MITSUBISHI ELECTRIC RESEARCH LABORATORIES
http://www.merl.com

Continuous Curvature Path Planning for Semi-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

X. Lan is with the Department of Mechanical Engineering, Boston University, Boston, MA 02215, USA, xlan@bu.edu.

S. Di Cairano is with the Mitsubishi Electric Research Laboratories, Cambridge, MA 02139, USA, dicairano@merl.com.

[1] 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 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 ψ 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_{init} and x_{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 the configuration space by X_{obs} and $X_{free} := X \setminus X_{obs}$, respectively. A feasible path in the configuration space is $\sigma : [0, S] \mapsto X_{free}$, where S is the total length of the path. Denote by $\Sigma_{X_{free}}$ all the feasible paths in X_{free}. Denote by σ_{x_1,x_2} the path between two states x_1 and x_2. Denote a reference path in X by σ_{ref}, where we note that σ_{ref} is in X, but not necessarily in X_{free}. Let $c : \Sigma_{X_{free}} \mapsto \mathbb{R}_{\geq 0}$ 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_2 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_{root}, for any vertex $v \in V$ inside the tree, $\mathsf{Cost}(v)$ is the optimal cost of the path from the root to v, that is, $\mathsf{Cost}(v) := c^*_{x_{root},v}$. 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_{obs}, an initial state $x_{init} \in X_{free}$, a goal state $x_{goal} \in X_{free}$ and a twice continuously differentiable reference path σ_{ref}, find a G^2 continuous path $\sigma^* : [0, S] \mapsto X_{free}$ such that (i) $\sigma^*(0) = x_{init}$ and $\sigma^*(S) = x_{goal}$, and (ii) $c(\sigma^*) = \min_{\sigma \in \Sigma_{X_{free}}} c(\sigma)$, where $c(\sigma)$ is a cost function which penalizes the difference of curvatures between the reference path σ_{ref} and σ. 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 $\mathsf{SampleFree} : \mathbb{Z}_{>0} \mapsto X_{free}$ returns independent identically distributed (i.i.d.) samples from X_{free}. In this paper, the samples are assumed to be uniformly distributed. In Figure 1, SampleFree returns x_{rand}.

Nearest Neighbor: Given a tree $T = (V, E)$ and a point $x \in X_{free}$, the function $\mathsf{Nearest} : (T, x) \mapsto v$ returns a vertex $v \in V$ which is closest to x in terms of a cost metric, that is, $\mathsf{Nearest}(T, x) := \mathrm{argmin}_{v \in V} c^*_{v,x}$. In Figure 1, Nearest returns $x_{nearest}$.

Steering: Given two states $x_1, x_2 \in X_{free}$ and a positive real number $\eta \in \mathbb{R}_{>0}$, the function $\mathsf{Steer} : (x_1, x_2, \eta) \mapsto x_3$ returns a state $x_3 \in X_{free}$ such that x_3 is a point on the optimal path connecting x_1 and x_2. In this paper, $x_3 = \mathsf{Steer}(x_1, x_2, \eta)$, 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 η 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 η. In Figure 1, Steer returns x_{new}.

Near Vertices: Given a tree $T = (V, E)$, a state $x = (p, \psi) \in X_{free}$ and a positive real number $r \in \mathbb{R}_{>0}$, the function $\mathsf{Near} : (T, x, r) \mapsto V' \subseteq V$ returns the vertices in V that are in a neighborhood of radius r from x. That is, $\mathsf{Near}(T, x, r) = \{v = (p_v, \psi_v) \in V, |\ \|p_v - p\| \leq r\}$. We choose r as a function of the number of vertices in the tree, and specifically, $r(|V|) = \min\{\gamma(\log|V|/|V|)^{1/2}, \eta\}$, where $\gamma > \gamma^* = (2(1 + 1/d)^{1/d} \mu(X_{free})/\zeta_2)^{1/d}$, $\mu(X_{free})$ is the volume of the free space, ζ_d is the volume of the unit ball in $\mathbb{R}^d$, and $|V|$ denotes the number of vertices in V. In Figure 1, Near returns x_1, x_2, x_3.

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

More details on RRT* and its primitives are in [3].

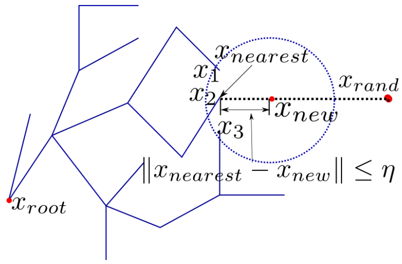


Fig. 1. Example of tree expansion in RRT*.

B. Path Planning in Static Environment

Next, we adapt RRT* to generate a path for semi-autonomous driving subject to global and local constraints, where global constraints are expressed in terms of path properties, and local constraints are expressed in terms of node properties. A global constraint is to avoid obstacles, which can be static or dynamic. We first discuss our planning algorithm in an environment with static obstacles and later we extend our work to consider dynamic obstacles. For static environment, we assume that we know the geometry, position and orientation of the obstacles. Furthermore, under normal conditions, the vehicle moves along a reference path, e.g., a driver commanded path, and we assume that we know an estimate of this path in advance. However, such a reference path may collide with obstacles, for example, a stopped vehicle, or a pedestrian on the road, or a lane obstacle. In these situations, we expect that the vehicle maneuvers to avoid such obstacles without excessive deviation from the reference path. Thus, we need to plan a path that starts from the reference path, performs the needed maneuvers and comes back to the reference path.

The local constraints are related to the kinematic and dynamic behavior of the vehicle. The main kinematic constraint considered in this paper is that the curvature κ of the maneuver path needs to be close to the reference curvature κ_{ref} of the reference path. For the dynamic constraints, the curvature should be continuous since discontinuities in the curvature may cause infinite variations of lateral acceleration and velocity which, obviously, cannot be achieved in practice. Hence, the path returned by the planning algorithm should be at least second order differentiable. In this work, we search for a G^2 continuous path, which means that two consecutive segments of the path should have the same tangent line and the same center of curvature at their joint point. In order to find a path satisfying these constraints, we propose the following modifications to RRT*.

1) Two Stage Sampling: In this work we sample in the configuration space. That is, we sample the position of the vehicle as well as its orientation $(p_{x_{rand}}, p_{y_{rand}}, \psi_{rand})$, where $(p_{x_{rand}}, p_{y_{rand}})$ is sampled uniformly from the bounded environment and ψ_{rand} is sampled uniformly between $(-\pi, \pi]$. The reason for sampling the orientation is that we require smoothness of the path and we want to limit the orientation change between two consecutive waypoints. Also, we use the orientation to remove redundant waypoints from the returned path, thus achieving a further smoothing of 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 $\mathcal{C}_{x_i,r'} := \{x = (p, \psi) \in X | \|p - p_i\| \leq r', |\psi - \psi_i| \leq \delta\}$. Here, the radius of the ball is usually chosen as $r' = \eta$, where η is the extension segment length of RRT*, and δ 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.

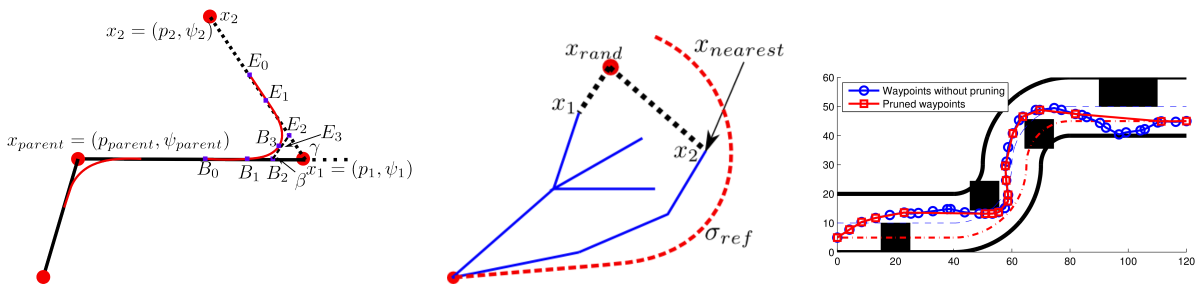
$$c(\sigma_{x_1,x_2}) = w_1\|p_1 - p_2\| + w_2(\theta_1 + \theta_2) + w_3\big(\kappa(x_1) - \kappa_{ref}(x_1, \sigma_{ref})\big)^2 \quad (1)$$

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, θ_1 is the angle between ψ_1 and the vector $\overrightarrow{p_1p_2}$. Specifically, $\theta_1 = \frac{u_1 \cdot \overrightarrow{p_1p_2}}{\|\overrightarrow{p_1p_2}\|}, u_1 = [\cos(\psi_1), \sin(\psi_1)]^T$. Similarly, θ_2 is the angle between ψ_2 and the vector $\overrightarrow{p_1p_2}$, $\theta_1, \theta_2 \in [0, \pi]$. $\kappa(x_1)$ 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 σ_{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

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

where q_4 is a parameter satisfying $q_1 = 7.2364, q_2 = \frac{2}{5}(\sqrt{6}-1), q_3 = \frac{q_2+4}{q_1+6}, q_4 = \frac{(q_2+4)^2}{54q_3}$, $\beta = \frac{\gamma}{2}$ and γ 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



(a) Path smoothing between two edges by G2CBS.

(b) Based on the proposed cost, x_2 is the nearest neighbor.

(c) Pruning unnecessary waypoints.

Fig. 2. Proposed modifications to RRT* for semi-autonomous driving.

determine the sign of $\kappa(x_1)$ by considering the direction of the cross product $\overrightarrow{p_{parent}p_1} \times \overrightarrow{p_1p_2}$. That is,

$$sgn(\kappa(x_1)) = \begin{cases} 1 & \text{if } (\overrightarrow{p_{parent}p_1} \times \overrightarrow{p_1p_2}) \cdot \vec{e} > 0 \\ 0 & \text{if } (\overrightarrow{p_{parent}p_1} \times \overrightarrow{p_1p_2}) \cdot \vec{e} = 0 \\ -1 & \text{if } (\overrightarrow{p_{parent}p_1} \times \overrightarrow{p_1p_2}) \cdot \vec{e} < 0 \end{cases} \quad (3)$$

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

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

$$\kappa_{ref}(x_1, \sigma_{ref}) = \kappa(\text{argmin}_{x_{ref} \in \sigma_{ref}} \|p_{ref} - p_1\|) \quad (4)$$

where $x_{ref} = (p_{ref}, \psi_{ref})$ is any point on σ_{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 is 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_{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_{goal}, we would like not to remove the branch leading to x_{goal}. If we set the tree root at x_{goal}, the branch leading to x_{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_{init} and x_{goal} when it generates a random node inside the cylinder centered at $x_{init} = (p_{init}, \psi_{init})$ and the path connecting the random node to x_{init} is collision free. The cylinder is defined as $C_{x_{init},\eta,\Psi} := \{x = (p, \psi) \in X | \|p - p_{init}\| \leq \eta, |\psi - \psi_{init}| \leq \Psi\}$, where Ψ is the maximum yaw difference between ψ_{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 $x_i = (p_i, \psi_i)$, $x_0 = x_{init}$ and $x_{n+1} = x_{goal}$. Let the pruned path be σ_p, set $\sigma_p = \emptyset$, and let $j = n + 1$. The pruning algorithm works as follows. Add x_j to σ_p. Starting from x_0, find all the waypoints in $\{x_0, x_1, \ldots, x_{j-1}\}$ such that $\sigma^*_{x_i,x_j} (i \in [0, j - 1])$ is collision free, denote these waypoints by $X_{candidate}$. For all the $x_k \in X_{candidate}$, find the one which minimizes $(\theta_k + \theta_j)$, where $\theta_k = \frac{u_k \cdot \overrightarrow{p_kp_j}}{\|\overrightarrow{p_kp_j}\|}$, $u_k = [\cos(\psi_k), \sin(\psi_k)]^T$ and $\theta_j = \frac{u_j \cdot \overrightarrow{p_kp_j}}{\|\overrightarrow{p_kp_j}\|}$, $u_j = [\cos(\psi_j), \sin(\psi_j)]^T$. Then let $j = k$ and repeat this process until $k = 0$, that is, a pruned path is found between x_{init} and x_{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

$$P(s) = \sum_{i=0}^{n} \binom{n}{i} (1 - s)^{n-i} s^i P_i \quad (5)$$

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

$$\begin{aligned} &B_0 = p_1 + Lu_1, \quad B_1 = B_0 - g_b u_1, \quad B_2 = B_1 - h_b u_1 \\ &B_3 = B_2 + k_b u_d, \quad E_0 = p_1 + Lu_2, \quad E_1 = E_0 - g_e u_2 \\ &E_2 = E_1 - h_e u_2, \quad E_3 = E_2 - k_e u_d \end{aligned}$$

where u_1 is the unit vector along the line $\overrightarrow{x_1x_{parent}}$, u_2 is the unit vector along the line $\overrightarrow{x_1x_2}$, u_d is the unit vector along the line $\overrightarrow{B_2E_2}$, and

$$h_b = h_e = q_3 L, \quad g_b = g_e = q_2 q_3 L, \quad k_b = k_e = \frac{6q_3 \cos\beta}{q_2 + 4} L$$

Algorithm 1 Removal of Redundant Points

Input: $\sigma = \{x_0, x_1, x_2, \dots, x_n, x_{n+1}\}$;
$x_j \leftarrow x_{n+1}$; $\sigma_p \leftarrow \emptyset$;
while 1 **do**
  Add x_j to σ_p;
  $X_{candidate} \leftarrow \emptyset$;
  for i=0 **to** i=j-1 **do**
    if CollisionFree(x_i, x_j) **then**
      Add x_i to $X_{candidate}$;
    end if
  end for
  $x_{min} \leftarrow x_k \in X_{candidate}$;
  $J_{min} \leftarrow (\frac{u_k \cdot \overrightarrow{p_k p_j}}{\|\overrightarrow{p_k p_j}\|} + \frac{u_j \cdot \overrightarrow{p_k p_j}}{\|\overrightarrow{p_k p_j}\|})$;
  for each $x_{k'} \in X_{candidate} \setminus \{x_k\}$ **do**
    $J \leftarrow (\frac{u_{k'} \cdot \overrightarrow{p_{k'} p_j}}{\|\overrightarrow{p_{k'} p_j}\|} + \frac{u_j \cdot \overrightarrow{p_{k'} p_j}}{\|\overrightarrow{p_{k'} p_j}\|})$;
    if $J < J_{min}$ **then**
      $J_{min} \leftarrow J$, $x_{min} \leftarrow x_{k'}$;
    end if
  end for
  $x_j \leftarrow x_{min}$;
  if $x_j == x_0$ **then**
    Add x_0 to σ_p;
    Break while loop;
  end if
end while
return σ_p;

where $\beta = \frac{\gamma}{2}$ and γ is the angle between vector $\overrightarrow{x_{parent} x_1}$ and $\overrightarrow{x_1 x_2}$. The parameters q_1, q_2, q_3, q_4 are

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

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,

$$L = \min\left\{\frac{\min\{\|\overrightarrow{p_{parent} p_1}\|, \|\overrightarrow{p_1 p_2}\|\}}{2}, \frac{\eta}{2}\right\}. \tag{7}$$

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 $\min\{\min\{\|\overrightarrow{p_{parent} p_1}\|, \|\overrightarrow{p_1 p_2}\|\}, \eta\}$. 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 ψ_{init} and one waypoint in the yaw direction ψ_{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.

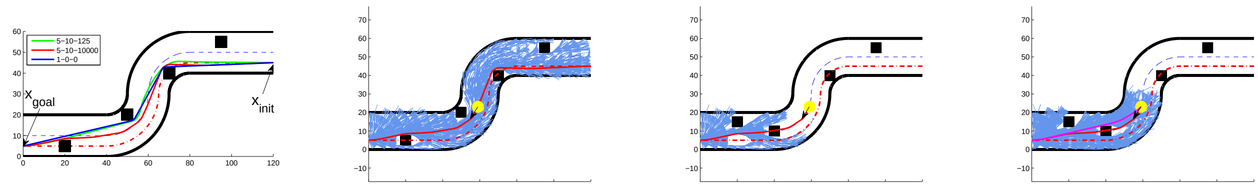
Algorithm 2 Replanning

Input: $\sigma = \{x_0, x_1, x_2, \dots, x_n, x_{n+1}\}$, $T = (V, E)$, X_{obs};
for each $e \in E$ **do**
  Let the two endpoints of e be: $x_p^e \rightarrow x_c^e$;
  if not CollisionFree(x_p^e, x_c^e) **then**
    Mark x_c^e as INVALID;
  end if
end for
if σ does NOT have any INVALID nodes **then**
  return σ;
else
  $V_{valid} = \emptyset$;
  for each node $x_i \in V$ **do**
    x_p = Parent(x_i);
    if x_p is INVALID **then**
      Mark x_i as INVALID;
    end if
    if x_i is VALID **then**
      $V_{valid} \leftarrow x_i$;
    end if
  end for
  Delete all the INVALID nodes from the tree;
  Apply RRT* to regrow the tree;
  Return a new path σ_{new};
end if

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

Algorithm 3 Path Planning for Semi-Autonomous Driving

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



(a) Legend denotes the weights $w_1 - w_2 - w_3$ in the cost.

(b) Current vehicle position before obstacle moves.

(c) Obstacle moves and tree is trimmed.

(d) The magenta curve is the replanned path.

Fig. 3. Black blocks: obstacles. Dashed red curve: reference path. Solid curve: planned smooth path. Blue paths: random tree. Yellow dot: vehicle.

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 $x_{init} = (120, 45, 0)$ and $x_{goal} = (0, 5, 0)$, respectively. The tree is rooted at $x_{root} = x_{goal}$. For the primitive procedures, we choose $\eta = 5$ for the Steer function. The parameter γ in the function Near is chosen as $\gamma = (2(1 + 1/2)^{1/2} \mu(X_{free})/\zeta_2)^{1/2} + 1$. In the second stage of sampling, δ is set to $\frac{\pi}{4}$. In the stopping criterion, Ψ is chosen as $\frac{\pi}{6}$. Figure 3(a) shows the path returned by our algorithm with different weights in the cost function. The blue curve is the smoothed path under the cost which only penalizes the Euclidean distance. It gives the shortest distance curve between x_{init} and x_{goal}. The red curve emphasizes the difference between the curvature of the RRT* path and the one of the reference path. From Figure 3(a) we see that it is significantly closer to the reference path in terms of curvature.

When the vehicle starts moving along the path returned by our algorithm (see Figure 3(b)), if some obstacles move to another position 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 G^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

[1] S. Zafeiropoulos and S. Di Cairano, “Vehicle yaw dynamics control by torque-based assist systems enforcing drivers steering feel constraints,” in *Proc. American Control Conference*, pp. 6746–6751, 2013.

[2] S. Zafeiropoulos and S. Di Cairano, “Governor-based control for rack-wheel coordination in mechanically decoupled steering systems,” in *Proc. 53rd IEEE Conf. Decision and Control*, pp. 4089–4094, 2014.

[3] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” *Int. Jour. Robotics Research*, vol. 30, pp. 846–894, 2011.

[4] S. M. LaValle, “Rapidly-exploring random trees: A new tool for path planning.” TR 98-11, Computer Science Dept., Iowa State University, October 1998.

[5] Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli, and J. P. How, “Real-time motion planning with applications to autonomous urban driving,” *IEEE Trans. Control Systems Technology*, vol. 17, no. 5, pp. 1105–1118, 2009.

[6] J. H. Jeon, S. Karaman, and E. Frazzoli, “Anytime computation of time-optimal off-road vehicle maneuvers using the rrt*,” in *50th IEEE Conf. Decision and Control and European Control Conference*, pp. 3276–3282, 2011.

[7] J. H. Jeon, R. V. Cowlagi, S. C. Peters, S. Karaman, E. Frazzoli, P. Tsiotras, and K. Iagnemma, “Optimal motion planning with the half-car dynamical model for autonomous high-speed driving,” in *Proc. American Control Conference*, pp. 188–193, 2013.

[8] S. Di Cairano, H. Tseng, D. Bernardini, and A. Bemporad, “Vehicle yaw stability control by coordinated active front steering and differential braking in the tire sideslip angles domain,” *IEEE Trans. Control Sys. Technology*, vol. 21, no. 4, pp. 1236–1248, 2013.

[9] A. Gray, Y. Gao, T. Lin, J. K. Hedrick, H. E. Tseng, and F. Borrelli, “Predictive control for agile semi-autonomous ground vehicles using motion primitives,” in *Proc. American Control Conference*, pp. 4239–4244, 2012.

[10] J. V. Frasch, A. Gray, M. Zanon, H. J. Ferreau, S. Sager, F. Borrelli, and M. Diehl, “An auto-generated nonlinear mpc algorithm for real-time obstacle avoidance of ground vehicles,” in *Proc. European Control Conference*, pp. 4136–4141, 2013.

[11] K. Yang and S. Sukkarieh, “An analytical continuous-curvature path-smoothing algorithm,” *IEEE Trans. Robotics*, vol. 26, no. 3, pp. 561–568, 2010.

[12] D. Ferguson, N. Kalra, and A. Stentz, “Replanning with rrts,” in *IEEE Int. Conf. Robotics and Automation*, pp. 1243–1248, 2006.

[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. The MIT Press, 2001.

[14] R. V. Cowlagi, S. C. Peters, S. Karaman, E. Frazzoli, P. Tsiotras, K. Iagnemma, *et al.*, “Optimal motion planning with the half-car dynamical model for autonomous high-speed driving,” in *Proc. American Control Conference*, pp. 188–193, 2013.