Motion Planning for Aggressive Autonomous Vehicle Maneuvers

Aliasghar Arab, Kaiyan Yu, Jingang Yi, and Dezhen Song

Abstract— **We present a motion planning algorithm for autonomous aggressive vehicle maneuvers. The motion planner takes advantages of the sparse stable trees (SST), the RRT* algorithm and the model predictive control (MPC) design. The use of the sparsity property helps to reduce the computational burden of the RRT* method by removing non-useful nodes in each iteration (i.e., rewiring) and therefore to quickly converge to the optimal solution. A goal-biased input is used to achieve a fast convergence. A nonlinear MPC is used to compute and find the attracting area for the generated trajectory with consideration of constraints of the system. We implement and demonstrate the motion planning algorithm on a 1/7-scale racing vehicle for autonomous aggressive maneuvers.**

I. Introduction

Aggressive vehicle maneuvers are commonly used by professional racing car drivers to achieve fast and agile performance. Understanding these skilled maneuvers can help to design autonomous driving capability and active safety features under extremely events, such as emergency maneuvers [1], [2]. Although various motion control strategies are proposed (e.g., [2]–[4]), motion planning for for these aggressive vehicle maneuvers is critical and still a challenging task.

It is challenging to design a time-optimal trajectory with a feasible initial condition [5]–[7]. Sampling-based motion planning algorithms, such as rapidly exploring random trees (RRT) [8] and sampling-based model predictive (SBMPC) [9], efficiently find feasible trajectories over complex dynamics, constraints and obstacles. These algorithms have demonstrated the ability to produce feasible open-loop trajectories but they cannot guarantee the optimality of the produced trajectory. The development of the RRT* algorithm guarantees the asymptotically optimality [10] through the nearest and rewiring strategies in the tree node expansion process. However, since the boundary-value problem of complex dynamics is difficult to solve, the rewiring process of the RRT* algorithm cannot be implemented accurately.

Recently, a modified RRT* algorithm is implemented to find a minimum-time trajectory for autonomous high-speed driving of vehicles [11]. To achieve a fast computation, a simple particle model is used to transform the steering problem for the half-car model [12]. The sparse-RRT method does not use the rewiring process and instead provides asymptotic near-optimality for planning without access to a steering function [13]. Motion primitives are used to generate feasible, fast trajectory for vehicle maneuvers and optimality of the trajectory is however not guaranteed [14]. Randomized motion planning algorithms are also proposed to generate open-loop sequences of minimum-time motions for successful simulation [11], [12], [15]. Closed-loop RRT-based motion planning algorithm is implemented in [16] for real-time autonomous vehicles but the optimality of the trajectory is not considered and guaranteed. Feedback-based motion planning algorithms are developed in [17] with guaranteed control stability and safety.

In this paper, we present a motion planning approach that takes the advantages of the local steering algorithm [12] and the sparse search method with integration of a nonlinear MPC (NMPC) [15]. One attractive property of the proposed motion planner is fast computation for real-time applications. Indeed, we implement and demonstrate the motion planning algorithm using a 1/7-scale autonomous vehicle. We also compare the performance under the autonomous driving control and the expert human driver. The main contribution of this work lies in the design and implementation of a reliable motion planning scheme for autonomous vehicle aggressive maneuvers. The proposed sparse-based motion planner is adopted from [15] and the use of the MPC for the steering function is inspired from [12]. We extend the work in [15] to take advantage of the steering function and that the rewiring process is used only for the dynamically reachable nodes to reduce the computational cost. The combined feedback linearization and NMPC design is used as the lower-level controller for robust motion performance [4]. The main focus of this paper lies in motion planning algorithms of the autonomous aggressive vehicle maneuvers while the companion paper [4] deals with the motion control of the vehicle to follow a given desired trajectory.

The remainder of the paper is organized as follows. We present the vehicle dynamics and motion control in Section II. The main motion planning algorithm is presented in Section III. Section IV presents the experimental results. Finally, we summarize the concluding remarks and future research directions in Section V.

II. Vehicle Dynamics and Motion Control

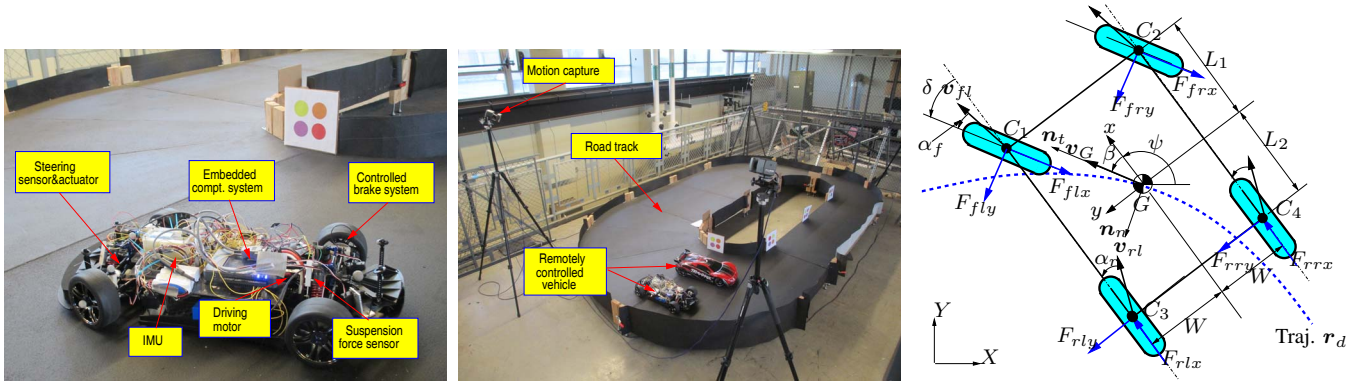
In this section, we present the dynamics model and motion control for aggressive maneuvers. The main part of this section has been discussed in [4] and we briefly present here for self-contain of the paper.

The scaled vehicle prototype is shown Fig. 1(a) and Fig. 1(b) shows the indoor vehicle track for experiments.

The work was supported in part by the US NSF under awards CMMI-0954966 and CMMI-1334389.

A. Arab, K. Yu and J. Yi are with the Department of Mechanical and Aerospace Engineering, Rutgers University, Piscataway, NJ 08854 USA (email: alias.arb@rutgers.edu; kaiyan.yu@rutgers.edu; jgyi@rutgers.edu).

D. Song is with the Department of Computer Science and Engineering, Texas A&M University, College Station, TX 77843 USA (email: dzsong@tamu.edu).



(a) (b) (c)

Fig. 1. (a) Autonomous scaled vehicle. (b) The indoor robotic vehicle testing track. (c) A schematic of the robotic vehicle motion.

Fig. 1(c) illustrates the schematic of the vehicle kinematics. The vehicle planar motion is captured by two frames: an initial frame $\mathcal{N}(X, Y)$ and a body-fixed frame $\mathcal{B}(x, y)$. The origin of $\mathcal{B}$ is located at the mass center G (also assumed as the geometric center of the vehicle chassis). The vehicle pose is denoted as $\boldsymbol{q} = [x\ y\ \psi]^T$ in $\mathcal{N}$, where $[x\ y]^T$ is the position vector of G and ψ is the yaw angle. The velocity vector is $\boldsymbol{v} = [v_x\ v_y\ \dot{\psi}]^T$. We denote the distances from G to the front and the rear wheels as L_1 and L_2, respectively and $L = L_1 + L_2$ is then the wheel base. We also denote the vehicle width as $2W$ and the mass and mass moment of inertia about the z-axis in $\mathcal{B}$ as m and I_z, respectively.

From [4], we obtain the equations of motion of the vehicle

$$\boldsymbol{M}\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{B}_x(\delta)\boldsymbol{F}_x + \boldsymbol{B}_y(\delta)\boldsymbol{F}_y, \tag{1}$$

where $\boldsymbol{F}_x = [F_{flx}\ \ F_{frx}\ \ F_{rlx}\ \ F_{rrx}]^T$, $\boldsymbol{F}_y = [F_{fly}\ F_{fry}\ F_{rly}\ F_{rry}]^T$, F_{ijx} and F_{ijy}, $i = f, r$, $j = l, r$, are the longitudinal and lateral forces at the front (rear), left (right) tires, respectively. Matrices $\boldsymbol{M}$, $\boldsymbol{C}$, $\boldsymbol{B}_x$ and $\boldsymbol{B}_y$ are given in [4] and we omit the details here. Input δ is the steering angle. Defining the state variable $\boldsymbol{z} = [\boldsymbol{q}^T\ \dot{\boldsymbol{q}}^T]^T$ and inputs $\boldsymbol{u} = [\delta\ \boldsymbol{F}_x]^T$, we can rewrite the dynamics model (1) as

$$\dot{\boldsymbol{z}} = \boldsymbol{f}(\boldsymbol{z}, \boldsymbol{u}). \tag{2}$$

To capture the tire-road interaction forces, we use the Pacejka's magic formula [18] and the model parameters are obtained through fitting experimental data. The vehicle center slip angle is $\beta = \tan^{-1}(v_y/v_x)$.

The motion control is built on a feedback linearization and an NMPC design. The former computes the desired tire friction forces and the NMPC tries to regulate the steering angle and tire forces to follow the desired profiles [4].

For the feedback linearization design, we consider to track the center point between C_1 and C_2 (in Fig. 1(c)) with position vector $\boldsymbol{r}_P = [x + L_1 c_\psi\ \ y - L_1 s_\psi]^T$. Taking the time derivative twice of $\boldsymbol{r}_P$ leads to

$$\ddot{\boldsymbol{r}}_P = \boldsymbol{\Lambda}\boldsymbol{M}^{-1}(\boldsymbol{B}_x\boldsymbol{F}_x + \boldsymbol{B}_y\boldsymbol{F}_y - \boldsymbol{C}\boldsymbol{v}) + \dot{\boldsymbol{\Lambda}}\boldsymbol{v}, \tag{3}$$

where

$$\boldsymbol{\Lambda} = \begin{bmatrix} \mathrm{c}_\psi & -\mathrm{s}_\psi & -L_1\,\mathrm{s}_\psi \\ \mathrm{s}_\psi & \mathrm{c}_\psi & L_1\,\mathrm{c}_\psi \end{bmatrix}$$

The feedback linearization control of force $\boldsymbol{F}_x$ is designed as $\boldsymbol{F}_x = \boldsymbol{\Gamma}^{-1}[\ddot{\boldsymbol{r}}_d + \boldsymbol{K}_d(\dot{\boldsymbol{r}}_d - \dot{\boldsymbol{r}}_P) + \boldsymbol{K}_p(\boldsymbol{r}_d - \boldsymbol{r}_P)] + \hat{\boldsymbol{\vartheta}}$, where $\boldsymbol{\Gamma} = \boldsymbol{\Lambda}\boldsymbol{M}^{-1}\boldsymbol{B}_x$, $\boldsymbol{\vartheta} = \boldsymbol{\Lambda}\boldsymbol{M}^{-1}\boldsymbol{B}_y\boldsymbol{F}_y + \dot{\boldsymbol{\Lambda}}\boldsymbol{v}$, and $\boldsymbol{K}_p$ and $\boldsymbol{K}_d$ are positive diagonal matrices. Defining $\boldsymbol{e} = \boldsymbol{r}_d - \boldsymbol{r}_P$, the closed-loop error dynamics $\ddot{\boldsymbol{e}} + \boldsymbol{K}_d\dot{\boldsymbol{e}} + \boldsymbol{K}_p\boldsymbol{e} = \hat{\boldsymbol{\vartheta}} - \boldsymbol{\vartheta}$ is stable for positive diagonal matrices $\boldsymbol{K}_p$ and $\boldsymbol{K}_d$.

For a given desired trajectory, an NMPC is used to design a steering controller to follow the path. We formulate the NMPC control problem in discretized system dynamics of (2) as

$$\begin{aligned} \min_{\Delta u(t)} \quad & \sum_{i=k}^{k+H_p} (l_1\|\boldsymbol{e}_i\|^2 + l_2\|\dot{\boldsymbol{e}}_i\|^2 + l_3\|\Delta\boldsymbol{u}_i\|^2) \\ \text{subject to} \quad & \boldsymbol{z}_{k+1,t} = \boldsymbol{f}(\boldsymbol{z}_{k,t}, \boldsymbol{u}_{k,t}) \\ & u_{\min} \le |\boldsymbol{u}_{k,t}| \le u_{\max}, \Delta u_{\min} \le |\Delta\boldsymbol{u}_{k,t}| \le \Delta u_{\max} \\ & \boldsymbol{u}_{k+1,t} = \Delta\boldsymbol{u}_{k+1,t} + \boldsymbol{u}_{k,t},\ k = t, \ldots, t + H_p, \end{aligned}$$

where l_j, $j = 1, \cdots, 3$, are constant weights, $\boldsymbol{z}(t) = [\boldsymbol{z}_{t,t}\ \boldsymbol{z}_{t+1,t}\ \cdots \boldsymbol{z}_{t+H_p,t}]$ with $\boldsymbol{z}_{t,t} = \boldsymbol{z}(t)$ is the sequence of states $\boldsymbol{z}(t)$ over the prediction horizon H_p at time t. Terms $\boldsymbol{u}_{k,t}$ and $\Delta\boldsymbol{u}_{k,t}$ are the kth input sequence $\boldsymbol{u}_t$ and $\Delta\boldsymbol{u}_t$ respectively.

III. Optimal Kinodynamic Motion Planning

Let compact sets $\boldsymbol{Z} \subset \mathbb{R}^n$ and $\boldsymbol{U} \subset \mathbb{R}^m$ denote the trajectories and the controls for system (2), respectively. Let $\boldsymbol{Z}_o \subset \boldsymbol{Z}$ and $\boldsymbol{Z}_g \subset \boldsymbol{Z}$ denote the obstacle and goal regions, respectively. The feasible region is then $\boldsymbol{Z}_f := \boldsymbol{Z}/\boldsymbol{Z}_o$. The vehicle is first located at $\boldsymbol{z}_0 \in \boldsymbol{Z}$ and must maintain inside feasible region $\boldsymbol{Z}_f$ and reach $\boldsymbol{Z}_g$. A feasible trajectory $\boldsymbol{\omega} \in \boldsymbol{Z}_f$ over duration T is a set of dynamic states of the vehicle system over $[0, T]$ which connect $\boldsymbol{z}_0$ to $\boldsymbol{Z}_g$ with corresponding controls $\boldsymbol{u}$. $\boldsymbol{\Omega}$ is the set of all feasible trajectories in $\boldsymbol{Z}_f$. The motion planning for given $\{\boldsymbol{Z}_f, \boldsymbol{U}, \boldsymbol{z}_0, \boldsymbol{Z}_g\}$ is to find trajectory $\boldsymbol{\omega} \in \boldsymbol{\Omega}$ while minimizing a cost function (i.e., traveling time). The cost function $c \neq 0$ for trajectory $\boldsymbol{z} \in \boldsymbol{Z}$ is defined as

$$J_c(\boldsymbol{z}) = \int_0^T c(\boldsymbol{z}(t))dt.$$

The RRT* algorithm [10] is a sampling-based motion planner that guarantees the optimality. The main approach of

RRT* is the use of the `Nearest` and `Rewire` procedures that need `Steering` function to connect two different nodes in the configuration space. In the Sparse-RRT [15] algorithm, the `Nearest` function is used but `Rewire` is not used because of the difficulty to solve boundary-value problem (BVP). A `Drain` function is also used to remove the nodes with higher costs in neighboring sets and keeps the lowest-cost node. In our approach, we instead use the NMPC controller to connect the nodes.

We extend the RRT* algorithm described in [11] and take modifications to add the attractive properties of the Sparse-RRT [15] as a `Drain` function. Unlike the approach in [12] to connect the nodes in rewiring process, the NMPC design is used to connect the nodes for the vehicle dynamic model. The Sparse-RRT* is illustrated in Algorithm 1. Fig. 2 also illustrates the construction of the `BestNearest`, `Drain` and `Rewire` processes in the Sparse-RRT* design and these processes are described in the following discussions.

Algorithm 1: Sparse-RRT*$(\boldsymbol{Z}_g, \boldsymbol{U}, \boldsymbol{z}_0, \boldsymbol{Z}_f, i_{\max})$

$\mathbb{V} \leftarrow \boldsymbol{z}_0$, $\mathbb{E} \leftarrow \emptyset$, $i \leftarrow 0$;
for $i < i_{max}$ **do**
  $\boldsymbol{z}_{\text{rand}} \leftarrow$ `Sample`$(\boldsymbol{Z}_f, \boldsymbol{Z}_g)$;
  $\boldsymbol{z}_{\text{near}} \leftarrow$ `BestNearest`$(\mathbb{V}, \boldsymbol{z}_{\text{rand}}, \Delta_{\text{near}})$;
  $\boldsymbol{z}_{\text{new}} \leftarrow$ `Steering`$(z_{\text{near}}, z_{\text{rand}}, \boldsymbol{Z}_f)$;
  if $\boldsymbol{z}_{new} \neq \emptyset$ **then**
    $\mathbb{V} \leftarrow \mathbb{V} \cup \{\boldsymbol{z}_{\text{new}}\}$, $\mathbb{E} \leftarrow \mathbb{E} \cup \{(\boldsymbol{z}_{\text{near}}, \boldsymbol{z}_{\text{new}})\}$;
    $\mathbb{V}, \mathbb{E} \leftarrow$ `Rewire`$(\mathbb{V}, \boldsymbol{z}_{\text{near}}, \boldsymbol{z}_{\text{new}})$;
    $\mathbb{V}, \mathbb{E} \leftarrow$ `Drain`$(\mathbb{V}, \boldsymbol{z}_{\text{new}}, \Delta_{\text{Drain}})$;
  $i \leftarrow i + 1$;

`Sample`: Inspired from the MPC-tree method [9], we take the goal-directed samples in the steering function design. The main idea of the goal-directed sampling process is to take a small percentage (e.g., 5%) of the samples biased toward the goal. Such process would help to quickly converge to the goal. The sampled point is presented in Fig. 2 with an orange circle as $\boldsymbol{z}_{\text{rand}}$.

`BestNearest`: The main concept of the `BestNearest` function is illustrated in Fig. 2. The circle around $\boldsymbol{z}_{\text{rand}}$ with a radius of Δ_{Near} defines the area that the algorithm searches for the neighbors. Similar to the RRT* [12] and Sparse-RRT [13] algorithms, the `BestNearest` function in Algorithm 1 tries to find a node with the minimum cost within a vicinity Δ_{near} of $\boldsymbol{z}_{\text{new}}$. The vicinity is limited only to the nodes that are dynamically feasible to be steered from $\boldsymbol{z}_{\text{near}}$ to $\boldsymbol{z}_{\text{new}}$. The `BestNearest` function is illustrated in Algorithm 3. The `Near` function returns all the nodes in $\mathbb{V}$ that are in vicinity of $\boldsymbol{z}_{\text{rand}}$ within a radius of Δ_{near}. The nodes in $\mathbb{Z}_{\text{Near}}$ are restricted to those that are dynamically feasible through `FeasibleNear` function in the algorithm.

`Steering`: Unlike random propagation in [13], in the Sparse-RRT* algorithm, the nearest node $\boldsymbol{z}_{\text{Near}}$ is driven toward $\boldsymbol{z}_{\text{rand}}$ (usually close to $\boldsymbol{z}_{\text{new}}$) using also the `Steering` function, as illustrated in Fig. 2. The advantage of using the `Steering` function is that when the system has an obstacle-free path to $\boldsymbol{Z}_g$, the goal-directed sampling helps a fast convergence to obtain the initial trajectory for real-time applications.

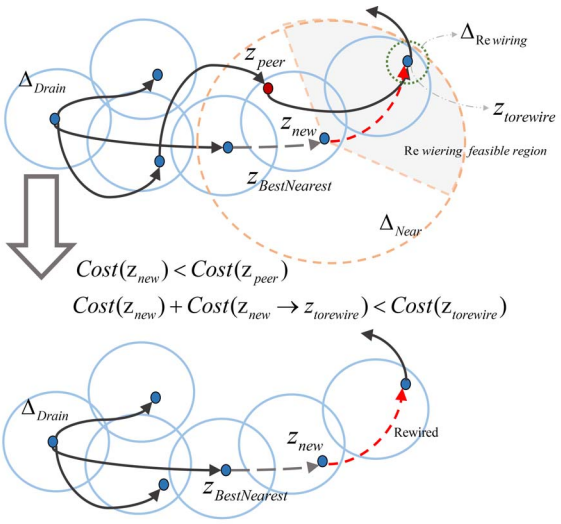


Fig. 2. Drain and Rewiring method after generating a new node is graphically presented in a selected picture.

Algorithm 2: `BestNearest`$(\mathbb{V}, \boldsymbol{z}_{\text{rand}}, \Delta_{\text{near}})$

$\mathbb{Z}_{\text{Near}} \leftarrow$ `Near`$(\mathbb{V}, \boldsymbol{z}_{\text{rand}}, \Delta_{\text{near}})$;
$\mathbb{Z}_{\text{Near}} \leftarrow$ `FeasibleNear`$(\mathbb{Z}_{\text{Near}}, \boldsymbol{z}_{\text{rand}})$;
if $\mathbb{Z}_{Near} = \emptyset$ **then Return** : `Nearest`$(\mathbb{V}, \boldsymbol{z}_{\text{rand}})$;
else Return : $\text{argmin}_{\boldsymbol{z} \in \mathbb{Z}_{\text{Near}}} c(\boldsymbol{z})$

`Rewire`: To find the nodes $\boldsymbol{z}_{\text{torewire}}$ that need to be rewired, the vicinity of $\boldsymbol{z}_{\text{new}}$ within a radius of Δ_{near} is used to restrict to the nodes that are dynamically feasible, as the gray circle area illustrated in Fig. 2. We also use a vicinity around $\boldsymbol{z}_{\text{new}}$ within a radius of Δ_{rewire} as the converging area from node $\boldsymbol{z}_{\text{torewire}}$. The main difference and advantage of the Sparse-RRT* algorithm with the Sparse-RRT in [13] is the added `Rewire` function. The reason of not using the `Rewire` function in [13] lies in difficulty to solve the BVP. At each iteration, after sampling a new state and extending the tree towards the new state, the `Rewire` function attempts to re-assign the parent of each nearby node. To reduce the number of the attempts, the vicinity of the nearby nodes is restricted to the ranges of the NMPC can steer to. In the RRT* algorithm [12], the number of times to invoke the `Steering` function is $O(n \log n)$ and the collision check is implemented after the rewiring by the `Steering` function. To reduce the computational burden, the collision check is enforced while propagating. We also use the closed-loop NMPC to steer the system from one node to another with its inherent optimization formulation.

`Drain`: The main idea of the `Drain` function is to divide the search space to subsections and then use search methods to connect the subsections and find the trajectory.

Comparing with other algorithms, the main difference in this work is that the subsections are chosen dynamically while the tree is growing. Unlike [15], the drain region Δ_{Drain} is selected regarding to the attraction region of the dynamic system. In Fig. 2, drain region Δ_{Drain} is illustrated as the blue circles where all the nodes in the vicinity of $\boldsymbol{z}_{\text{new}}$ within Δ_{Drain} are called $\boldsymbol{z}_{\text{peer}}$. The `Drain` function is illustrated in Algorithm 3. In algorithm 3, z.`children` is the number of the children of node $\boldsymbol{z}$ and $\mathbb{Z}_{\text{Useless}}$ is set of the nodes which have higher cost than the best reached node and $\mathbb{Z}_{\text{Reached}}$ is the set of the nodes that reach the goal.

Algorithm 3: $\texttt{Drain}(\mathbb{V}, \boldsymbol{z}_{\text{new}}, \Delta_{\text{Drain}})$

$\mathbb{Z}_{\text{Drain}} \leftarrow \texttt{Near}(\mathbb{V}, \boldsymbol{z}_{\text{new}}, \Delta_{\text{Drain}})$;
if $c(\boldsymbol{z}_{new}) \geq \text{argmin}_{\boldsymbol{z} \in \mathbb{Z}_{Drain}} c(\boldsymbol{z})$ **then** remove($\boldsymbol{z}_{\text{new}}$);
else
  for $\boldsymbol{z} \in \mathbb{Z}_{Drain}$ **do**
    if $\boldsymbol{z}$.`children` $= 0$ **then** remove($\boldsymbol{z}$);
if $\mathbb{Z}_{Reached} \neq \emptyset$ **then**
$\mathbb{Z}_{\text{Useless}} \leftarrow \{\boldsymbol{z} \in \mathbb{V} \,|c(\boldsymbol{z}) \geq \text{argmin}_{\boldsymbol{z} \in \mathbb{Z}_{\text{Reached}}} c(\boldsymbol{z})\}$;
for $\boldsymbol{z} \in \mathbb{Z}_{Useless}$ **do** remove($\boldsymbol{z}$);

IV. Simulation and Experimental Results

A. Simulation Results

We conduct simulation comparison of the running time, number of the nodes and cost function convergence among the RRT*, SST and Sparse-RRT* (SRRT*) algorithms. Fig. 3 shows the comparison results (average of 20 random runs) of these algorithms: the running time (Fig. 3(a)), the number of the nodes (Fig. 3(b)) and the cost (i.e., traveling time) (Fig. 3(c)). For the computation load, the Sparse-RRT* performs slightly better than the SST and the RRT*. The number of the nodes for the RRT* is increasing linearly with the iterations while for the SST and the Sparse-RRT* algorithms, they reach almost constants after certain iterations due to the sparse properties used in the algorithms. The costs of the Sparse-RRT* and the RRT* finally maintain at the same lower-level, smaller than that of the SST. All of these results demonstrate the attractive properties of the Sparse-RRT* algorithm.

Table I shows the simulation results under different algorithms for the U-turn vehicle maneuver. Similarly, the number of the nodes for the Sparse-RRT* is less than the other two methods. The cost function converges similar to that of the RRT* while the traveling time under the SST is much larger than the other algorithms. Similar to that shown in Fig. 3(a), the Sparse-RRT* algorithm is faster than both the other two methods.

TABLE I
Simulation results under three different motion planning algorithms (50000 iterations)

	Method	Nodes	Time	Cost
U-turn (Time)	RRT*	27350	412	2.87
	SST	6105	323	3.18
	Sparse-RRT*	4573	291	2.93

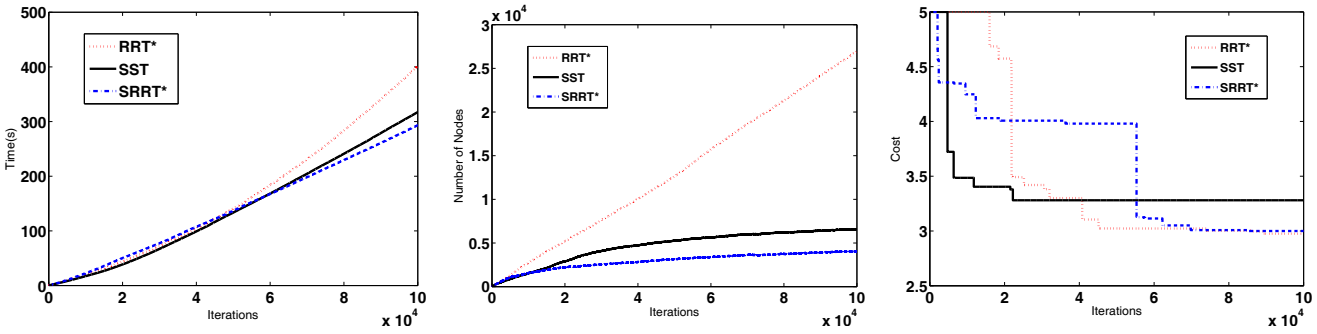
B. Experimental Results

We built a 1/7-scale vehicle (Traxxas XO-1 model) for the experiments as shown in Fig. 1(a). The vehicle is modified by adding various onboard sensors and actuators [4]. In our experiments, the real-time position information of the vehicle is obtained through the optical motion capture systems (8 Bonita cameras from Vicon Inc.) An embedded real-time system (myRIO from National Instruments) is installed on the vehicle to collect and sample all onboard sensors and control the actuators at a frequency of 1 kHz. The motion capture system is synchronized with the onboard embedded system through the wireless networks connected by a laptop computer. A track is built and used for the experiments as shown in Fig. 1(b). The shape and geometry (i.e., straight-line and U-turn shapes) of the track are specially designed to test the aggressive vehicle maneuvers.

The time-optimal trajectory computation by the Sparse-RRT* is not fast enough for online motion planning. In our experiments, the optimal motion planning is first computed off-line and the motion controller is implemented to follow the trajectory. The NMPC is implemented to follow the minimum time trajectory computed by the Sparse-RRT* off-line. Fig. 4 shows the time-optimal trajectory generated by the Sparse-RRT* algorithm (i.e., the solid line). The blue-dot line in the figure indicates the NMPC tracking results after the optimal trajectory is generated by Sparse-RRT*. The main reason of conducting the NMPC in simulation is to check that the small tolerances in the rewiring process do not produce any problems.

For comparison purpose, we also invited an expert RC vehicle driver to the campus and conducted driving tests. The human expert driving tests were conducted for several times and all motion data were recorded. Taking the same starting and the ending points, we conducted the autonomous driving tests. The mathematical model of the vehicle is first obtained and the values of the vehicle parameters are listed in Table II. Then the motion controller is implemented to track the optimal trajectory generated by the motion planner.

Fig. 5 shows the comparison results of the motion planning under the human expert driving and the autonomous control. The Sparse-RRT* is used to generate the time-optimal trajectory for the autonomous driving test $\boldsymbol{r}_d$ as shown in the figure. Under the motion control design, the vehicle tries to follow the desired trajectory with the shortest time. As shown in Fig. 5, the NMPC tries to minimize the tracking error and the trajectories under the motion controller and the human driver do not follow the motion planner result. Furthermore, we also compute and list the traveling times under these two cases in Table III. For comparison purpose, we list the maneuver agility metrics in the table. These agility metrics include the accumulated lateral jerk and the accumulated relative lateral acceleration of the vehicle [1]. Both metrics listed in the table are calculated over the traveling distances.



(a) (b) (c)

Fig. 3. Comparing the number of the nodes, computational time and convergence of the cost function for vehicle motion planning. (a) The elapsed time of different methods for the car mode. (b) Number of the generated nodes for the car model. (c) Convergence of the cost function (Time) for the car model.

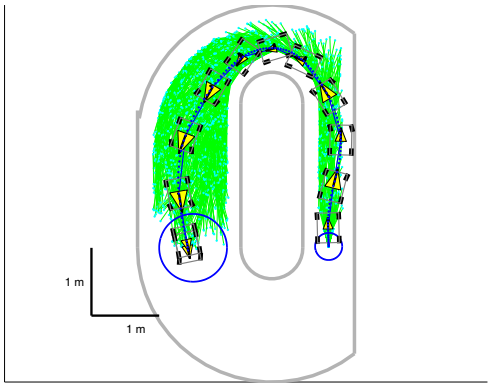


Fig. 4. The time-optimal trajectory using the Sparse-RRT*. The results are obtained under $500,000$ iterations, $2,573$ nodes, $1,570$ rewiring and $601,260$ of drained nodes in $1,063.97$ seconds. ($\Delta_{\text{Near}} = 0.5$ and $\Delta_{\text{Drain}} = 0.1$). The optimal traveling time is found as 2.93 s.

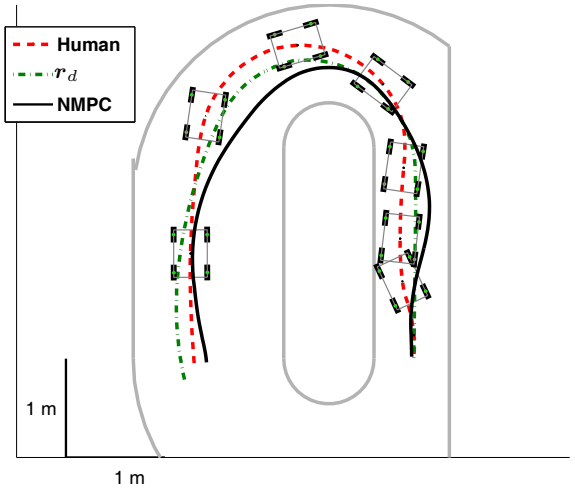


Fig. 5. Trajectory comparison under two different autonomous control designs and human test.

It is clear from those results that the autonomous controller achieves the shorter traveling time and higher agility than those by the human expert driver.

Fig. 6 further shows the vehicle motion comparisons under the control design and human's test. Fig. 6(a) shows the longitudinal and lateral velocities, Fig. 6(b) demonstrates

TABLE II

THE VALUES OF THE TESTING VEHICLE PARAMETERS

m (kg)	L_1 (m)	L_2 (m)	W (m)	h (m)	I_z (kg m^2)
6.0	0.2	0.2	0.15	0.05	0.25

TABLE III

TRACKING PERFORMANCE COMPARISON UNDER TWO CONTROLLERS AND THE HUMAN DRIVER

Controller	Traveling time (s)	Agility metric 1 (m/s^3)	Agility metric 2
Proposed	3.18	5.31	0.58
Human	3.26	7.36	0.65

the yaw-rate ($\dot{\psi}$) comparisons, Fig. 6(c) shows the slip angle comparisons and finally, Fig. 6(d) shows the steering angle comparison. It is clearly shown in these figures that under the NMPC, the vehicle runs smoothly without quickly changing of the steering and slip angles, which is observed by the performance of the human expert driver. Both the autonomous and human expert driving show large slip angles (around more than 10 degs). This observation is similar to the comparison between the professional racing drivers and the typical human drivers presented in [1].

V. CONCLUSION AND FUTURE WORK

We presented the Sparse-RRT*-based motion planning algorithms for autonomous aggressive vehicle maneuvers. The Sparse-RRT* motion planner took advantages of the Sparse-RRT and the RRT* algorithms. The advantage of sparse property for motion planning helped to reduce the computational burden by removing un-useful nodes in the searching process. The attractive property of the RRT* lies in fast convergence to the optimal solution. We implemented and tested the motion planner using a 1/7-scale autonomous vehicle. Comparison with human expert driver was presented in the paper. The experimental results have demonstrated the high agility maneuvering performance under the autonomous driving control with the motion planner. We are currently extending the control design to conduct more tests on different surface conditions. Integration of the motion planner into the real-time control system design is another ongoing research effort.

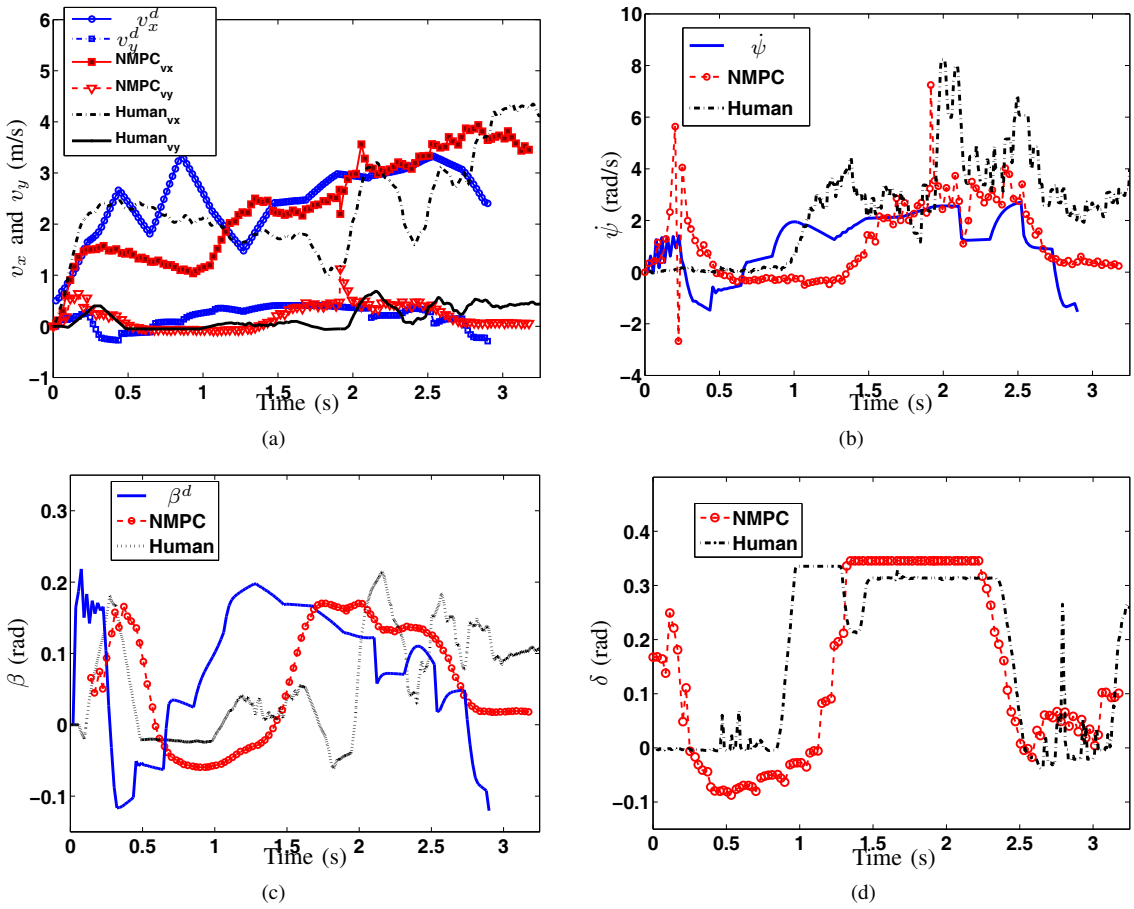


Fig. 6. Experimental comparison results under the autonomous controller and the human expert driver. (a) Vehicle velocity profiles. (b) Yaw-Rate ($\dot{\psi}$) profiles. (c) The body center slip angle (β) profiles. (d) Steering angle (δ) profiles.

REFERENCES

[1] J. Yi, J. Li, J. Lu, and Z. Liu, "On the dynamic stability and agility of aggressive vehicle maneuvers: A pendulum-turn maneuver example," *IEEE Trans. Contr. Syst. Technol.*, vol. 20, no. 3, pp. 663–676, 2012.

[2] R. Y. Hindiyeh, "Dynamics and control of drifting in automobiles," Ph.D. dissertation, Dept. Mech. Eng., Stanford Univ., Stanford, CA, 2013.

[3] D. Tavernini, E. Velenis, R. Lot, and M. Massaro, "The Optimality of the Handbrake Cornering Technique," *ASME J. Dyn. Syst., Meas., Control*, vol. 136, no. 4, pp. 1–11, 2014.

[4] A. Arab, K. Yu, and J. Yi, "Motion control of autonomous aggressive vehicle maneuvers," 2016, submitted to 2016 *IEEE Int. Conf. Adv. Intelli. Mechatronics*, downloadable at http://coewww.rutgers.edu/∼jgyi/AIM2016.pdf.

[5] N. R. Kapania, J. Subosits, and J. C. Gerdes, "A sequential two-step algorithm for fast generation of vehicle racing trajectories," in *Proc. ASME Dyn. Syst. Control Conf.*, Columbus, OH, October 2015.

[6] T. Lipp and S. Boyd, "Minimum-time speed optimisation over a fixed path," *Int. J. Control*, vol. 87, no. 6, pp. 1297–1311, 2014.

[7] J. P. Timings and D. J. Cole, "Minimum maneuver time calculation using convex optimization," *ASME J. Dyn. Syst., Meas., Control*, vol. 135, no. 2, pp. 1–9, 2013.

[8] S. LaValle and J. Kuffner, J.J., "Randomized kinodynamic planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, vol. 1, Detroit, MI, 1999, pp. 473–479.

[9] D. D. Dunlap, E. G. Collins Jr, and C. V. Caldwell, "Sampling based model predictive control with application to autonomous vehicle guidance," in *Florida Conference on Recent Advances in Robotics*, 2008.

[10] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robot. Res.*, vol. 30, no. 7, pp. 846–894, 2011.

[11] J.-H. Jeon, S. Karaman, and E. Frazzoli, "Anytime computation of time-optimal off-road vehicle maneuvers using the RRT*," in *Proc. IEEE Conf. Decision Control*, Orlando, FL, 2011, pp. 3276–3282.

[12] 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. Amer. Control Conf.*, Washington DC, 2013, pp. 188–193.

[13] Y. Li, Z. Littlefield, and K. E. Bekris, "Asymptotically optimal sampling-based kinodynamic planning," *Int. J. Robot. Res.*, 2016, in press.

[14] 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. Amer. Control Conf.*, Montreal, Canada, 2012, pp. 4239–4244.

[15] Z. Littlefield, Y. Li, and K. E. Bekris, "Efficient sampling-based motion planning with asymptotic near-optimality guarantees for systems with dynamics," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, Tokyo, Japan, 2013, pp. 1779–1785.

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

[17] R. Tedrake, I. R. Manchester, M. Tobenkin, and J. W. Roberts, "Lqr-trees: Feedback motion planning via sums-of-squares verification," *Int. J. Robot. Res.*, vol. 29, no. 8, pp. 1038–1052, 2010.

[18] H. B. Pacejka, *Tire and Vehicle Dynamics*, 2nd ed. Warrendale, PA: SAE International, 2006.

[19] A. Majumdar, A. A. Ahmadi, and R. Tedrake, "Control design along trajectories with sums of squares programming," in *Proc. IEEE Int. Conf. Robot. Autom.*, Karsruhe, Germany, 2013, pp. 4054–4061.