver policy involving additional straight-line movements, called *maneuver 2*, can be used to achieve the same turn, as shown in Figure 4. Note that v_7v_8 is executed by the robot by moving backward. The pair of patterns *maneuver 1* and *maneuver 2* provide two alternatives for a car to achieve the same turn while satisfying the nonholonomic constraint, i.e., inequality (4).

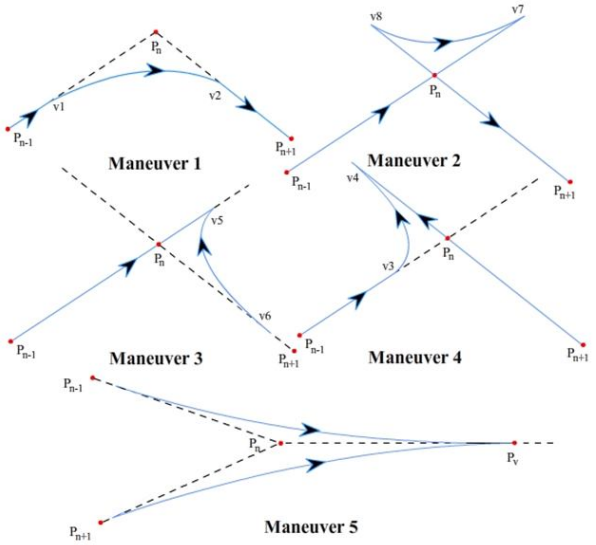

Figure 4. Five types of basic maneuver patterns, where the arrow shows the orientation of the robot. The robot changes motion directions from forward to backward or vice versa at each sharp intersection.

If α is too small so that the *maneuver 1* and *maneuver 2* patterns cannot be applied to the robot without violating the nonholonomic constraint, the patterns *maneuver 3* and *maneuver 4* (see Figure 4) can be used to make the turn by reversing the robot. The smaller α is, the larger the angle β is, and therefore, it is more likely that (4) will be satisfied. The two patterns show alternative ways of achieving the same result.

Maneuver 5, also shown in Figure 4, is another alternative pattern to execute a sharp turn by reversing the robot. Although reversing the robot may not meet the requirement of certain turn by itself, it can be a necessary intermediate movement that leads to a correct turn eventually. Figure 5 shows an example.

Note that each basic maneuver pattern can be used in two ways, with the robot driving either forward or backward.

III. Our On-line Planning Approach

Our on-line nonholonomic (ON) planner uses the novel maneuver patterns introduced above to find nonholonomic trajectories efficiently to reach a goal configuration based on the rough topological information of a sequence of knot points or positions (e.g., reading from a GPS navigator) and on sensing. It enables simultaneous planning and execution of robot motion in real-time.

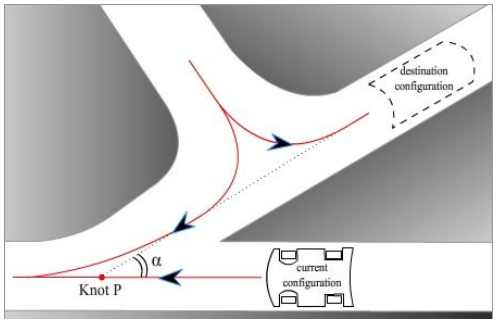

Figure 5. A *maneuver 3* and a backward *maneuver 5* enable the robot to make a sharp turn indicated by the knot point *P* on a path. The shaded area indicates obstacles.

A. *Overview of the ON planner*

Our planner first extracts a subsequence *s* of visible knot positions, i.e., within the sensing range, to generate a nonholonomic sub-trajectory γ (Section III.B), which starts from the initial configuration of the robot and ends near the last (i.e., the furthest visible) knot point[2] of *s*. It then initializes a set *P* of sub-trajectories by creating multiple copies of γ. Next it improves and diversifies *P* through iterations of *planning cycles* using a number of modification operations (Section III.E) and to get the best sub-trajectory γ_{best}. The goodness or fitness of a sub-trajectory is evaluated through an *evaluation function* coding certain optimization criteria (Section III.D). A sub-trajectory is *feasible* if it is collision-free and satisfies the nonholonomic constraints.

Once a feasible γ_{best} is generated, our system lets the robot follow γ_{best} while continuing sensing-based planning to both update and improve the set *P* of sub-trajectories. When the next *adaptation cycle* arrives, the robot can switch to a better sub-trajectory so that it always follows the best one. For that purpose each sub-trajectory is always updated (Section III.C) to start from the robot configuration and velocity at the end of the current adaptation cycle to enable smooth switch to a better trajectory at the beginning of a new adaptation cycle. Note that an adaptation cycle here only dictates when the robot can change to a better trajectory and is thus longer than a planning cycle. As the robot moves, its range of sensing changes too, like a sliding window. Planning adjusts to the change of sensing range from each *sensing cycle*[3] by updating the subsequence *s* of visible knot points and accordingly the set *P* of sub-trajectories (Section III.C) and by evaluating the feasibility and fitness values of the updated sub-trajectories. In this way, when a previous best sub-trajectory becomes worse or infeasible, the robot can quickly

[2] We only plan a sub-trajectory based on the knot points within the sensing range to save computation because the environment beyond that point forward is not perceivable.

[3] Note that a sensing cycle is different from an adaptation cycle because the former depends on the sensor(s) but the latter has to be longer than a planning cycle, i.e., depends on the planner.

switch to a different and better sub-trajectory in the next adaptation cycle. This also explains why our planner manages a set P of varied sub-trajectories instead of just a single sub-trajectory at any time.

If the best sub-trajectory is not feasible, the robot still follows it until the first infeasible segment is encountered. It may already switch to a better sub-trajectory before then, or it will stop moving but not stop planning until a better alternative is found and can be switched to. This form of interweaving sensing, planning, and adaptation is inspired by the RAMP approach [17,18] for mobile manipulators.

The overall structure of our system is shown in the following algorithm.

Algorithm 1: **ON planner**
Input: a sequence of knot points of a rough path
extract the 1st visible subsequence s of knot points;
obs <= sensed obstacle information;
generate a nonholonomic sub-trajectory γ from s;
evaluate γ and create n copies to form a set P;
improve P and return the best γ;
repeat:
  simultaneously do (1) and (2)
  (1) let the robot follow γ
  (2) begin (planning):
    if near the end of current adaptation cycle *then*
      update starting point of s &
          starting segments in P;
      evaluate P and get γ_{best};
      $\gamma <= \gamma_{best}$;
    end if;
    if new sensing cycle *then*
      obs ← sensed obstacle information;
      update s and P;
      evaluate P
    end if;
    improve P
    end (planning)
until the destination is reached.

The **improve** routine can be outlined below:

Algorithm 2: **improve***(P)*
repeat:
  modify a randomly chosen non-best sub-trajectory;
  evaluate its fitness and rank it in P;
until time T_{plan} elapsed;
return the best sub-trajectory.

In the following subsections, we will explain the major components **generate**, **update**, **evaluate** and **modify** in more details.

B. *Sub-trajectory generation*

Given a sequence s of knot positions, a corresponding nonholonomic trajectory is constructed in the following steps:

1) Use straight-line segments to connect the sequence of knot points between the start and end positions of s to produce a translational path.

2) At the start knot point, decide whether the robot should have its front or back facing the second knot point, depending on which way to drive the robot will minimize the orientation change of the robot to align itself with the first straight-line segment of the path from its very initial configuration. The segment between the initial configuration and the first knot point follows a Bezier curve with one additional control point.

3) Modify each connection of two adjacent straight-line segments along the path with one of the basic parameterized maneuver patterns introduced in section II. The modification takes into account the robot's velocity before entering the pattern, given that the robot moves as fast as possible on the straight-line segment leading to the maneuver pattern. For each connection, the planner tries the basic maneuver patterns 1—5 one by one in that order to see which pattern will lead to a feasible turn, i.e., collision free and satisfying the nonholonomic constraint inequality (4), from one straight-line segment to the next. It does not matter if the robot's orientation is reversed after the turn from forward to backward or vice versa. Note that maneuver patterns 3—5 reverse the robot's orientation, and note also that maneuver patterns 2—5 introduce additional knot points as Bezier control points. If no basic maneuver pattern can lead to a feasible turn at a knot point, the pattern that is either collision-free (but does not satisfy the nonholonomic constraint) or yields the least collision cost (see next subsection) is used.

Note that if two adjacent knot points on a path are close to each other, there can be an overlap of the corresponding maneuver patterns. Figure 6 shows an example, where the robot leaves the pattern of knot p at point B_{pe}, but the maneuver pattern for knot q starts from point B_{qs}. In such a case, the robot moves backward on the line segment connecting p and q before entering the pattern for q.

4) If from the last knot point of s, the goal configuration of the robot is visible, decide if the robot should enter the goal configuration in forward or backward fashion depending on which way minimizes orientation change from the robot's orientation at the last knot point to the goal configuration. Again, the corresponding segment follows a Bezier curve with one additional control point.

5) Next, the resulting curve of smoothly connected linear and Bezier segments is converted to a trajectory as a function of time. We compute the velocity and acceleration at each point of the curve and the corresponding duration of the curve numerically, subject to the nonholonomic constraint (4) and the maximum velocity and acceleration constraints of the vehicle. Since the curve (of a sub-path) is short and the method involves only simple additions and multiplications, the generation of the corresponding sub-trajectory is done very efficiently on-line. Note that such a

sub-trajectory may not be collision free and is subject to further improvement (Section III.E).

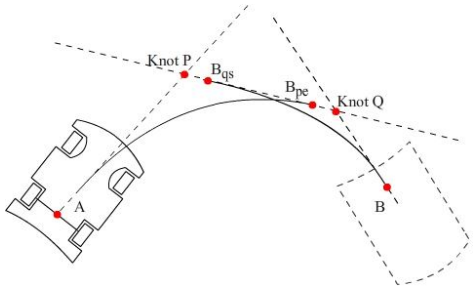

Figure 6. Two maneuver patterns may overlap.

C. *Sub-trajectory update*

As the robot moves, its range of sensing changes, like a sliding window shown in Figure 7. In each new sensing cycle, the newly visible knot points are added to the end of the subsequence *s* of visible knot points. Accordingly, each sub-trajectory in *P* is also updated to include additional trajectory segments generated from the newly added knot points in *s* in ways as described in steps 1), 3), 4), and 5) in the last subsection.

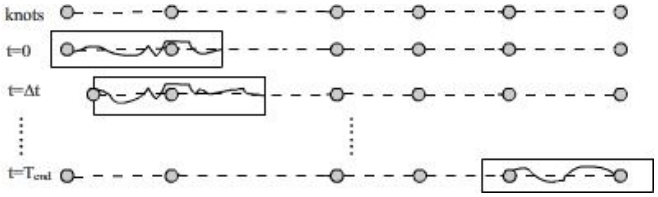

Figure 7. Updating sub-trajectories in a "sliding window" fashion

As introduced in Section III.A, the starting point of *s* and the starting configuration, velocity, and acceleration of each sub-trajectory in *P* should be updated to those of the robot (as it moves) at the end of each adaptation cycle, based on which, the first segment of each sub-trajectory is updated. All the knot points of *s* already passed by the robot are dropped. All the segments executed by the robot are dropped from the sub-trajectory that the robot follows.

D. *Sub-trajectory fitness evaluation*

A sub-trajectory is *feasible* if it is collision-free and satisfies the nonholonomic constraint (4) for every point on the trajectory. We use the time duration of a feasible trajectory to measure its fitness: the shorter duration, the better is the trajectory.

For an infeasible sub-trajectory, its fitness is evaluated as the sum of the collision cost for each trajectory segment colliding with an obstacle as explained below**.** Here we assume that a moving obstacle is slow enough that it can be viewed static during one sensing cycle.

We view each obstacle as a conical hill with a single peak located at approximately the geometrical center of its convex hull (see Figure 8). Each circle of the hill is assigned a gravity potential value: the closer the circle is to the center, the higher is its potential value. If a trajectory segment collides with an obstacle, the gravity potential of the inner most circle that it intersects is used as the collision cost of the segment. Thus, a trajectory segment that deeply penetrates into an obstacle is penalized more severely in evaluation. Unlike traditional exploring methods that completely rely on randomness to fix a collision, our use of a potential value as a collision cost can help the planner to create a collision-free trajectory more efficiently. If an obstacle is only partially known, the potential values can be built just based on the known part. Even if the known part of an obstacle is roughly a line or curve, we can still define potentials for discrete points on it.

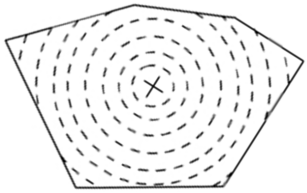

Figure 8. The gravity potential field inside an obstacle

Once a trajectory segment is modified (to be described below), the collision cost of only that segment needs to be re-computed, and evaluation can be very efficient.

E. *Sub-trajectory modification*

We use four operators **add**, **delete**, **adjust-curve** and **disturb** to improve and diversify (see Algorithm 2) the shapes of nonholonomic trajectories in real-time.

Add/delete adds/deletes a randomly chosen intermediate control point of a sub-trajectory to change it.

Adjust-curve examines each Bezier curve segment of a trajectory and tries to improve it in the following way. If the segment does not satisfy the nonholonomic constraint, no matter whether it is collision-free, the operator pushes the initial and end control points of the Bezier curve segment away from the middle control point a random distance within a small neighborhood to make the curve flatter. If the segment is already feasible, it randomly pulls the two points toward the middle control point a random distance to make the curve smaller in order to save time. The intention of the adjust-curve operator is to lower the cost value of a trajectory to improve it, as shown in Figure 9.

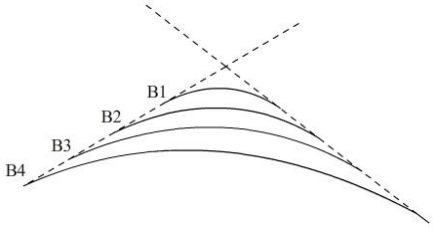

Figure 9. Operator **adjust-curve** extends and shrinks a Bezier curve.

The operator **disturb** is used only on a sub-trajectory that is not collision-free in order to improve it. For each linear or curve segment that is not collision free, **disturb** randomly varies the coordinates of a related knot/control point within a small neighborhood to hopefully lower the gravity potential value associated with the segment. As the planning process

favors the improved sub-trajectories with lower costs, repeated use of this operator over planning cycles often quickly results in feasible sub-trajectories.

Note that **adjust_curve** and **disturb** are local modification operators, whereas **add** and **delete** are global modification operators. Together, they achieve a good balance to produce efficient and varied sub-trajectories in *P*.

IV. Implementation and Experimental Results

Our planner is implemented in C++ and runs on an Intel core2 duo (3.0 G*Hz*) machine with Windows Operation System. We tested the planner in a simulation environment. The robot takes the shape of a 4.6×3.1 m^2 rectangular box. The distance between the robot's front and back axles is 2.7 *m*, and the maximum turning angle φ_{max} of the front wheels is 45°. The speed limit is 10m/s for both backward and forward movements, and the acceleration limits are ± 4m/s^2.

We assume the robot can sense obstacles within its neighborhood of radius 44.5m with a frequency of 10 *Hz*.

In the shown experiments, the size of trajectory set *P* maintained by the planner was 10 (i.e., 10 trajectories were in the set). Our planner uses an adaptation rate of 10*Hz*.

Figure 10 shows the snapshots of the operation of a vehicle in one task environment, guided by our ON planner. The vehicle is shown as a box with an arrow inside pointing to its front. Its initial and final configurations are indicated in red and black respectively. The blue straight line segments connect the given knot points about the path (assumed from some GPS navigator input), which even run through an obstacle. This is because the GPS navigator input only indicates a series of rough knot positions in the environment without other information of geometry. At each moment, the (parts of) obstacles within the sensing range of the robot are outlined in red with gray stripes, and the (parts of) obstacles that the robot cannot sense are outlined in gray. The best feasible nonholonomic sub-trajectory generated within the sensing range is shown in red (note that only the spatial curve is shown). As the robot moved along a sub-trajectory, the sensing range changes, and more obstacles were discovered, shown in Figure10 (b)(c)(d). The planner took into account the new obstacles in planning and constantly sought new feasible and better sub-trajectories to enable the robot to switch to. The accumulative effect of such real-time planning, improvement, and adaptation results in the smoothly executed, near optimal trajectory.

Note that there was an obstacle (e.g., another vehicle) moving towards right from top left during the period. The robot was not able to sense this obstacle until *after* the time of the 3rd snapshot Figure 10(c). However, once the robot sensed this obstacle, it planned its trajectory quickly to avoid the obstacle, as shown in Figure10 (d). This illustrates that the ON planner can deal with unknown changes in the environment effectively. In this run, the time was less than 1 second between when the moving obstacle was discovered to block the current best sub-trajectory and when a new feasible sub-trajectory was found each time we run (and re-run) the task.

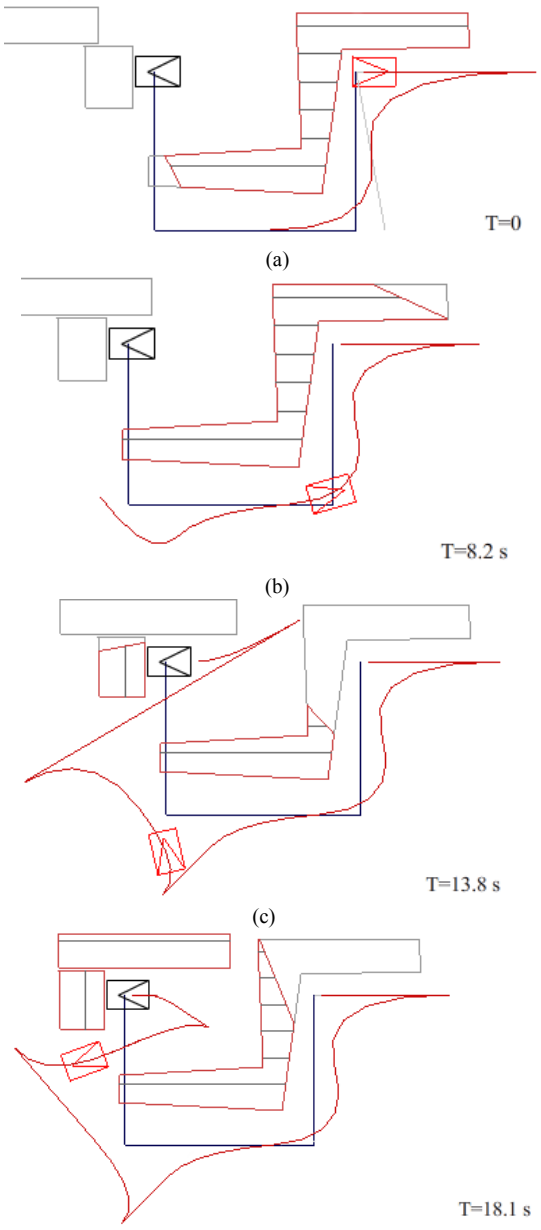

Figure 10. Snapshots of on-line navigation (example 1)

Figure 11 shows the snapshot of another example, when the goal configuration of the vehicle (at the right side of the figure) was reached. The executed path is the red curve, and the blue straight-line segments connect the input knot positions. Note that in both examples, the GPS inputs were quite inaccurate. It shows that our ON planner can be effective with very limited path information, which does not have to be from the GPS (when it is not available) and could be from any rough knowledge source of the environment, or the knot points could be decided on the fly by our planner.

Table 1 shows the running statistics for both example tasks, averaged over 5 times. Table 1(a) shows the average

length and elapsed time of the executed trajectory as well as the average speed (over both forward and backward segments). Note that the average speed is about 10.5 miles/hr and about 12.6 miles/hr respectively in the two examples, which are quite fast (under the 10m/s speed limit and ±4 m/s^2 acceleration limits) for the kind of maneuvering required.

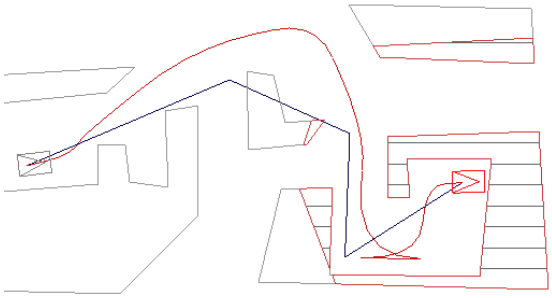

Figure 11. A snapshot when the goal configuration is reached (example 2)

Table 1(b) shows the average total number of planning cycles, planning frequency, and the average number of planning cycles per adaptation cycle (whose frequency is fixed at 10 *Hz*). Clearly our on-line planner is very efficient.

Table 1(c) shows the average numbers of forward and backward segments and the average total time durations of forward and backward driving. The sum of the numbers of forward and backward segments gives the total number of linear and Bezier segments in a trajectory.

Note that the smooth, executed trajectories for both examples consist of many segments. The trajectory of example 1 consists of more backward segments than forward segments on average, and the smooth path shown in Figure 11 for the example 2 actually consists of 16 segments, with one backward segment, which is the bottom straight line segment.

(a)

Example	Length (*m*)	Time (*s*)	Avg. speed (*m/s*)
1	119.81	25.2	4.7
2	114.98	20.4	5.64

(b)

Example	# Plan cycle	Planning freq. (*Hz*)	# Plan cycle per adaptation cycle
1	13839	549	54.7
2	11546	566	56.6

(c)

Example	#Forward segments	#Backward segments	$T_f(s)$	$T_b(s)$
1	5	7	10.5	14.7
2	15	1	16.8	3.6

Table 1. Statistics of examples 1 and 2

V. CONCLUSION

We introduced an efficient and effective on-line planner to enable a nonholonomic vehicle to simultaneously plan and execute nonholonomic and collision-free trajectories in unknown and crowded environments based on sensing towards its goal. A set of basic maneuver patterns using Bezier curves allow quick construction of nonholonomic trajectories with great flexibility. The robot can move in both forward and backward fashions to maximize efficiency. Future work includes extending our planner to consider realistic sensing scenarios and sensing uncertainty and to handle more complex nonholonomic robot models and constraints, including constraints on vehicle dynamics. We also plan to test the algorithm on certain real mobile robot system in real environments.

REFERENCES

[1] G. Lafferriere, H.J. Sussmann, "A differential geometric approach to motion planning," *Nonholonomic Motion Planning*, Z. Li and J.F. Canny *Eds*, Kluwer Int. Series Engin. & Computer Science 192, 1992.

[2] D. Tilbury, R. Murray, S. Sastry, "Trajectory generation for the n-trailer problem using Goursat normal form," *IEEE Trans. on Automatic Control*, Vol. 40 (5), pp. 802-819, 1995.

[3] P. Rouchon, M. Fliess, J. Levine and P. Martin, "Flatness and motion planning: the car with n trailers," *European Control Conference*. pp. 1518-1522, 1993.

[4] C. Fernandes, L. Gurvits, Z.X. Li, "A variational approach to optimal nonholonomic motion planning," *IEEE Int. Conf. on Robotics and Automation*, pp. 680-685, Sacramento, 1991.

[5] J.P. Laumond, S. Sekhavat, F. Lamiraux, *Lectures Notes in Control & Info. Sciences 229*. Springer, pp. 13, 1998.

[6] S. M. LaValle, J.J. Kuffner, "Rapidly-exploring random trees: Progress and prospects," in *Algorithmic and Computational Robotics: New Directions*, B. R. Donald, K. M. Lynch & D. Rus, *Eds.* K. Peters, pp. 293-308, 2001.

[7] R. Pepy, A. Lambert, "Safe Path Planning in an Uncertain-Configuration Space using RRT," *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pp. 5376-5381, 2006.

[8] K. Bekris, L. Kavraki, "Greedy but Safe Replanning under Kinodynamic Constraints," *IEEE ICRA*, pp. 704-710, 2007.

[9] Y. Kuwta, G. Fiore, J. Teo, E. Frazzoli, "Motion Planning for Urban Driving using RRT," *IEEE IROS*, pp. 1681-1686, 2008.

[10] W. Cheng, Z. Tang, "Path Planning for Nonholonomic Car-like Mobile Robots Using Genetic Algorithms," *The 8th Int. Conf. on Signal Processing*, 4:16-20, 2006.

[11] X. Ge, H. Li, "Nonholonomic Motion Planning of Space Robotics Based on the Genetic Algorithm with Wavelet Approximation," *IEEE Int. Conf. on Control and Automation*, pp.1977 -1980, 2007.

[12] G. Erinc, S. Carpin, "A genetic algorithm for nonholonomic motion planning," *IEEE Int. Conf. on Robotics and Automation (ICRA)*, pp. 1843-1849, 2007.

[13] S. Sekhavat, P. Svestka, J.P. Laumond, M. Overmars, "Multi-Level Path Planning for Nonholonomic Robots using Semi-Holonomic Subsystems," *Workshop on Algorith. Foundations of Robotics*, 1996.

[14] L.E. Dubins. "On curves of minimal length with a constraint on average curvature and with prescribed initial and terminal positions and tangents,"*American J. Math*., 79:497–516, 1957.

[15] J. A. Reeds, L. A. Shepp, "Optimal paths for a car that goes both forward and backward," *Pacific J. Math*., 145(2):367–393,1990.

[16] D.Shin, S. Singh, "Path Generation for Robot Vehicles Using Composite Clothoid Segments," Technical Report CMU-RI-TR-90-31, Carnegie Mellon University, 1990.

[17] J. Vannoy, J. Xiao, "Real-time Motion Planning of Multiple Mobile Manipulators with a Common Task Objective in Shared Work Environments," *IEEE ICRA*, Rome, Italy, pp. 20-26, April 2007.

[18] J. Vannoy, J. Xiao, "Real-time Adaptive Motion Planning (RAMP) of Mobile Manipulators in Dynamic Environments with Unforeseen Changes," *IEEE Transactions on Robotics*, 24(5):1199-1212, Oct. 2008.