A Fast PRM Planner for Car-like Vehicles

Authors: Hyung Ju Suh, James Deacon, Qifan Wang

***Abstract*— We present a probabilistic roadmap method (PRM) for Ackermann vehicles by exploring the non-holonomic constraint of the Ackermann linakge. Similar to most PRM methods, the configuration space is first sampled in order to guarantee no-collision of the vehicle. Then, the sampled states are connected using proximity and our own algorithm for modeling the non-holonomic properties of the Ackermann. Finally, a path-searching algorithm is applied to find the shortest path from the start to goal state. We show that the path planner is successful in finding the shortest path while incorporating the non-holonomic constraints, and discuss some important computational properties of this method.**

I. Introduction

Many robotic path-planning algorithms assume a holonomic or a differential vehicle, where the robot can move in all three degrees of freedom in $SE(2)$. (Or in the case of differential vehicle, approximate the full degree of freedom by turning in place) However, often times ground-based vehicles use steering schemes that make the vehicle non-holonomic. A popular example is a car-like vehicle that only has two degrees of freedom in throttle and steering. This condition puts additional constraints on the robot's kinematics, which makes it difficult to devise a general non-holonomic path planner that is able to satisfy every steering scheme.

Among popular steering schemes of a car-like vehicle, we focus on the Ackermann steering model, which is shown to approximate the steering curve nicely in low speeds and zero slip-angles [1]. We also show that the proposed method can be extended to other steering methods such as bicycle, rack-pinion, or Anti-Ackermann steering, as long as they satisfy the criteria that the vehicle's change in orientation can be fully recovered once its change in position is known within a certain resolution.

Popular methods behind holonomic path planning of robotic vehicles have largely based themselves on the Probabilistic Roadmap (PRM) method [2], which views path planning in a graph generation and searching context. Many variations of the PRM exist that takes into different sampling methods, connection methods, and collision-checking methods [3]. However, non-holonomic properties of certain vehicles are difficult to incorporate into the PRM framework.

Some alternatives to non-holonomic path planning are based on motion primitives. In Pivtoraiko and Kelly [4,5], the non-holonomic motion planning is done by dividing the space into different lattices and utilizing motion primitives to connect the movement from one lattice to another. While this method is efficient and tractable, the Euclidean lattice doesn't fully allow to consider the shapes of various obstacles. Additionally, utilizing motion primitives confines the allowed motion into a set of certain motions, which blocks some paths from being fully utilized for optimal path planning.

Other approaches utilizes other shape representations instead of curves and lines to model the paths of the Ackermann vehicle. The most representative work is Dubin's path [6], which models the path as a series of circles and lines tangent to the circles. Reeds and Sheep [7] explore a variation where the vehicle can go backwards. Other approaches utilize sinusodial curves [8], or Bezier curves and splines [9,10] to approximate an Ackermann path. These methods have been most successful in describing the path of Ackermann vehicles. However, the math behind various curves and shape representations hasn't successfully been integrated into sample-based motion planning, as few of these works deal with the no-collision constraint, and the solutions are not proven to be tractable.

In this work we propose a simple revision to the PRM method by dealing with non-holonomic motion constraint as a graph-connection problem between states. By retaining the PRM method, we argue that we can keep some nice properties of the PRM, such as its completeness, tractability, and probabilistic convergence. The rest of the paper is arranged as follows: Section II. will first describe the geometry of the Ackermann steering curve. Section III.A will describe the C-space sampling procedure, as well as our definitions for collision detection. This corresponds to building a vertex set of our graph. Section III.B will deal with generating the edges of the graph by applying the proximity condition and the Ackermann model. Finally, the results of these theories are simulated and shown in Section IV. In Section V. we will discuss some interesting properties and details of our method.

II. Non-holonomic Ackermann Model

An Ackermann vehicle only has 2 degrees of freedom for steering and throttle. However, it operates on a 3 dimensional space in $SE(2)$. Therefore the vehicle is inherently not fully controllable, and thus considered non-holonomic. Due to these inherent limitations in controllability, path planning for non-holonomic vehicles need to consider the manner of this loss to avoid paths that the vehicle cannot follow.

Although it is difficult to fit all the non-holonomic characteristics of various vehicles into a single framework, we attempt to achieve a generic representation of car-like vehicles (fixed rear wheel and front wheel steering) by representing the steering curve as a family of functions:

$$y = S(\delta, x) = \{s(x) | s(x) \text{ is a steering geometry with steering angle parameter of } \delta\}$$

which could be tuned differently for different steering linkages, such as rack & pinion, anti-Ackermann, bicycle, four-wheel steering, and etc. The Ackermann linkage will specifically be used for this work. Additionally, we achieve a functional representation of steering by setting the resolution of the path planner to be within the limits of a bijective function relation. For instance, we approximate a circle by an arc by limiting the region of interest.

In order to model the steering geometry for the path planner, we utilize the Ackermann model described in Jazar's work [1].This is illustrated in Figure 1.

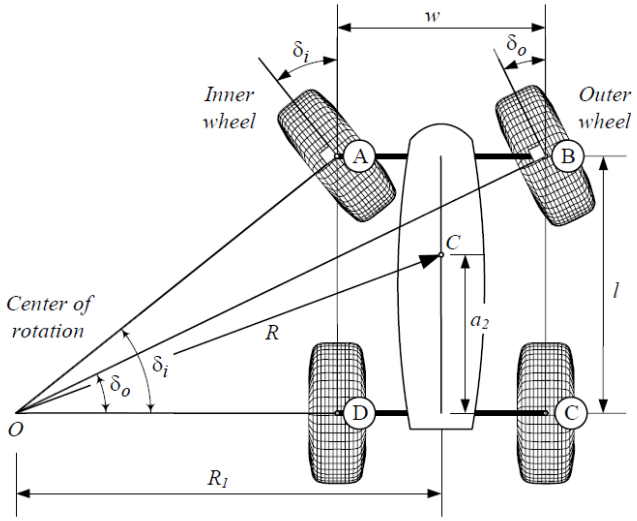

Fig. 1. Model of Ackermann Geometry, from [1]

The radius of the vehicle can be obtained from the vehicle geometry and steering geometries

$$R(\delta) = \sqrt{a_2^2 + l^2 \cot^2 \delta}$$

where δ is the cotangent average of δ_0 and δ_i. The steering center O is defined as an intersection between a circle of R from the vehicle center C, and the horizontal line drawn by connecting the back wheels. In our framework, we consider the steering curve from the vehicle's body frame as illustrated in Figure 2.

As a function of δ, this steering curve can then be expressed as the following family of functions:

$$S(\delta, x) = \begin{cases} y = -\sqrt{R(\delta)^2 - (x+a_2)^2} + \sqrt{R(\delta)^2 - a_2^2} & \text{if } \delta > 0 \\ y = \sqrt{R(\delta)^2 - (x+a_2)^2} - \sqrt{R(\delta)^2 - a_2^2} & \text{if } \delta < 0 \\ y = 0 & \text{if } \delta = 0 \end{cases}$$

An important characteristic of the steering function $S(\delta, x)$ is that for a given Euclidean position (x, y), there only exists one allowed steering function $S(x)$ that allows the robot to reach the state. Thus this accurately describes the Ackermann geometry by connecting the angular state θ as a function of x, y. These functions are validated and plotted

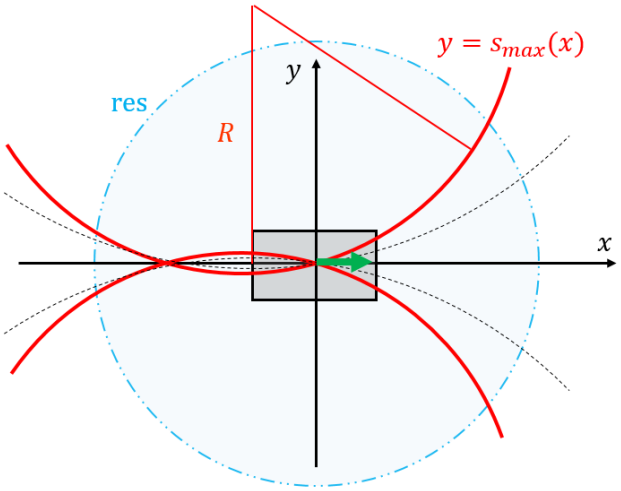

Fig. 2. Function representation of Steering Model

for the geometries of a car vehicle. The example used is the Mercedes CLA250. The maximum steering angle δ is decided based on its curb-to-curb steering diameter. This is illustrated in Figure 3.

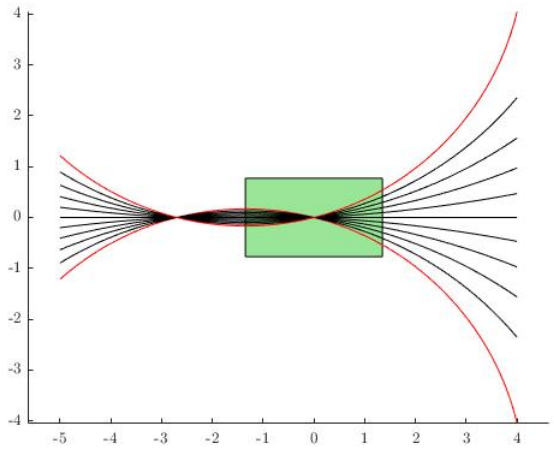

Fig. 3. Plot of steering functions from δ_{max} to $-\delta_{max}$

Finally, in order to correctly define the domain in which $S(\delta, x)$ is valid as a function representation, we present a relation that resolution has with the wheelbase and the maximum steering angle. This can be achieved with observing when the slope of $S(\delta, x)$ blows up, and is described by the equation

$$res \leq \sqrt{R(\delta_{max})^2 + (R(\delta_{max}) - a_2)^2}$$

To give a sense of the scale of this resolution, for Mercedes CLA250 with a wheelbase of $2.7m$ and maximum steering angle of $\delta = 26.85°$, the resolution of the planner must be smaller than $6.8925m$.

III. Motion Planning Method

We view motion planning in a graph-search framework, taking an approach in a Probabilistic Road-Map (PRM) framework. PRM approaches usually take the following steps in order to generate a feasible path from one state to another:

1) Sample the C-space and collect collision-free vertices
2) Iterate through vertices and connect them based on proximity
3) Search graph for the shortest path from start to goal

While we repeat the first and last steps of the PRM method, we address the non-holnomic constraints of the Ackermann vehicle by incorporating the constraint in edge connection. Apart from the proximity condition defined by the resolution of the path planner, we impose additional constraints as described in Section III.B.

A. Vertex Generation: C-Space Sampling

In this section we illustrate the method used for sampling the C-space, and how collision-detection is achieved. The high-level overview of the algorithm is to sample states $\vec{X} = (x, y, \theta)$, and only include states that do not collide with obstacles. Obstacle and robot representation can either be done continuously (analytically), or by discretizing the obstacle and the robot. In our approach, the robot is discretized while the obstacles are encapsulated by a combination of booleans.

Robot discretization is done by dividing the robot geometry into a set of points $R(x,y)$. We choose this discretization to a regular lattice spacing. On the other hand, the obstacles will be represented as a collection of booleans such that for every line segment on the boundary, we define a linear inequality. This way we can check for every discretized point on the robot if the robot meets the boolean condition of not being within the obstacle, and achieve collision detection. This idea is illustrated in Figure 4.

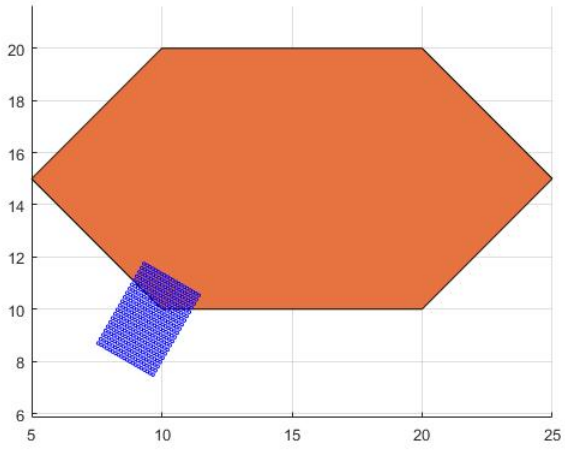

Fig. 4. Illustration of collision between discretized robot and boolean obstacle

Note that in order to apply a boolean condition, the obstacles must be a convex shape. To extend this idea to concave shapes, we can make use of the fact that every concave vehicle can be broken down into series of convex shapes [11]. Thus the algorithm for C-space sampling is illustrated as follows:

Algorithm 1 C-space Sampling Algorithm

Given number of sample points N
Given map dimensions $[map_{-x}, map_x, map_{-y}, map_y]$
Initialize $V = \emptyset$
while True **do**
  Choose x_r randomly from $[map_{-x}, map_x]$
  Choose y_r randomly from $[map_{-y}, map_y]$
  Choose θ_r randomly from $[0, 2\pi]$
  for all $[x,y]^T \in R(x,y)$ **do** ▷ Robot Discretization
    $\begin{bmatrix} x' \\ y' \end{bmatrix} = R(\theta_r) \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} x_r \\ y_r \end{bmatrix}$ ▷ Coordinate Transform
    if $[x', y']^T$ is not in obstacle **then**
      $\vec{X} = [x, y, \theta] \in V$
    end if
  end for
  if $|V| = N$ **then**
    break
  end if
end while

B. Edge Generation: Non-holonomic Model

In traditional edge generation in graph-based methods for holonomic vehicles, the edges are usually connected based on a proximity function based on the L2 (Euclidean) norm. The algorithm loops through all the vertices, looks at close points, and generates the edge. This traditional algorithm is illustrated in Algorithm 2.

Algorithm 2 Holonomic Edge Generation

Given $V = \{\vec{X} | \vec{X} = [x, y, \theta]^T\}$, *res*
Initialize $E = \emptyset$
for all $\vec{X}_i \in V$ **do**
  Initialize $P_i = \emptyset$
  for all $\vec{X}_j \neq \vec{X}_i \in V$ **do** ▷ Proximity Search
    if $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \leq res$ **then**
      $\vec{X}_j \in P_i$
    end if
  end for
  for all $\vec{X}_j \in P_i$ **do**
    if $(\vec{X}_i, \vec{X}_j) \notin E$ **then** $(\vec{X}_i, \vec{X}_j) \in E$
    end if
  end for
end for

We revise this algorithm by imposing a non-holonomic constraint on the vehicle by relating θ to x and y. The high-level idea of the non-holonomic constraint algorithm is given as follows:

1) Observe near configurations in the body frame of the robot's state

2) Based on other configuration's Euclidean coordinates (x,y), calculate its allowed angle θ using the non-holonoic model
3) If the sampled state's angle θ is within the tolerance value of the calculated value, then connect the graph

This can be expressed mathematically as follows. Given two configurations $\vec{X}_i = [x_i, y_i, \theta_i]^T$ and $\vec{X}_f = [x_f, y_f \theta_f]^T$, we can use coordinate transform to view $\vec{X}_f$ from the body frame of $\vec{X}_i$, and denote this state $\vec{X}' = [x', y', \theta']$. This is a simple transform using planar displacements,

$$\vec{X}' = \begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} R(-\theta_i)\left(\begin{bmatrix} x_f \\ y_f \end{bmatrix} - \begin{bmatrix} x_i \\ y_i \end{bmatrix} \right) \\ \theta_f - \theta_i \end{bmatrix}$$

After this, we can proceed to observe $\vec{X}_f$ in the body frame of $\vec{X}_i$. This is illustrated in Figure 5.

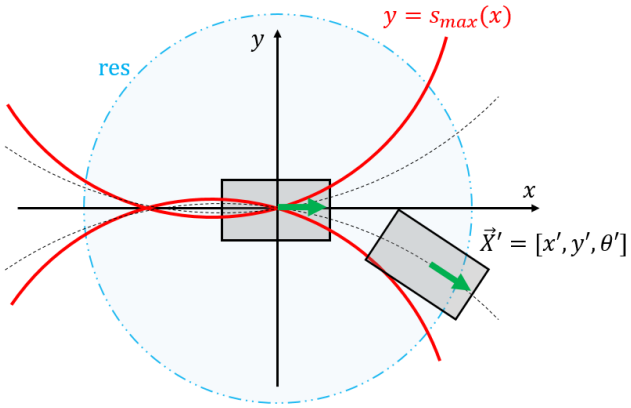

Fig. 5. Other states of robot viewed from body frame of current state

Next, we apply the steering function by calculating θ' from x' and y'. From the equations of Ackermann kinematics, we can express the radius of steering R as a function of x' and y'

$$R(x', y') = \sqrt{\left(\frac{x'^2 + 2a_2 x' + y'^2}{2y'} \right)^2 + a_2^2}$$

We can immediately put a constraint based on this calculated radius, since the vehicle has a minimum turning radius based on δ_{max}. Calculating the radius allows us to recover the full steering curve $S(x)$ from x' and y'. We calculate the allowed angle value of taking a slope of this curve.

$$\theta_c(x', y') = \frac{dy}{dx} = \begin{cases} atan\left(\frac{x+a_2}{\sqrt{R(x',y')^2 - (x+a_2)^2}}\right) & \text{if } y > 0 \\ -atan\left(\frac{x+a_2}{\sqrt{R(x',y')^2 - (x+a_2)^2}}\right) & \text{if } y < 0 \\ 0 & \text{if } y = 0 \end{cases}$$

Then, the final check is to compare this value with the sample value θ' to see if this is an allowable configuration. We can do this by defining a tolerance for the path planner, such that

$$|\theta' - \theta_c(x', y')| \leq tol$$

Finally, if the edge is connected, we can weight the edges by the Euclidean distance between the position of the two states so that the path searching algorithm can take the shortest path. The overview of the final edge-generation algorithm is illustrated in Algorithm 3.

Algorithm 3 Non-Holonomic Edge Generation

Given $V = \{\vec{X} | \vec{X} = [x, y, \theta]^T\}$, *res*, and *tol*,
Initialize $E = \emptyset$
for all $\vec{X}_i \in V$ **do**
  Initialize $P_i = \emptyset$
  for all $\vec{X}_j \neq \vec{X}_i \in V$ **do** ▷ Proximity Search
    if $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \leq res$ **then**
      $\vec{X}_j \in P_i$
    end if
  end for
  for all $\vec{X}_j \in P_i$ **do** ▷ Non-Holonomic Constraint
    Compute $\vec{X}'$ by coordinate transformation
    Compute $\theta_c(x', y')$ from $\vec{X}'$
    if $|\theta_c(x', y') - \theta'| < tol$ **and** $R(x', y') < R_{max}$ **then**
      $(\vec{X}_i, \vec{X}_j) \in E$
      $w(\vec{X}_i, \vec{X}_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$
    end if
  end for
end for

C. Shortest Path Search Algorithm

By III.A and III.B, we have generated a complete graph $G = (V, E)$ with all the desired vertices and edges. Therefore, what is left to do is to search the graph from our desired start configuration $\vec{X}_i$, to the final configuration $\vec{X}_f$. Among possible shortest-path searching algorithms, we choose Dijkstra's algorithm[12] for the graph search process because of its simplicity. One alternative is the A* algorithm [13], which works on the same principle with Dijkstra's algorithm with an added heuristic function and a priority queue. The A* will outperform Dijkstra's algorithm in most cases.

However, we argue that the usual heuristic function used in A*, which is the Euclidean distance between positions of $\vec{X}_i$ and $\vec{X}_f$, may not be effective as it is in holonomic motion planning. This is because non-holonomic motion planning doesn't necessarily take the Euclidean distance as a good heuristic. In this regard, even inexperienced humans have a hard time reaching from one state to another when driving a car in confined space (such as parking) which tells us that designing a good heuristic function for non-holonomic vehicles is a challenging task.

Dijkstra's algorithm has been extensively described in various sources, but we describe the pseudocode in algorithm 4 for a review.

Algorithm 4 Dijkstra's Algorithm for Shortest Path Search

Given Graph $G = (V, E)$ and weights $w(E)$
Given source v_s and target v_t
Initialize $Q = \emptyset$
for $v \in V$ **do**
    $dist(v_s, v) = \infty$
    $prev(v) =$ undefined
    $v \in Q$
end for
$dist(s, s) = 0$
while $Q \neq \emptyset$ **do**
    $u =$ vertex in Q with minimum dist
    $u \notin Q$
    for v as neighbor of Q **do**
        if $dist(v_s, u) + w(u, v) < dist(v_S, v)$ **then**
            $dist(v_s, v) = dist(v_s, u) + w(u, v)$
            $prev(v) = u$
        end if
    end for
end while

IV. Results

In this section, we present the results of the theory of Sec.III, tested in a simulation environment. We will first present the validity of C-space sampling and our collision detection algorithm. Then we will test the holonomic motion planner to illustrate the results of the PRM method. Finally, we present the results of our non-holonomic motion planner.

A. C-space Sampling

First we present the result of sampling the C-space according to Algorithm 1. The sampling result with various convex shapes is illustrated in Figure 6. In addition to the algorithm above, we also augment the car boundaries by 1 meter in order to ensure no collisions in between the points.

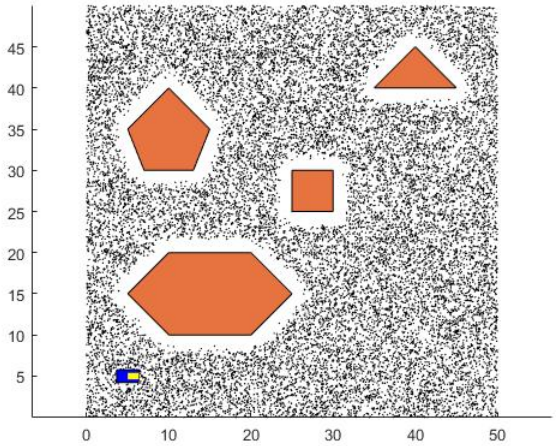

Fig. 6. Sampling result on various convex objects. Block dots represent the x, y coordinates of the samples, and the orange polygons represent the obstacles. Car size (blue) illustrated for scale.

We can observe that the sampling result accurately simulates the C-obstacles once the sample is dense enough. In Figure 6, the number of points were done as $N = 20000$. In Figure 7, we illustrate the fact that a concave object can be represented by a series of convex objects in modeling the C-space.

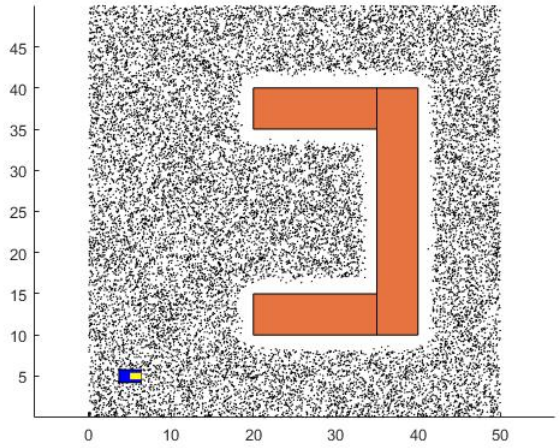

Fig. 7. Sampling result on a concave object, as a combination of convex objects. Block dots represent the x, y coordinates of the samples, and the orange polygons represent the obstacles. Car size (blue) illustrated for scale.

Through Figure 6 and 7, we have shown that our method for collision-detection is good enough to handle obstacles of various shapes, regardless of its convexity.

B. Holonomic Path Planner

In this section, we present the result of the holonomic motion planner using Algorithm 2, which is the traditional PRM method. For the map in Figure 6, the result of path planning is illustrated in Figure 8

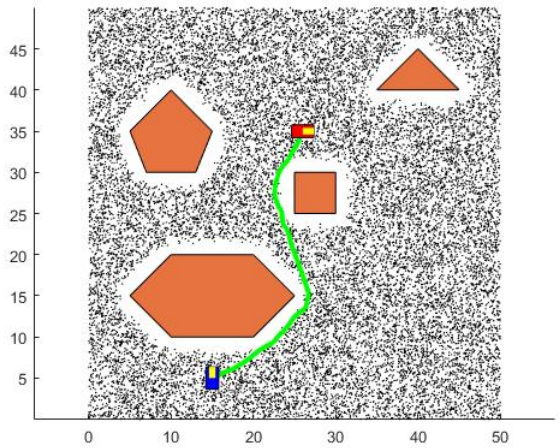

Fig. 8. Result of the holonomic path planner. The vehicle starts from blue to red, and the path is drawn in green

We show another example of the holonomic path planner in the concave obstacle of Figure 9.

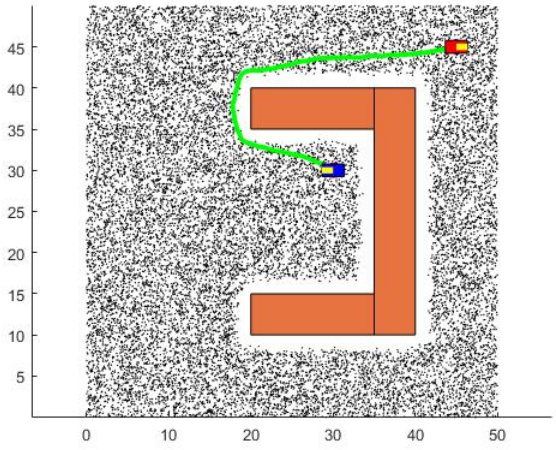

Fig. 9. Result of the holonomic path planner. The vehicle starts from blue to red, and the path is drawn in green

In both executions, the PRM method is successful in finding a shortest path assuming that the vehicle is holonomic.

C. Non-Holonomic Path Planner

In this section, we finally present the result of our own algorithm on non-holonomic path planning.

1) Without Obstacles: Results of the non-holonomic path planning without obstacles is quite meaningful because it allows us to observe how one state is reached from another purely using the vehicle's Ackermann kinematics. Figure 10 illustrates one case of this algorithm.

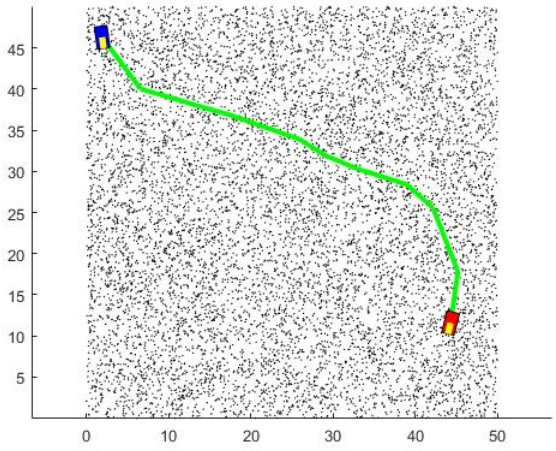

Fig. 10. Result of the non-holonomic path planner that emphasizes smooth turning capabilities. The vehicle starts from blue to red, and the path is drawn in green

We can see that unlike the holonomic motion planners, which will likely rotate, draw a straight line, and rotate, a non-holonomic motion planner approximates the shortest path by a smooth path that is more friendly to its kinematics. Next, Figure 11 illustrates the ability of the path planner to maneuver with point turns. The use of graph-searching easily allows it to go backwards with a different curvature, which allows the presence of cusps in point-turning maneuvers that approximate the kinematic shortest path better. Often times in free space, the solution to point-turning is found by the intersection of two steering circles that form a cusp. This is also known as the Reeds-Shepp curve [7].

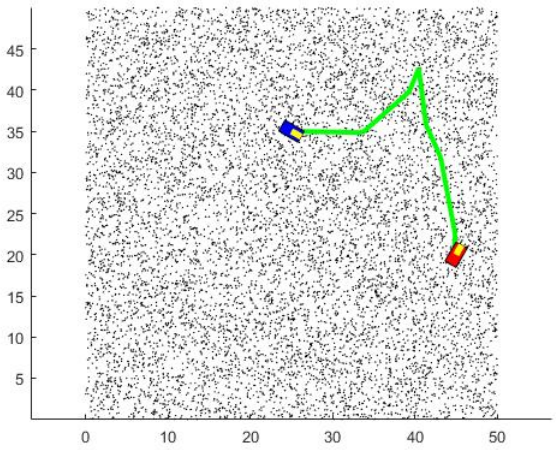

Fig. 11. Result of the non-holonomic path planner that emphasizes point-turning capability. The vehicle starts from blue to red, and the path is drawn in green

Finally, Figure 12 shows two configurations that are close to each other, in which multiple point turns must be made. The figure illustrates a 3-point turn that the vehicle takes in order to get from start to goal.

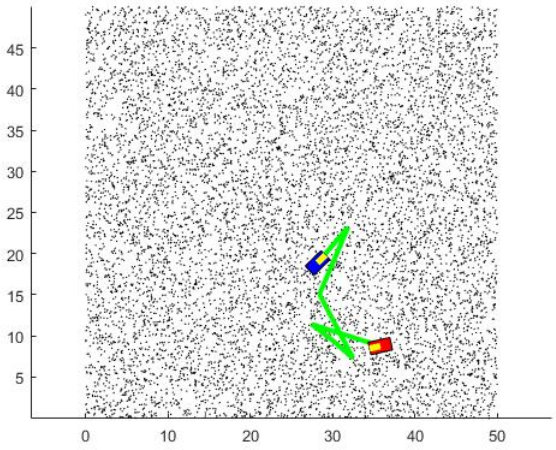

Fig. 12. Result of the non-holonomic path planner. The vehicle starts from blue to red, and the path is drawn in green

2) With Obstacles: In this section we put some plausible environments as obstacles to test the non-holonomic planner with obstacle spaces. Specifically, we show two cases of parking in which the vehicle must make point turns and

maneuvers to get to the goal. Figure 13 illustrates typical backwards parking start and goal configurations.

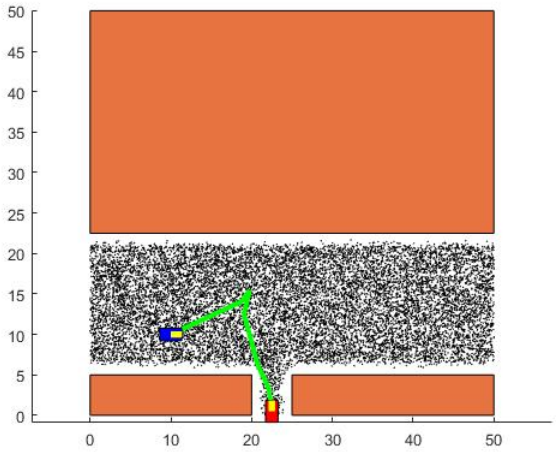

Fig. 13. Result of the non-holonomic path planner with obstacles, with a backwards-parking scenario. The vehicle starts from blue to red, and the path is drawn in green. Yellow marks the forward orientation of the vehicle

We can observe that the path nicely represents what a human would do in order to park the car backwards, and successfully finds a path from the start to goal configuration without any collisions. Additionally, in order to address the parallel-parking problem mentioned in [10], we simulate the start and goal configurations for parallel parking as well. The close-up of the resulting path is illustrated in Figure 14

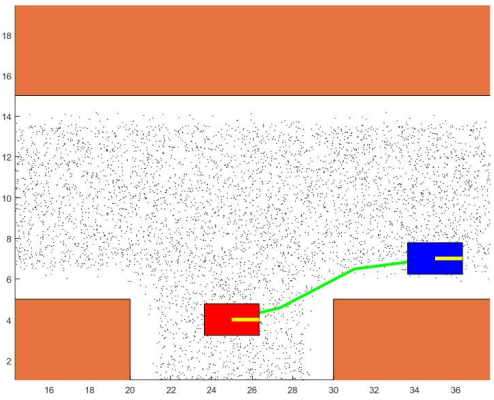

Fig. 14. Result of the non-holonomic path planner with obstacles, with a parallel parking scenario. The vehicle starts from blue to red, and the path is drawn in green. Yellow marks the forward orientation of the vehicle

We can see that the algorithm approximates the parallel-parking curve quite well, and will probabilistically converge to the shortest path with more densely-sampled points given the properties of the PRM method.

V. Further Work & Discussion

In this section we comment on certain aspects of our path planner, that has to do with resolution, sampling, tractability, and comparison with other methods.

A. Resolution

One of the major shortcomings of this method is that the points need to be sampled sufficiently dense within a given space. This deals with the fact that the condition for recovering the steering curve loses functional representation when the points are sampled more sparsely than the resolution criteria. However, this can be solved with a more careful formulation of the steering curve if we choose to let go of the functional representation, and instead incorporate a full circle instead of the arc. These arcs are useful because we can fully recover the predicted change in orientation from the change in position, and in the case of circles, it is likely that multiple solutions exist and we will lose this property.

On the other hand, one could argue that this resolution is sufficiently dense. Using the Mercedes CLA250 as an example, we showed that the minimum resolution needed for the path planner is along three-fold bigger than the vehicle size. In order to reconstruct the motion finely with a sample-based method, a three-fold resolution is not too coarse of a resolution.

B. Random Sampling

We present a more interesting problem here. In the PRM framework with holonomic robots, we were allowed to randomly sample x, y and θ independent of each other, because the robot has all three degrees of freedom to move. However for this non-holonomic planner, there is an added condition that even if the positions of the samples are finely sampled, there needs to be sufficiently many states with allowable angles in order for the path planner to find a feasible path. This issue can be reformulated with the following problem: this method ends up wasting the positions of many sampled states because the angle state is not allowable from the current state, even if that position is reachable with a different orientation. Thus the method, as is, requires a very dense sampling that probabilistically guarantees that an allowable state with the right position and the right angles always exists.

Thus this method can be improved with a more careful sampling method that guarantees less position states and more angular states. One option is to sample the (x, y) space, and augment each state with a regular-spaced angles. This assures that for each sampled positions, we have sufficiently many angles to make the position allowable. This brings the proposed method closer to the lattices used in motion primitive methods, and will require further discussion in completeness.

C. Tractability

We consider our algorithm to be tractable since it does not add much more asymptotic time complexity to the PRM method. The added constraint to represent non-holonomic motion does not do any additional search compared to the original algorithm, (this can easily be seen by comparing Algorithm 2 and Algorithm 3), and always computes a fixed

amount based on the nearby points it is searching. Therefore our algorithm multiplies a constant time complexity $O(1)$ to the PRM method, and is asymptotically tractable at the same complexity as the PRM. Thus our algorithm can be considered probabilistically complete as well.

To give a sense of the node and time tradeoff, the non-holonomic simulation runs with $N = 10000$ nodes take around 5 to 10 minutes on a regular desktop computer, depending on the size of the resolution. (If resolution is low while the sample number is high, the graph will be much more connected, and this will require more time in graph searching). However, using the angle resolution augmentation mentioned in Section V.B, it is possible to sample much less points to bring it closer to real-time.

D. Comparison with other methods

There are four methods that can be mentioned and compared with the presented path planner, with different characteristics of the generated path: the general non-holonomic PRM, the motion primitive approach, the curve primitive approach, and the non-holonomic Rapidly-exploring Random Trees (RRT).

The general non-holonomic PRM approaches the constraint by solving boundary-value problems of the general control function of the vehicle, and then verifying collision later by taking a lazy-PRM approach [14,15]. This method is not too different from our proposed method, but our method is inherently more model-based and the equations and conditions are faster to evaluate compared to numerically searching for solutions, or solving implicit equations. However, utilizing the purely kinematic relations makes it difficult to incorporate the dynamics in our framework, and a revision is needed to extend our method to kinodynamic planning.

We argue that our generated path retains much more generality compared to curves generated by motion primitives, because we are able to discretize the space randomly, instead of having regular lattices. This method is much more adaptive to various obstacle shapes, and since motion primitives only allow certain paths to be taken between each lattices, our paths have much more freedom to explore the allowed space.

Curve primitive approaches that utilize sine functions or splines often analytically guarantee optimality, but are often costly to implement, especially in collision checking. Many of the new methods based on splines utilize optimization techniques [10,16], which is often hard to calculate in real-time. In terms of quality of the path, however, these show most accurate paths between the states.

Finally, the non-holonomic RRT method is also a good competitor to this method. While the RRT with non-holonomic constraints can be tractable, it can be hard and inefficient to generally deal with parking problems that have point-turns and cusps. The RRT is also not a probabilistically complete method and doesn't guarantee rate of convergence.

REFERENCES

[1] Jazar, R.N., "Vehicle Dynamics: Theory and Application", Springer, 1st ed. 2009, ISBN 978-0387742434

[2] Kavraki, L. E., Svestka, P., Latombe, J.-C., Overmars, M. H. (1996), "Probabilistic roadmaps for path planning in high-dimensional configuration spaces", IEEE Transactions on Robotics and Automation, 12 (4): 566580, doi:10.1109/70.508439.

[3] Geraerts, R., Overmars, M. H. (2002), "A comparative study of probabilistic roadmap planners", Proc. Workshop on the Algorithmic Foundations of Robotics (WAFR'02)

[4] Pivtoraiko, M., Kelly, A., Efficient constrained path planning via search in state lattices, in International Symposium on Artificial Intelligence, Robotics, and Automation in Space, 2005

[5] Pivtoraiko, M., Kelly, A., "Kinodynamic Motion Planning with State Lattice Motion Primitives", 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, San Francisco, Sep. 25-30, 2011

[6] Dubins, L.E., "On Curves of Minimal Length with a Constraint on AVerage Curvature, and with Prescribed Initial and Terminal Positions and Tangents", American Journal of Mathematics 79(3):497-516, July 1957

[7] Reeds, J.A., Shepp,L.A., "Optimal Paths for a Car that Goes Both Forwards and Backwards", Pacific Journal of Math, 145(1990), pp. 367-353

[8] Murray, R.M., Sastry, S.S., "Nonholonomic Motion Planning: Steering Using Sinusoids", IEEE Transactions on Automatic Control, Vol.38, No.5, May 1993

[9] Gloderer, M., Hertle, A., "Spline-Based Trajectory Optimization for Autonomous Vehicles with Ackermann Drive", 2010

[10] Zhao L., Zheng G., Li, J., "Automatic Parking Path Optimization Based on Bezier Curve Fitting", Proceedings of the IEEE International Coneference on Automation and Logistics, Zhengzhou, China, August 2012

[11] Chazelle, B., Dobkin, D.P., "Optimal Convex Decompositions", Computational Geometry, Elseiver, pp.63-133, 1985

[12] Dijkstra,E.W., "A note on two problems in connexion with graphs", Numerische Mathematik, 1: 269-271, 1959, doi:10.1007/BF01386390

[13] Hart, P.E., Nilsson, N.J., Raphale, B., "A Formal Basis for the Heuristic Determination of Minimum Cost PAths", IEEE Transactions on Systems Science and Cybernetics SSC4. 4(2):100-107, 1968, doi:10.1109/TSSC.1968.300136

[14] LaValle, S.M., "Planning Algorithms", Cambridge University Press, 2006

[15] Bohlin, R., Kavraki, L., "Path Planning Using Lazy PRM", IEEE International Conference on Robotics & Automation (ICRA), April 2000

[16] Chen, C., Rickert, M., Knoll, A., "Path Planning with Orientation-Aware Space Exploration Guided Heuristic Search for Autonomous Parking and Maneuvering", IEEE Intelligent Vehicles Symposium (IV), 2015