Continuous Curvature Path Planning for Autonomous Vehicle Maneuvers Using RRT*

Lan, X.; Di Cairano, S.
TR2015-085  July 2015

Abstract

This paper proposes a sampling based planning technique for planning maneuvering paths for semi-autonomous vehicles, where the autonomous driving system may be taking over the driver operation. We use Rapidly-exploring Random Tree Star (RRT*) and propose a two-stage sampling strategy and a particular cost function to adjust RRT* to semi-autonomous driving, where, besides the standard goals for autonomous driving such as collision avoidance and lane maintenance, the deviations from the estimated path planned by the driver are accounted for. We also propose an algorithm to remove the redundant waypoints of the path returned by RRT*, and, by applying a smoothing technique, our algorithm returns a G squared continuous path that is suitable for semi-autonomous vehicles. In order to deal with sudden changes in the environment, we apply a replanning procedure to enable our algorithm to rapidly react to the changes in a real-time manner, without full recomputation of the RRT* solution. Numerical simulations demonstrate the effectiveness of the proposed method.

2015 European Control Conference (ECC)
Continuous Curvature Path Planning for Semi-Autonomous Vehicle Maneuvers Using RRT*

Xiaodong Lan and Stefano Di Cairano

Abstract—This paper proposes a sampling based planning technique for planning maneuvering paths for semi-autonomous vehicles, where the autonomous driving system may be taking over the driver operation. We use Rapidly-exploring Random Tree Star (RRT*) and propose a two-stage sampling strategy and a particular cost function to adjust RRT* to semi-autonomous driving, where, besides the standard goals for autonomous driving such as collision avoidance and lane maintenance, the deviations from the estimated path planned by the driver are accounted for. We also propose an algorithm to remove the redundant waypoints of the path returned by RRT*, and, by applying a smoothing technique, our algorithm returns a $G^2$ continuous path that is suitable for semi-autonomous vehicles. In order to deal with sudden changes in the environment, we apply a replanning procedure to enable our algorithm to rapidly react to the changes in a real-time manner, without full recomputation of the RRT* solution. Numerical simulations demonstrate the effectiveness of the proposed method.

I. INTRODUCTION

In this paper we consider the problem of planning a continuous curvature path for semi-autonomous driving. As information technology and artificial intelligence develop rapidly, it is becoming possible to use computers to assist daily driving, even to make the driving process entirely autonomous. While eventually autonomous driving will be possible, it is reasonable to expect that for a certain period the human driver and the autonomous system may need to coexist and possibly share control of the vehicle. This may introduce challenges as the decision between the driver and the path planner may be different at some times, and a resolution between the two that keeps the vehicle and the driver safe needs to be obtained [1], [2]. For instance, when the semi-autonomous planning system takes over from the driver for objectives such as collision avoidance of lane keeping, besides achieving such standard planning goals, additional objectives such as minimizing the modifications of the estimated path planned by the driver may need to be accounted for. The interaction between the path planner and the driver is critical to maintain vehicle drivability under semi-autonomous operation and driver acceptance of the system. Limiting the difference between the planned path and a reference path may be of use also in the case of autonomous driving, where a reference path may be provided by higher level planners or navigation systems. When the vehicle moves along the reference path, it should also have the ability to deal with sudden changes in the conditions on the road, such as a crossing pedestrian, a passing vehicle, or changes in traffic lights.

In this work we adapt Rapidly-exploring Random Tree Star (RRT*)[3] and propose some smoothing techniques to get a $G^2$ continuous path for the maneuvering of semi-autonomous vehicles. Our algorithm also enables the vehicle to do rapid replanning when sudden changes occur in the environment. Hence, the semi-autonomous vehicle can achieve rapid and real-time reaction to various kinds of road conditions. The approach is based on incremental sampling-based path planning algorithms, also known as randomized path planners, particularly the Rapidly-exploring Random Tree (RRT) algorithm [4], and its recent variant with optimality guarantees, RRT* [3]. Both of them successively construct a tree to rapidly cover the whole environment. The difference is that when adding new vertices to the tree, RRT* compares the cost of different vertices and connect this new vertex to the lowest cost vertex inside the tree. Hence, RRT* is guaranteed to asymptotically approach the minimum cost feasible path almost surely, if one exists. There are some existing work using RRT or RRT* to plan trajectories for autonomous driving. For example, [5] uses closed-loop RRT to plan trajectories for autonomous vehicles in an urban environment. Also, [6] uses RRT* to plan time-optimal trajectories for vehicle maneuvers, and [7] uses RRT* to plan trajectories for autonomous high-speed driving. Other methods for path planning include model predictive control (MPC), which allows to plan trajectories while accounting for complex vehicle dynamics [8]. In [9] MPC is combined with motion primitives to design controls for agile maneuvering of ground vehicles. In [10] an algorithm based on MPC is proposed for real-time obstacle avoidance for ground vehicles. However, all planners above are developed explicitly for fully autonomous driving. Here, we develop our algorithm considering the semi-autonomous driving case, where it is of importance to account for the path smoothness, i.e., we aim at obtaining a $G^2$ continuous path$^1$ for semi-autonomous driving. In particular, here we consider the difference between reference (i.e., driver) and semi-autonomous path in terms of curvature, since the driver is particularly sensitive to lateral acceleration and yaw rate, which occur while the vehicle turns, and hence the difference in turning behavior (i.e., in path curvature) should be limited.

In this paper, we first propose some extensions to RRT* to adapt to path planning for semi-autonomous driving. In

$^1$A $G^2$ stands for geometric continuity of 2nd order, which is a less stringent condition of $C^2$ continuity, but often more appropriate for geometric curves.
particular, we use a two stage sampling to bias the growth of the random tree such that it can explore the whole environment rapidly, but also refine the path returned to improve the level of optimality of the solution. Also, the cost function penalizes the non-smoothness of the path. We also penalize the difference between the curvature of the RRT* path and the reference path, since in semi-autonomous driving it is expected that the maneuver path should be close to the estimated driver path in terms of curvature. Next, because of the randomness of RRT*, the path returned by RRT* often has some unnecessary waypoints. To deal with this problem, we propose an algorithm to remove such redundant waypoints, and we use the smoothing technique proposed in [11] to obtain a \( G^2 \) continuous path. To enable our algorithm to deal with sudden changes in environment, like [12] we apply the replanning procedure to update the path after changes in environment are detected, while avoiding full recomputation due to the limited available time.

Thus, the proposed algorithm generates paths that are \( G^2 \) continuous and minimize the modifications to the estimated driver path, improves path quality by a two stage sampling, and reacts to changes in the environment in a real-time manner.

The rest of the paper is organized as follows. The path planning problem is formulated in Section II. The algorithm is introduced in Section III. Results of simulations are given in Section IV, and conclusions are drawn in Section V.

II. PROBLEM FORMULATION

Denote a configuration of the vehicle in a 2D environment by \( x := (p_x, p_y, \psi) \), where \( p_x \) and \( p_y \) are the \( x \) and \( y \) coordinates of the vehicle’s position, and \( \psi \) is the orientation of the vehicle, \( \psi \in (-\pi, \pi] \). In the remainder of this paper, we also use \( x = (p, \psi) \) to denote a configuration, where \( p = (p_x, p_y) \) is the position of the vehicle in the 2D environment.

Denote the initial configuration and goal configuration of the vehicle by \( x_{\text{init}} \) and \( x_{\text{goal}} \), respectively. Denote the bounded and connected configuration state space by \( X \in \mathbb{R}^3 \). Denote the obstacle region and the obstacle-free region in \( X_{\text{obs}} \) and \( X_{\text{free}} := X \setminus X_{\text{obs}} \), respectively. A feasible path in the configuration space is \( \sigma : [0, S] \rightarrow X_{\text{free}} \), where \( S \) is the total length of the path.

Denote by \( \Sigma_{X_{\text{free}}} \) all the feasible paths in \( X_{\text{free}} \). Denote by \( \sigma_{x_1,x_2} \) the path between two states \( x_1 \) and \( x_2 \). Denote a reference path in \( X \) by \( \sigma_{\text{ref}} \), where we note that \( \sigma_{\text{ref}} \) is in \( X \), but not necessarily in \( X_{\text{free}} \). Let \( c : \Sigma_{X_{\text{free}}} \rightarrow \mathbb{R}_{\geq 0} \) be the cost function, which assigns a non-negative cost to all nontrivial collision-free paths. We assume that \( c \) is additive, that is, if \( x_3 \) is a point on the path connecting \( x_1 \) and \( x_3 \), then \( c(\sigma_{x_1,x_3}) = c(\sigma_{x_1,x_2}) + c(\sigma_{x_2,x_3}) \). Denote by \( \sigma_{x_1,x_2}^* \) the optimal path connecting \( x_1 \) and \( x_2 \) with respect to \( c \) and the corresponding optimal cost by \( c_{x_1,x_2}^* := c(\sigma_{x_1,x_2}^*) \). Given a tree \( T = (V, E) \) rooted at \( x_{\text{root}} \), for any vertex \( v \in V \) inside the tree, \( \text{Cost}(v) \) is the optimal cost of the path from the root to \( v \), that is, \( \text{Cost}(v) := c_{x_{\text{root}},v}^* \). The path planning problem for semi-autonomous driving is as follows.

Problem 1: Given a bounded and connected configuration space \( X \), an obstacle region \( X_{\text{obs}} \), an initial state \( x_{\text{init}} \in X_{\text{free}} \), a goal state \( x_{\text{goal}} \in X_{\text{free}} \) and a twice continuously differentiable reference path \( \sigma_{\text{ref}} \), find a \( G^2 \) continuous path \( \sigma^* : [0, S] \rightarrow X_{\text{free}} \) such that (i) \( \sigma^*(0) = x_{\text{init}} \) and \( \sigma^*(S) = x_{\text{goal}} \), and (ii) \( c(\sigma^*) = \min_{\sigma \in \Sigma_{X_{\text{free}}}} c(\sigma) \), where \( c(\sigma) \) is a cost function which penalizes the difference of curvatures between the reference path \( \sigma_{\text{ref}} \) and \( \sigma \). If no such path exists, report failure.

III. PATH PLANNING FOR SEMI-AUTONOMOUS DRIVING

In this section, we combine Rapidly-exploring Random Tree Star (RRT*) with some smoothing techniques to get a \( G^2 \) continuous path for semi-autonomous driving. We first discuss our planning algorithm in the case of a static environment, then propose a replanning procedure to deal with dynamic environments.

A. RRT*

Before we present our algorithm, we first introduce the primitive procedures used in RRT*.

**Sampling:** The function \( \text{SampleFree} : \mathbb{Z}_{>0} \rightarrow X_{\text{free}} \) returns independent identically distributed (i.i.d.) samples from \( X_{\text{free}} \). In this paper, the samples are assumed to be uniformly distributed. In Figure 1, \( \text{SampleFree} \) returns \( x_{\text{rand}} \).

**Nearest Neighbor:** Given a tree \( T = (V, E) \) and a point \( x \in X_{\text{free}} \), the function \( \text{Nearest} : (T, x) \rightarrow v \) returns a vertex \( v \in V \) which is closest to \( x \) in terms of a cost metric, that is, \( \text{Nearest}(T, x) := \arg \min_{v \in V} c_{x,v} \).

**Steering:** Given two states \( x_1, x_2 \in X_{\text{free}} \) and a positive real number \( \eta \in \mathbb{R}_{>0} \), the function \( \text{Steer} : (x_1, x_2, \eta) \rightarrow x_3 \) returns a state \( x_3 \in X_{\text{free}} \) such that \( x_3 \) is a point on the optimal path connecting \( x_1 \) and \( x_2 \). In this paper, \( x_3 = \text{Steer}(x_1, x_2, \eta) \), where \( x_3 := (p_3, \psi_3) \in \sigma_{x_1,x_2}^* \) such that \( \|p_3 - p_1\| \leq \eta, \psi_3 = \psi_2 \), and where \( \eta \) is the extension segment length in RRT*. Thus, \( x_3 \) is relatively close to \( x_1 \) to have negligible probability that the path from \( x_1 \) to \( x_3 \) collides with obstacles, we limit the Euclidean distance between \( x_1 \) and \( x_3 \) to \( \eta \).

**Near Vertices:** Given a tree \( T = (V, E) \), a state \( x = (p, \psi) \in X_{\text{free}} \) and a positive real number \( r \in \mathbb{R}_{>0} \), the function \( \text{Near} : (T, x, r) \rightarrow V' \subseteq V \) returns the vertices in \( V \) that are in a neighborhood of radius \( r \) from \( x \). That is, \( \text{Near}(T, x, r) := \{ v \in V : \|p_v - p\| \leq r \} \). We choose \( r \) as a function of the number of vertices in the tree, and specifically, \( r(|V|) = \min(\gamma |\log|V|/|V|^2, \eta) \), where \( \gamma > \gamma^* := \frac{2(1 + 1/d)^{1/d}}{\mu(X_{\text{free}})/\zeta_3^{1/d}} \mu(X_{\text{free}}) \) is the volume of the free space, \( \zeta_3 \) is the volume of the unit ball in \( \mathbb{R}^d \), and \( |V| \) denotes the number of vertices in \( V \). In Figure 1, \( \text{Near} \) returns \( x_{1,2,3} \).

**Collision Test:** Given two states \( x_1, x_2 \in X_{\text{free}} \), the Boolean function \( \text{CollisionFree}(x_1, x_2) \) returns True if the optimal path \( \sigma_{x_1,x_2}^* \) between \( x_1 \) and \( x_2 \) lies in \( X_{\text{free}} \) and False otherwise.

More details on RRT* and its primitives are in [3].
the path, as described later. The sampling takes place in two stages. In the first stage, we sample in the whole environment. In this stage, the goal of sampling is to rapidly explore the entire environment and to find a good enough path between the initial configuration and the goal configuration. After such a path is found in the first stage, then the sampling enters into the second stage. In the second stage, we sample in the neighborhood of the waypoints and refine the path. Given a path with waypoints \( \{x_{init}, x_1, x_2, \ldots, x_n, x_{goal}\} \), we first choose a waypoint from these waypoints randomly, denote the randomly chosen waypoint by \( x_i \), then we sample uniformly in the cylinder centered at this chosen waypoint \( C_{x_i, r'} := \{x = (p, \psi) \in X \mid \|p - p_i\| \leq r', |\psi - \psi_i| \leq \delta\} \). Here, the radius of the ball is usually chosen as \( r' = \eta \), where \( \eta \) is the extension segment length of RRT*, and \( \delta \) is usually chosen in \( (0, \frac{\pi}{2}) \). When executing the algorithm, we usually draw more samples in the first stage than in the second stage.

The reason is that it is important to let RRT* explore the entire environment thoroughly to find a good enough path before entering into the second stage to perform refinement.

2) Cost Function: Given two states \( x_1 = (p_1, \psi_1) \) and \( x_2 = (p_2, \psi_2) \), where \( p_1 = (p_{x_1}, p_{y_1}) \) and \( p_2 = (p_{x_2}, p_{y_2}) \), we choose the cost function as follows.

\[
c(\sigma_{x_1, x_2}) = w_1 \|p_1 - p_2\| + w_2(\theta_1 + \theta_2) + w_3(\kappa(x_1) - \kappa_{ref}(x_1, \sigma_{ref}))^2
\]  

(1)

where \( w_1 \) is the weight on the Euclidean distance, \( w_2 \) is the weight on the smoothness of the path, and \( w_3 \) is the weight on the difference between the curvature of the RRT* path and the reference path, \( \|\cdot\| \) is the Euclidean norm, \( \theta_1 \) is the angle between \( \psi_1 \) and the vector \( \overrightarrow{p_1p_2} \). Specifically, \( \theta_1 = \frac{\psi_1 - \psi_2}{\|\overrightarrow{p_1p_2}\|} \), \( u_1 = [\cos(\psi_1), \sin(\psi_1)]^T \). Similarly, \( \theta_2 \) is the angle between \( \psi_2 \) and the vector \( \overrightarrow{p_1p_2} \), \( \theta_2 \in (0, \pi) \). \( \kappa(x_1) \) is the curvature at vertex \( x_1 \) and \( \kappa_{ref}(x_1, \sigma_{ref}) \) is the corresponding reference curvature at \( x_1 \). Next we show how to calculate \( \kappa(x_1) \) and \( \kappa_{ref}(x_1, \sigma_{ref}) \) given \( x_1, x_2 \) and \( \sigma_{ref} \).

In the smoothing procedure, we will use \( G^2 \) Continuous Cubic Bézier Spiral (G2CBS) to connect two edges [11]. Because of this, we use the maximum curvature of the G2CBS curve as the curvature at the joining vertex of the two edges, see Figure 2(a). Let vertex \( x_1 \) be given and its parent in the tree be \( x_{parent} = (p_{parent}, \psi_{parent}) \), then the magnitude of the curvature at \( x_1 \) is given by

\[
|\kappa(x_1)| = \frac{q_4 \sin \beta}{L \cos^2 \beta}
\]  

(2)

where \( q_4 \) is a parameter satisfying \( q_1 = 7.2364, q_2 = \frac{2}{\sqrt{5}} (\sqrt{6} - 1), q_3 = q_2 + 4, q_4 = \frac{q_2 + 4}{2q_3}, \beta = \frac{\pi}{2} \) and \( \gamma \) is the angle between the vector \( \overrightarrow{p_{parent}p_1} \) and the vector \( \overrightarrow{p_1p_2} \), \( \beta \in (0, \frac{\pi}{2}) \), and \( L \) is the distance between \( B_0 \) and \( p_1 \). The sign of \( \kappa(x_1) \) is determined by the rotation direction of the unit tangent vector at \( p_1 \) in the 2D plane. Specifically, if \( B_0 \) is the starting point of the curve and \( E_0 \) is the end point of the curve, then if the unit tangent vector rotates counterclockwise, \( \kappa(x_1) > 0 \). Instead, if it rotates clockwise, then \( \kappa(x_1) < 0 \). We can
determine the sign of $\kappa(x_1)$ by considering the direction of the cross product $\frac{p_{\text{parent}}p_1}{\lVert p_{\text{parent}}p_1 \rVert} \times \frac{p_1p_2}{\lVert p_1p_2 \rVert}$. That is,

$$\text{sgn}(\kappa(x_1)) = \begin{cases} 
1 & \text{if } \left(\frac{p_{\text{parent}}p_1}{\lVert p_{\text{parent}}p_1 \rVert} \times \frac{p_1p_2}{\lVert p_1p_2 \rVert}\right) \cdot \vec{e} > 0 \\
0 & \text{if } \left(\frac{p_{\text{parent}}p_1}{\lVert p_{\text{parent}}p_1 \rVert} \times \frac{p_1p_2}{\lVert p_1p_2 \rVert}\right) \cdot \vec{e} = 0 \\
-1 & \text{if } \left(\frac{p_{\text{parent}}p_1}{\lVert p_{\text{parent}}p_1 \rVert} \times \frac{p_1p_2}{\lVert p_1p_2 \rVert}\right) \cdot \vec{e} < 0 
\end{cases}$$

where $\vec{e}$ is a unit vector perpendicular to both $\frac{p_{\text{parent}}p_1}{\lVert p_{\text{parent}}p_1 \rVert}$ and $\frac{p_1p_2}{\lVert p_1p_2 \rVert}$ which completes a right-handed system. Here $\left(\frac{p_{\text{parent}}p_1}{\lVert p_{\text{parent}}p_1 \rVert} \times \frac{p_1p_2}{\lVert p_1p_2 \rVert}\right) \cdot \vec{e} = 0$ means $p_{\text{parent}}p_1$ and $p_1p_2$ are collinear, and the curvature is $\kappa(x_1) = 0$.

We compute $\kappa_{\text{ref}}(x_1, \sigma_{\text{ref}})$ by projecting $x_1$ to a point on the reference path $\sigma_{\text{ref}}$ and using the curvature of the resulting point as reference curvature for $x_1$. In this work, we use the nearest point projection,

$$\kappa_{\text{ref}}(x_1, \sigma_{\text{ref}}) = \kappa(\arg\min_{x_{\text{ref}} \in \sigma_{\text{ref}}} \|p_{\text{ref}} - p_1\|)$$

where $x_{\text{ref}} = (p_{\text{ref}}, \psi_{\text{ref}})$ is any point on $\sigma_{\text{ref}}$. One advantage of (4) is that the reference curvature of each vertex inside the tree needs only to be calculated once. After it is calculated, we can store it in memory, and use it in later tree expansions. This is computationally efficient, especially for the case of a large number of iterations in RRT*. Based on cost function (1), the nearest neighbor is selected by calculating the cost from each vertex to the random node and by setting the vertex which has the lowest cost as the nearest neighbor to the random node. For example, in Figure 2(b), if the weight on the curvature difference between the RRT* path and the reference path is increased, $x_2$ becomes the nearest neighbor. If the focus is on reducing the Euclidean distance and hence $w_1$ is (relatively) large, $x_1$ is the nearest neighbor.

3) Root of the Tree and Path Found Criterion: We set the RRT* tree root at $x_{\text{goal}}$, which is especially useful in the case of dynamic obstacles. When the environment changes, we need to remove the branches of the tree that collide with obstacles. Since our goal is to find a path leading to $x_{\text{goal}}$, we would like not to remove the branch leading to $x_{\text{goal}}$. If we set the tree root at $x_{\text{goal}}$, the branch leading to $x_{\text{goal}}$ is never removed unless the entire tree becomes invalid, which seldom happens and usually requires a goal state re-definition. As for the stopping criterion, the algorithm will return a path between $x_{\text{init}}$ and $x_{\text{goal}}$ when it generates a random node inside the cylinder centered at $x_{\text{init}} = (p_{\text{init}}, \psi_{\text{init}})$ and the path connecting the random node to $x_{\text{init}}$ is collision free. The cylinder is defined as

$$C_{\text{init}, \eta, \Psi} := \{x = (p, \psi) \mid \|p - p_{\text{init}}\| \leq \eta, |\psi - \psi_{\text{init}}| \leq \Psi\},$$

where $\Psi$ is the maximum yaw difference between $\psi_{\text{init}}$ and its parent’s configuration.

4) Remove Redundant Points: Because of the randomness of the algorithm, there are some unnecessary waypoints in the path returned by RRT*. We can remove the redundant waypoints as follows. Let the waypoints returned by RRT* be $\sigma = \{x_0, x_1, x_2, \ldots, x_n, x_{n+1}\}$, where $x_i = (p_i, \psi_i)$, $x_0 = x_{\text{init}}$ and $x_{n+1} = x_{\text{goal}}$. Let the pruned path be $\sigma_p$, set $\sigma_p = \emptyset$, and let $j = n + 1$. The pruning algorithm works as follows. Add $x_j$ to $\sigma_p$. Starting from $x_0$, find all the waypoints in $\{x_0, x_1, \ldots, x_{j-1}\}$ such that $\sigma^*_p, \sigma_j (i \in [0, j - 1])$ is collision free, denote these waypoints by $X_{\text{candidate}}$. For all the $x_k \in X_{\text{candidate}}$, find the one which minimizes $(\theta_k + \theta_j)$, where $\theta_k = \frac{\psi_k - \psi_{\text{init}}}{\|p_k - p_{\text{init}}\|}$, $u_k = [\cos(\psi_k), \sin(\psi_k)]^T$ and $\theta_j = \frac{\psi_j - \psi_{\text{init}}}{\|p_j - p_{\text{init}}\|}$, $u_j = [\cos(\psi_j), \sin(\psi_j)]^T$. Then let $j = k$ and repeat this process until $k = 0$, that is, a pruned path is found between $x_{\text{init}}$ and $x_{\text{goal}}$. An example of pruning is given in Figure 2(c). The pseudo code of the pruning algorithm is shown as Algorithm 1.

5) Smoothing: We use $G^2$ Continuous Cubic Bézier Spiral (G2CBS) [11] to generate a continuous curvature path between two consecutive line segments, see Figure 2(a). A Bézier curve is defined as

$$P(s) = \sum_{i=0}^{n} \binom{n}{i} (1-s)^{n-i} s^i P_i$$

where $n$ is the degree of the Bézier curve, $s \in [0, 1]$, $P_i$ are the control points. As in [11], the eight control points in (5) are

$$B_0 = p_1 + Lu_1, \quad B_1 = B_0 - g_0u_1, \quad B_2 = B_1 - h_bu_1$$
$$B_3 = B_2 + h_bu_d, \quad E_0 = p_1 + Lu_2, \quad E_1 = E_0 - g_eu_2$$
$$E_2 = E_1 - h_eu_2, \quad E_3 = E_2 - k_eu_d$$

where $u_1$ is the unit vector along the line $\overrightarrow{x_{1}x_{\text{parent}}}$, $u_2$ is the unit vector along the line $\overrightarrow{x_{\text{nearest}}x_{\text{parent}}}$, $u_d$ is the unit vector along the line $\overrightarrow{B_2E_2}$, and

$$h_b = h_e = \frac{q_3L}{2}, \quad g_0 = g_e = \frac{q_3L}{2}, \quad k_e = \frac{6q_3\cos \beta}{q_2 + 4}L$$
where $\beta = \frac{\gamma}{2}$ and $\gamma$ is the angle between vector $\overrightarrow{x_{parent}x_1}$ and $\overrightarrow{x_1x_2}$. The parameters $q_1, q_2, q_3, q_4$ are

$$q_1 = 7.2364, q_2 = \frac{2}{5}(\sqrt{6} - 1), q_3 = \frac{q_2 + 4}{q_1 + 6}, q_4 = \frac{(q_2 + 4)^2}{54q_3},$$

$L$ is the distance between $p_1$, where $x_1 = (p_1, \psi_1)$, and $B_0$, and it is also the distance between $p_1$ and $E_0$. Based on this, we choose $L$ as,

$$L = \min \left\{ \frac{\min\{\|p_{parent}p_1\|, \|p_1p_2\|\}}, \eta \right\} \cdot \frac{\eta}{2}. \tag{7}$$

The reason for choosing $L$ by (7) is that one edge is used twice for smoothing, so $2L$ should be less than or equal to $\min\{\|p_{parent}p_1\|, \|p_1p_2\|\}$, $\eta$. By the smoothing technique, we obtain a $G^2$ continuous path between $p_{init}$ and $p_{goal}$. If we also require that the path is continuous at $p_{init}$ and $p_{goal}$, we add one waypoint in the yaw direction $\psi_{init}$ and one waypoint in the yaw direction $\psi_{goal}$, and then use RRT* to find a set of waypoints between these two new waypoints. After this, we apply the smoothing technique to all the waypoints from $p_{init}$ to $p_{goal}$, which results in a $G^2$ continuous path between $p_{init}$ and $p_{goal}$, which is also smooth at both $p_{init}$ and $p_{goal}$.

C. Path Planning in Dynamic Environment

Next, similar to [12], we propose a replanning methodology when changes occur in the environment. We assume that we can quickly capture the changes in the environment and need to rapidly replan if the original path is affected by such changes. There are two choices for replanning. Obviously, one can replan from scratch, which however is a computationally expensive operation, especially if the environment changes frequently. On the other hand one can modify the original planning information and perform only minimal computation to adjust the path to the new environment. In this paper, we use the second approach since it reduces the computations, and hence the time to obtain the new path. The replanning procedure consists of trimming the tree and regrowing the tree. When a new obstacle appears or an obstacle changes position, first the edges which collide with the new obstacles are found. For each edge, we denote the two endpoints as parent endpoint and child endpoint, respectively. For all the edges which intersect with obstacles, we mark their child endpoints as invalid. After all the child endpoints are marked, we check whether the original path from $x_{init}$ to $x_{goal}$ has any invalid nodes. If it has, then we trim and regrow the tree. Trimming is performed by traversing through each node of the tree and marking all the child of the invalid nodes as invalid. Then, all the invalid nodes and the edges connecting to them are removed from the tree. After the tree is trimmed, it can be grown again to find a new path. When regrowing the tree, it is usually enough to generate samples in the neighborhood of the area affected by the new obstacles, which can generate a new branch to cover this area and find a new path quickly with reduced computations. The pseudocode of the replanning procedure is given as Algorithm 2.

Algorithm 1 Removal of Redundant Points

Input: $\sigma = \{x_0, x_1, x_2, \ldots, x_n, x_{n+1}\}$
1: $x_i \leftarrow x_{n+1}; \; \sigma_p \leftarrow \emptyset$
2: while $1$ do
3:   Add $x_i$ to $\sigma_p$
4:   $X_{candidate} \leftarrow \emptyset$
5:   for $i=0$ to $i=j-1$ do
6:     if CollisionFree$(x_i, x_j)$ then
7:        Add $x_i$ to $X_{candidate}$
8:     end if
9:   end for
10:   $x_{min} \leftarrow x_k \in X_{candidate}$
11:   $J_{min} \leftarrow (\frac{u_1}{p_k p_j}) + \frac{u_2}{p_k p_j}$
12:   for each $x_k \in X_{candidate} \setminus \{x_k\}$ do
13:      $J \leftarrow (\frac{u_1}{p_k p_j}) + \frac{u_2}{p_k p_j}$
14:  if $J < J_{min}$ then
15:    $J_{min} \leftarrow J, x_{min} \leftarrow x_k$
16: end if
17: end for
18: $x_j \leftarrow x_{min}$
19: if $x_j = x_0$ then
20:   Add $x_0$ to $\sigma_p$
21: end if
22: Break while loop;
23: end while
24: return $\sigma_p$

Algorithm 2 Replanning

Input: $\sigma = \{x_0, x_1, x_2, \ldots, x_n, x_{n+1}\}, T = (V, E), X_{obs}$
1: for each $e \in E$ do
2:   Let the two endpoints of $e$ be: $x^e_c \rightarrow x^e_o$
3:   if not CollisionFree$(x^e_c, x^e_o)$ then
4:     Mark $x^e_c$ as INVALID;
5:     end if
6: end for
7: if $\sigma$ does NOT have any INVALID nodes then
8:   return $\sigma$
9: else
10: $V_{valid} = \emptyset$
11: for each node $x_i \in V$ do
12:   $x_p \leftarrow $ Parent$(x_i)$
13:   if $x_p$ is INVALID then
14:     Mark $x_i$ as INVALID;
15:     end if
16: end if
17: if $x_i$ is VALID then
18:   $V_{valid} \leftarrow x_i$
19: end if
20: end for
21: Delete all the INVALID nodes from the tree;
22: Apply RRT* to regrow the tree;
23: Return a new path $\sigma_{new}$;
24: end if

Our planning algorithm for semi-autonomous driving is summarized as Algorithm 3.

Algorithm 3 Path Planning for Semi-Autonomous Driving

Input: $x_{init}, x_{goal} \in X, \sigma_{ref}$
1: Run RRT* algorithm with stage one sampling;
2: Run RRT* algorithm with stage two sampling;
3: Remove redundant waypoints by Algorithm 1;
4: Apply smoothing technique to get $G^2$ continuous path;
5: If environment changes, replan by Algorithm 2;
D. Computational Complexity

As discussed in [3], the computational complexity of RRT* is $O(N \log N)$. In our algorithm, we have added the replanning procedure which needs to traverse the tree, which we perform by depth-first search (DFS). The time complexity of DFS is $O(N)$ [13], and hence the computational complexity Algorithm 3 is still $O(N \log N)$.

IV. Numerical Simulations

This section presents numerical simulations of Algorithm 3. We consider planning a maneuvering path for the semi-autonomous vehicle on a roadway. We choose the parameters of the simulation as follows. The initial configuration and final configuration are $x_{\text{init}} = (120, 45, 0)$ and $x_{\text{goal}} = (0, 5, 0)$, respectively. The tree is rooted at $x_{\text{root}} = x_{\text{goal}}$. For the primitive procedures, we choose $\eta = 5$ for the $\text{Steer}$ function. The parameter $\gamma$ in the function $\text{Near}$ is chosen as $\gamma = (2(1 + 1/2)^{1/2} \mu(X_{f_{\text{free}}})/\zeta_2)^{1/2} + 1$. In the second stage of sampling, $\delta$ is set to $\frac{\gamma}{2}$. In the stopping criterion, $\Psi$ is chosen as $\frac{\gamma}{6}$. Figure 3(a) shows the path returned by our algorithm with different weights in the cost.

When the vehicle starts moving along the path returned by our algorithm (see Figure 3(b)), if some obstacles move or a new obstacle is detected, we need to replan. When replanning, we first trim the tree and remove the branches which collide with the obstacles, see Figure 3(c). Based on the trimmed tree, we use RRT* to find a new path between the current vehicle position and the goal position. Figure 3(d) shows the path found by the replanning procedure.

V. Conclusions and Future Work

In this paper we have developed an RRT*-based algorithm for semi-autonomous vehicles. Our algorithm is based on a two stage sampling, which accounts for obstacle avoidance, path length, and also path curvature and curvature difference with an estimated driver reference path. A smoothing technique is applied to get a $C^2$ continuous path. By applying the replanning procedure, our algorithm can also deal with sudden changes in the environment. The two stage sampling and the replanning procedure rapidly provides new and updated paths. This work can be extended to deal with uncertain environment, such as in presence of uncertainty in obstacle positions. Also, the vehicle dynamics can be taken into account, for instance, following the ideas in [14] which considers the differential constraint on the vehicle dynamics.

REFERENCES