# A Motion Planner for Car-Like Robots Based on Rapidly-Exploring Random Trees

Rômulo Ramos Radaelli, Claudine Badue, Michael André Gonçalves, Thiago Oliveira-Santos, Alberto F. De Souza

Departamento de Informática, Universidade Federal do Espírito Santo, Vitória, ES, Brazil

rmradaelli@gmail.com, claudine@lcad.inf.ufes.br, es.michael@gmail.com, todsantos@inf.ufes.br, alberto@lcad.inf.ufes.br

**Abstract.** We propose a motion planner for car-like robots based on the rapidly-exploring random tree (RRT) method. Our motion planner was designed especially for cars driving on roads. So, its goal is to build trajectories from the car's initial state to the goal state in real time, which stay within the desired lane bounds and keep a safe distance from obstacles. For that, our motion planner combines several variants of the standard RRT algorithm. We evaluated the performance of our motion planner using an experimental robotic platform based on a Ford Escape Hybrid. Our experimental results showed that our motion planner is capable of planning trajectories in real time, which follow the lane and avoid collision with obstacles.

**Keywords:** Motion planning; car-like robots; rapidly-exploring random trees.

## 1 Introduction

Mapping, localization and control are very important tasks in autonomous robotics. Mapping involves the creation of a map of the environment around the robot, that may contain information pertaining to the places that the robot may or may not be able to navigate, localization involves the estimation of the robot's state relative to the map, and control involves the translation of control commands of velocity and steering wheel angle into acceleration, brake and wheel efforts. Another very important task in autonomous robotics, and the focus of this paper, is motion planning. To perform this task, a software module receives as input the map of the environment, the initial robot's state relative to the map, and the goal state; and produces as output a trajectory from the initial robot's state to the goal state. The trajectory might be represented in several ways. For a car-like robot, we represent it using a list of commands of velocity and steering wheel angle, along with the respective execution durations (a list of triplets $(v,\ \varphi,\ \Delta t)$). A proper trajectory leads the robot from its initial state to the goal state, while avoiding collision with known obstacles.

Different methods can be found in the literature to address the problem of motion planning for car-like robots. This type of robots is subject to both kinodynamic constraints (arising from kinematic and dynamic constraints) and environmental constraints (arising from obstacles). A possible approach is to reduce and discretize the state space so that classical search methods like A* [1] or D* [2] can find a collision-free trajectory from the car's initial state to the goal state in the constrained space. This approach usually converges very fast and provides good results. Nevertheless, the trajectory found with those methods might not be followed exactly by the car, because, in its building, not all the constraints of the movement of the car can be properly considered. Thus, the solution has to be further processed to generate an improved version that can be followed by the car. Another possible approach is to employ a more accurate model of the car motion and use sampling based search methods that can deal with the high-dimensional state space to find a feasible solution. An example of this type of method is the Rapidly-Exploring Random Tree (RRT), which was first introduced by LaValle [3]. RRT has been successfully used in the motion planning of car-like robots–the motion planning subsystem of the fourth placed winner of the DARPA Urban Challenge, the MIT robot "Talos", is based on the RRT [4].

In this paper, we propose a motion planner based on RRT for car-like robots that are subject to both kinodynamic and environmental constraints. Our motion planner was designed especially for cars driving on roads. So, its goal is to build trajectories from the car's initial state to the goal state in real time, which stay within the desired lane bounds and keep a safe distance from obstacles. For that, our motion planner combines two previous variants of the standard RRT to (i) reduce its sensitivity to the distance metric [5] and (ii) attempt to quickly grow the tree toward the goal state [6]. Our motion planner also incorporates five unique RRT variants to: (i) bias the location of sampling random states toward the lane region, (ii) select the most promising control commands for extending states, (iii) choose the best trajectories, (iv) discard non-promising states, and (v) reuse part of the trajectory built in the previous planning cycle. Although, in the form we have design them, they are unique to our work, the main ideas behind these last five RRT variants are similar to those proposed by Kuwata et al. [4] (first and third ones), Powers et al. [7] (second one), Frazzoli et al. [8] (fourth one), and Bekris and Kavraki [9] (fifth one).

We evaluated the performance of our motion planner using an experimental robotic platform based on a Ford Escape Hybrid. Our experimental results showed that our motion planner is capable of planning trajectories in real time, which follow the lane and avoid collision with obstacles. To the best of our knowledge, the combination of techniques we have employed to solve the motion planning problem is unique and the results we have obtained are satisfactory.

This paper is organized as follows. After this introduction, in Section 2, we present the motion model of car-like robots we have used and the standard RRT algorithm. In Section 3, we describe our approach for motion planning for car-like robots. In Section 4, we describe our experimental methodology and, in Section 5, we analyze our experimental results. Our conclusions and directions for future work follow in Section 6.

## 2 Background

The motion model for car-like robots can be described as follows. Let $x$ and $y$ be the car's location, given by the midway of the two rear wheels; $\theta$ the car's orientation; $L$ the distance between the front and rear wheels' axles; $v$ the car's velocity; $a$ the car's acceleration; $\varphi$ the steering wheel angle, given by the average of the angle of the right and left front wheels; and $r$ the rate of change of the steering wheel angle. Also, let $x_t = (x, y, \theta, v, \varphi)$ be the state of the car at time $t$ and $u_t = (v', \varphi', \Delta t)$ the control command at time $t$. So, after the small $\Delta t$ time interval, the car will be at state $x_{t+1} = (x', y', \theta', v', \varphi')$ given by $x' = x + \Delta t\, v' \cos\theta$, $y' = y + \Delta t\, v' \operatorname{sen}\theta$, and $\theta' = \theta + \Delta t\, v' \frac{\tan\varphi'}{L}$, where $v' = v + \Delta t\, a$ and $\varphi' = \varphi + \Delta t\, r$.

A high-level description of the standard RRT algorithm [10] is given in the following. The initial car's state, $x_{init}$, is added to an initially empty tree, $T$. At each iteration, a random state, $x_{rand}$, is firstly taken from the collision-free state space, $X_{free}$. Secondly, the state closest to $x_{rand}$ already present in $T$, $x_{near}$, is identified according to a distance metric. Thirdly, $x_{near}$ is extended to a new state, $x_{new}$, as follows. A control command is selected either randomly or according to a specific criterion. A possible criterion is to choose the command that yields a $x_{new}$ as close as possible to $x_{rand}$. The command is then applied to $x_{near}$ over a small time interval, which creates $x_{new}$. If $x_{new}$ lies in $X_{free}$, then a vertex representing $x_{new}$ and a directed edge representing the command that takes $x_{near}$ to $x_{new}$ are added to $T$. A trajectory is found when $x_{new}$ reaches the goal state, $x_{goal}$. The iterative procedure is executed until a stop criterion is satisfied (e.g., $x_{goal}$ is reached or a maximum number of iterations is achieved).

## 3 Our Approach for Motion Planning

Our motion planner incrementally builds $T$ from $x_{init}$ using random states biased toward the lane region, and plans a trajectory from $x_{init}$ to $x_{goal}$ in real time, which stays within the desired lane bounds and keeps a safe distance from obstacles. At each planning cycle, our motion planner receives as input: (i) an updated map of the environment around the car; (ii) the initial car's state relative to the map; (iii) the location of the lane center relative to the map; (iv) an updated list of goal states, comprising intermediate goal states and the final goal state; and (v) the trajectory built in the previous planning cycle. Our motion planner then produces as output a trajectory from the initial car's state, $x_{init}$, to the first goal state, $x_{goal}$, of the updated list of goal states. The trajectory is represented by a list of control commands (triplets ($v$, $\varphi$, $\Delta t$)) that leads the car from $x_{init}$ to $x_{goal}$ and is planned at a fixed frequency (12.5 hz in our experiments).

It is important to note that, in spite of being ready to, our motion planner does not handle movable obstacles yet. For this, it would be enough to add a system for tracking the movable obstacles states in the present and estimating their states in the future.

The movable obstacle state would comprise obstacle's geometry, pose, and velocity. When selecting $x_{near}$ or control commands to extend $x_{near}$, given the future states of movable objects, our motion planner would discard states or commands which would lead to a collision with a movable obstacle.

### 3.1 Reducing Metric Sensitivity

Our motion planner incorporates the RC-RRT [5], a variant of the standard RRT designed for reducing its sensitivity to the distance metric. The RC-RRT collects information during the exploration of the state space and extends $T$ according to both the distance metric and exploration information, which can make it less sensitive to the distance metric. For each state in $T$, the RC-RRT records whether a control command has already been applied to the state. If a command has already been applied to the state, it is *discarded*, i.e., it is not considered for the state anymore. If all possible commands have already been applied to the state, it is *discarded*, i.e., it is not considered as a $x_{near}$ anymore. For each state in $T$, RC-RRT also computes a constraint violation frequency (CVF). For each $x_{new}$ inserted to $T$, its CVF is initialized to zero. When a control command applied to the state leads to a collision, the CVF of the state is increased by $\frac{1}{m}$, the CVF of the parent state is increased by $\frac{1}{m^2}$, and the CVF of the $k$-th parent state is increased by $\frac{1}{m^{k+1}}$, where $m$ is the number of possible commands. So, each state's CVF is bounded to the $[0, 1]$ interval. When selecting a state to be extended, RC-RRT verifies if the commands for the state are exhausted. If so, the state is discarded; otherwise, the state is discarded with a probability equal to its CVF.

### 3.2 Biasing the Location of Random States Toward the Lane Region

Another variant of the standard RRT is employed to bias the location of random states toward the lane region. For that, with a given probability, $x_{rand}$ is taken from the lane region instead of from the whole state space as follows. A first sample is taken randomly from the lane center between $x_{init}$ and $x_{goal}$, and a second sample is taken randomly from a circle with a small radius centered in the first sample. The second sample is then considered as $x_{rand}$.

### 3.3 Selecting the Most Promising Control Commands

Another variant of the standard RRT is employed to select the most promising control commands for extending states. It considers not only the proximity to $x_{rand}$, but also other criteria associated with environmental and traffic-law constraints, namely distance from obstacles, proximity to the lane center, maintenance of velocity limits, and execution of authorized maneuvers. The space of control commands is discretized and each possible command, $u$, receives a cost calculated according to the following equation:

$$u.cost = \sum_i cost(u, c_i) * weight(c_i), \quad (1)$$

where $c_i$ is a command selection criterion, $cost(u,\ c_i)$ is the cost assigned to $u$ with regard to $c_i$, and $weight(c_i)$ is the weight assigned to $c_i$. Both $cost(u,\ c_i)$ and $weight(c_i)$ are bounded to $[0, 1]$. The sum of the weights of all criteria is equal to one, i.e., $\sum_i weight(c_i) = 1$. When selecting a control command for extending a state, for each possible command, our motion planner verifies if the command has already been applied to the state. If so, the command is discarded; otherwise, it is applied to the state to check if $x_{new}$ lies in $X_{free}$. If not, the command is discarded, and the CVF of the state and the parent states are increased. The non-discarded command with smallest cost is selected. Our motion planning considers the five criteria described below to select a control command.

1. Proximity to $x_{rand}$. Higher costs are attributed to commands that take $x_{new}$ farther from $x_{rand}$, while lower ones are assigned to commands that bring $x_{new}$ closer to $x_{rand}$.
2. Distance from obstacles. An occupancy grid map is used to represent the probability of occupancy by obstacles. An obstacle distance grid map is used to represent the proximity to obstacles. The obstacle distance map is derived from the occupancy map as follows. If the value of an occupancy map cell is higher than a given threshold, then it is considered to be occupied and the corresponding cell of the obstacle distance map is set to one. Otherwise, it is considered free and the corresponding cell of the obstacle distance map is made equal to the corresponding cell of the occupancy map. The obstacle distance map is used to attribute costs to control commands according to the distance from obstacles. Higher costs are attributed to commands that bring $x_{new}$ closer to an obstacle, while lower ones are assigned to those that take $x_{new}$ farther from an obstacle.
3. Proximity to the lane center. A lane distance grid map is used to represent the proximity to the lane center. In the lane distance map, a value in the $[0, 1]$ interval indicates the distance to the lane center: the closer to 1 a grid cell, the farther it is to the lane center. The lane distance map is used to attribute costs to control commands according to the proximity to the lane center. Higher costs are attributed to commands that extend a state to another farther from the lane center, while lower ones are assigned to those that extend a state to another closer to the lane center.
4. Maintenance of velocity limits. A minimum cost is attributed to velocity commands equal to the maximum limit; the cost increases as the difference between the velocity command and the desired limits increases.
5. Execution of authorized maneuvers. A maximum cost is attributed to control commands that move the car backwards, while a minimum one is assigned to those that move it forward.

### 3.4 Selecting the Best Trajectories

Another variant of the standard RRT is employed to select the best trajectories. It takes into account not only the time to achieve $x_{goal}$, but also other criteria associated with environmental constraints, namely distance from obstacles and proximity to the lane center (second and third criteria described above in Section 3.3). The trajectory

cost is defined by the cost of its last state, i.e., the state that reaches $x_{goal}$. The initial state, $x_{init}$, receives a cost equals to zero. Each of the other states, $x$, receives a cost computed according to the following equation:

$$x.cost = x_{parent}.cost + time(x_{parent}, x, u) + \sum_i cost(x, c_i) * weight(c_i), \quad (2)$$

where $x_{parent}$ is the parent state; $time(x_{parent}, x, u)$ is the time taken by the control command, $u$, to bring $x_{parent}$ to $x$; $cost(x,\ c_i)$ is the cost assigned to $x$ with regard to $c_i$; and $weight(c_i)$ is the weight assigned to $c_i$. Both $cost(x,\ c_i)$ and $weight(c_i)$ are bounded to $[0, 1]$. The sum of the weights of all criteria is equal to one, i.e., $\sum_i weight(c_i) = 1$.

At each planning cycle, $T$ is made empty and the cost of the best trajectory, $P_{best}$, is initialized to infinity. Every time a new trajectory, $P_{new}$, is found, the cost of the best trajectory is updated to $P_{best}.cost = \min(P_{best}.cost, P_{new}.cost)$.

### 3.5 Growing the Tree Toward the Goal State

The Connect heuristic [6] is employed to attempt to quickly grow $T$ toward $x_{goal}$ after each successful extension ($x_{new}$ addition). Instead of attempting to extend $T$ by a single step, the Connect heuristic repeats the extension step until $x_{goal}$ or an obstacle is reached. In this way, if $x_{goal}$ is in the car's direction and there is no obstacle between the car and $x_{goal}$, a trajectory may be generated with a smaller number of iterations.

### 3.6 Discarding Non-Promising States

Another variant of the standard RRT is employed to discard non-promising states. For that, a state, $x$, has its lower bound on the cost-to-go to $x_{goal}$ estimated as:

$$x.cost\_to\_go = \frac{\|x - x_{goal}\|}{v_{max}}, \quad (3)$$

where $\|x - x_{goal}\|$ is the Euclidean distance between $x$ and $x_{goal}$, and $v_{max}$ is the maximum velocity. Every time a new trajectory is found, for each state, $x$, if $x.cost + x.cost\_to\_go \geq P_{best}.cost$, then $x$ can be safely discarded from $T$, as it cannot provide a better solution than the one that has just been found. Additionally, whenever $x_{new}$ is created, if $x_{new}.cost + x_{new}.cost\_to\_go \geq P_{best}.cost$, then $x_{new}$ is discarded.

### 3.7 Reusing Part of the Previous Trajectory

Another variant of the standard RRT is employed to reuse part of the trajectory built in the previous planning cycle. A large part of the trajectory built in the previous planning cycle might still be valid and can be used to speed up the search for a new solution. The valid states – those that lie in $X_{free}$ and have not been followed by the car yet – are added to $T$.

### 3.8 Our Motion Planner Algorithm

A high-level description of our motion planner algorithm is given in the following. The initial car's state, $x_{init}$, and the valid states of the previous trajectory are added to $T$. At each iteration, the random state, $x_{rand}$, is firstly taken from the lane region with a given probability. Secondly, $x_{new}$ is identified using the Euclidean distance and the CVF information as follows. For each state in $T$, $x$, the motion planner checks if all possible commands have already been applied to $x$. If so, $x$ is discarded; otherwise, the probability of discarding $x$ is equal to the value of its CVF. The non-discarded $x$ with least distance to $x_{rand}$ is selected as $x_{near}$. Thirdly, $x_{near}$ is extended as follows. For each possible command, $u$, the motion planner verifies if $u$ has already been applied to $x_{near}$. If so, $u$ is discarded; otherwise, $u$ is applied to $x_{near}$ to check if $x_{new}$ lies in $X_{free}$. If not, $u$ is discarded, and the CVF of $x_{near}$ and the parent states are increased. The non-discarded $u$ with smallest cost is applied to $x_{near}$ to verify if $x_{new}.cost + x_{new}.cost\_to\_go \geq P_{best}.cost$. If so, $x_{new}$ is discarded; otherwise, a vertex representing $x_{new}$ and a directed edge representing the command that takes $x_{near}$ to $x_{new}$ are added to $T$. Fourthly, if the extension is successful, the Connect heuristic repeats the extension step until $x_{goal}$ or an obstacle is reached. Fifthly, if $x_{goal}$ is reached and $P_{new}.cost = x_{new}.cost < P_{best}.cost$, then the best trajectory is updated and $T$ is *pruned*, i.e., for all states, $x$, in $T$, if $x.cost + x.cost\_to\_go \geq P_{best}.cost$, then $x$ is discarded from $T$.

The iterative procedure is executed until the minimum planning time is reached. However, if no trajectory is found during the minimum planning time, the iterative procedure is executed until the maximum planning time (or timeout). In case of timeout, the car is stopped.

## 4 Experimental Methodology

Our Intelligent and Autonomous Robotic Automobile (IARA) is based on a Ford Escape Hybrid (**Fig. 2**(a)). It has several high-end sensors, including: two Point Grey Bumblebee XB3 stereo cameras and two Point Grey Bumblebee 2 stereo cameras, one Light Detection and Ranging (LIDAR) Velodyne HDL 32-E, and one GPS-aided Attitude and Heading Reference System (AHRS/GPS) Xsens MTiG. To process the data coming from the sensors, the platform can hold up to four Dell Precision R5500. We implemented many software modules for IARA that currently allows for its autonomous operation, such as modules for mapping, localization, behavior selection, path following, control, and motion planning (that is the focus of this paper). We also implemented a software module for autonomous vehicle simulation to help in the development and testing of all the other IARA's modules (that was also used to evaluate the performance of the motion planner presented in this paper).

The main parameters of our motion planner are: maximum car's velocity, $v_{\max}$; maximum steering wheel angle, $\varphi_{\max}$; minimum planning time, $t_{min}$; maximum planning time (or timeout), $t_{max}$; maximum distance between states, $d_{max}$, given by the maximum distance $x_{near}$ and $x_{new}$; and the probability of taking $x_{rand}$ from the

lane region, $p_{bias}$. The values of the main parameters used in our simulation experiments were: $v_{\max} = 2.5$ m/s, $\varphi_{\max} = 26.4$ deg, $t_{min} = 120$ s, $t_{max} = 120$ s, $d_{max} = 3.5$ m, and $p_{bias} = 0.8$. The values of the parameters used in the experiments with IARA are the same as those used in the simulation experiments, except for $t_{min} = 0.08$ s and $t_{max} = 0.8$ s.

The experiments with our autonomous vehicle simulator were carried out on maps and other data computed using sensor logs captured by IARA while manually driven on a part of the road that surrounds the main campus of our university. For defining the location of the lane center, IARA was driven along the center of the desired lane segment of the road. During the course, estimated car's locations were acquired. These locations are considered as lane center estimates and provided as input to our motion planner.

The experiments with IARA were conducted on a parking lot of the campus of our university. We defined a fake lane in the parking lot and driven IARA along the center of the fake lane for acquiring car's locations to be provided as lane center estimates.

# 5 Experimental Results and Discussion

## 5.1 Experiments with the Autonomous Vehicle Simulator

To evaluate the contributions of each RRT variant considered in this paper, we examined the performance of our motion planner in the simulator using five different configurations. The Configuration 1 incorporates only the standard RRT. The Configuration 2 incorporates only the RC-RRT [5]. The Configuration 3 incorporates (a) the Configuration 2 and (b) discarding of non-promising states using only the time criteria. The Configuration 4 incorporates (a) the Configuration 2 and (b) selection of the most promising control commands for extending states, discarding of non-promising states, and selection of the best trajectory using various criteria associated with environmental and traffic-law constraints. The Configuration 5 incorporates (a) the Configuration 2, (b) the Configuration 4, and (c) biasing of the location of random states toward the lane region and the Connect heuristic [6].

In the simulation experiments, the list of goals was composed of only the final $x_{goal}$. Thus, the motion planner executed only a single planning cycle. Consequently, the reuse of previous trajectories was not evaluated. Finally, it was used the same $x_{init}$ and final $x_{goal}$.

**Fig. 1** shows the results of our simulation experiments. In each of these figures, the blue rectangle denote $x_{init}$, the yellow rectangle the final $x_{goal}$, yellow lines $T$ edges, grey lines edges leaving states with a high CVF (equal to 1), red lines edges leaving discarded states, green lines trajectories found, and the blue line the best trajectory found.

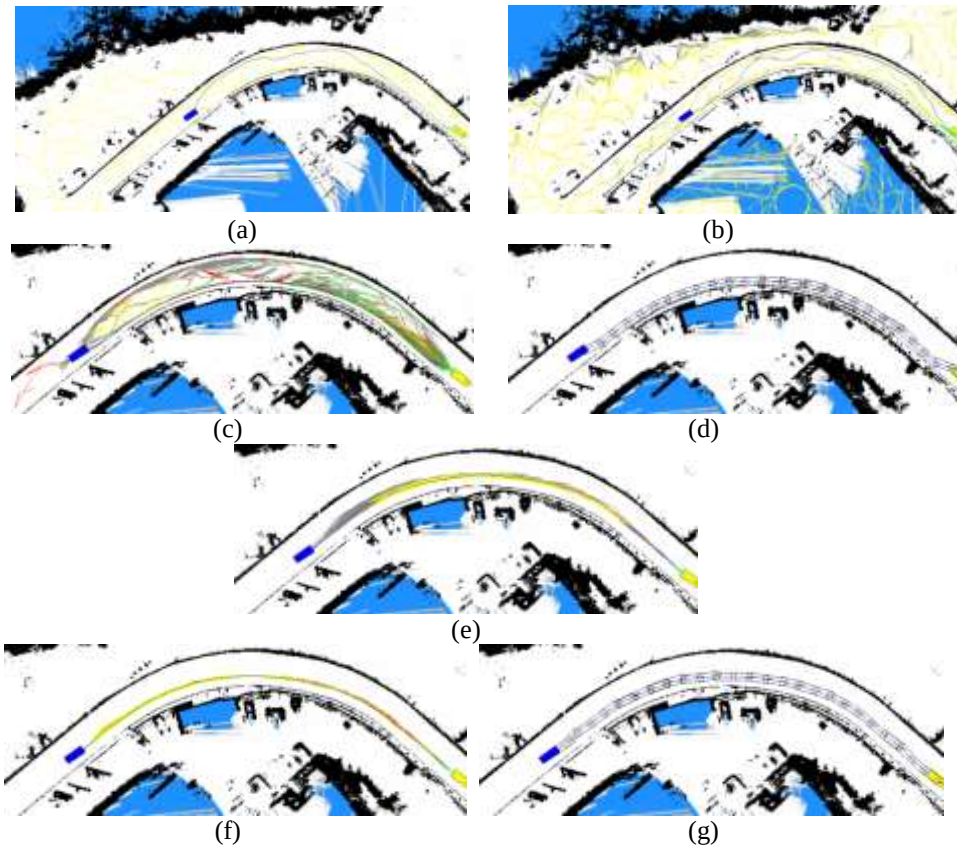


**Fig. 1.** Performance of our motion planner using: (a) Configuration 1, (b) Configuration 2, (c, d) Configuration 3, (e) Configuration 4, and (f, g) Configuration 5. For visibility purposes, we provide two different views of the same result: views (c) and (d) show *T* and the best trajectory found for Configuration 3, and views (f) and (g) show *T* and the best trajectory found for Configuration 5.

**Fig. 1**(a) shows the performance of our motion planner using Configuration 1. As it can be observed in **Fig. 1**(a), the state space was sparsely explored. Also, the final trajectory (blue line) passes close to obstacles and far from the lane center.

**Fig. 1**(b) shows the performance of our motion planner using Configuration 2. Our motion planner with the RC-RRT explored the state space much better than with the standard RRT. The reason is that the RC-RRT avoids similar expansions by applying a control command to a state only once. This guarantees that new *T* expansions will always explore new regions. The RC-RRT also penalizes states whose expansion attempts are likely to fail based on the CVF. As it can be seen in **Fig. 1**(b), states with a high CVF (grey lines) are those close to obstacles that, most of the time, extend to obstacles.

Despite the fact that the RC-RRT explored the state space better than the standard RRT, the solutions found by both algorithms are equivalent. Both algorithms explore

the whole state space, instead of focusing the search on the region of interest (the lane), which result in feasible solutions but clearly far from the optimum one. To avoid exploration of non-interesting regions, our motion planner with Configuration 3 estimates the lower bound on the cost-to-go to $x_{goal}$ for all states using the time criteria, and discards and prevents the insertion in $T$ of those that have lower bound costs (plus their own costs) higher than the best trajectory found until the current moment. **Fig. 1**(c, d) shows the performance of our motion planner using Configuration 3. For visibility purposes, we provide two different views of the same result: **Fig. 1**(c) shows $T$ built during the planning cycle and **Fig. 1**(d) shows the best trajectory found. As it can be observed in **Fig. 1**(c) and (d), the motion planner focused the exploration on the lane region (**Fig. 1**(c)) and found a trajectory close to the optimum one (**Fig. 1**(d)). However, although the trajectory found is close to the optimum one in terms of time, it is far from the desired one in terms of distance from obstacles and proximity to the lane center.

To provide trajectories that do not violate environmental and traffic-law constraints, our motion planner adopts a more sophisticated approach to select the most promising control commands for extending states, to choose the best trajectories, and to discard non-promising states considering other criteria (namely distance from obstacles, proximity to the lane center, maintenance of velocity limits and execution of authorized maneuvers). **Fig. 1**(e) shows the performance of our motion planner using Configuration 4. As it can be seen in **Fig. 1**(e), the trajectory found maintains distance from the curbs and stays within the lane bounds.

To further restrict exploration to the region of interest, our motion planner biases the location of random states toward the lane region. Also, to attempt to quickly grow $T$ toward $x_{goal}$, whenever an expansion is well succeeded, our motion planner employs the Connect heuristic. **Fig. 1**(f) and (g) show the performance of our motion planner using Configuration 5. For visibility purposes, we provide two different views of the same result: **Fig. 1**(f) shows $T$ built during the planning cycle and **Fig. 1**(g) shows the best trajectory found. As it can be observed in **Fig. 1**(f) and (g), the motion planner restricted the exploration on the lane region even further (**Fig. 1**(f)) and built a trajectory from a safe distance from the curbs and within the lane bounds (**Fig. 1**(g)).

### 5.2 Experiments with IARA

Different from the simulation experiments, in the experiments with IARA the list of goals was composed of several goal states. Thus, the motion planner executed several planning cycles and, consequently, reused part of previous trajectories. Also, it was used approximately the same $x_{init}$ (in the real world it is impossible to pose the car in the same state more than once) and the same final $x_{goal}$.

We executed the same experiment with IARA ten times. The experiments were executed with a safety driver and a computer operator inside the car. The former was responsible for stopping the car if any problem occurred, and the later for operating and observing the progress of the IARA's autonomous operating system.

**Fig. 2**(b) shows the estimated IARA's locations acquired during each of the ten runs. In this figure, different-colored lines denote the estimated IARA's locations in

distinct runs. In all runs, the motion planner built trajectories within the fake lane around the parking lot. The whole trajectory length (from $x_{init}$ to $x_{goal}$) was 145.82 m long on average and the whole trajectory time was of 60.18 s on average. Also, although $T$ is built randomly, in all runs our motion planner built very similar trajectories. The reason is that the search of the state space was biased toward the lane region, which constrained the possible solutions. Finally, the motion planner was capable of planning trajectories in only 0.08 s (12.5 hz).


(a) (b)

**Fig. 2.** (a) Intelligent and Autonomous Robotic Automobile (IARA); (b) Estimated locations of IARA acquired during each one of the ten runs

## 6 Conclusions and Future Work

We presented a motion planner for car-like robots driving on roads based on RRT. Our motion planner combines several variants of the standard RRT algorithm to reduce its sensitivity to the distance metric, build trajectories within the desired lane bounds, and speed up its convergence.

We evaluated the performance of our motion planner using an autonomous vehicle simulator. To evaluate the contributions of each RRT variant considered in this paper, we examined the performance of our motion planner using several combinations of them, ranging from the most naïve (that includes only the standard RRT) to the most sophisticated one (that includes all the RRT variants considered). The results of the simulation experiments showed that the most sophisticated configuration outperforms the others in terms of distance from obstacles and proximity to the lane center. We also analyzed the performance of our motion planner using the Intelligent and Autonomous Robotic Automobile (IARA) based on a modified Ford Escape Hybrid. The results of the experiments with IARA showed that our motion planner is capable of planning trajectories in real time, which stay close to the lane center and keep a safe distance from obstacles.

Directions for future work include: (i) smoothing of the trajectory to make it more close to that generated by a human driver; (ii) interaction with other environmental structures, such as movable obstacles and traffic signs; (iii) motion planning on unstructured parking lots; and (iv) comparison of the performance of our motion planner and human drivers using IARA.

## References


1. Dolgov, D., Thrun, S., Montemerlo, M., Diebel, J.: Path planning for autonomous driving in unknown environments. In Khatib, O., Kumar, V., Pappas, G., eds. : Experimental Robotics: The Eleventh International Symposium. Springer, Berlin (2009) 55-64
2. Urmson, C., Anhalt, J., Bagnel, D., Baker, C., Bittner, R., Clark, M., Dolan, J., Duggins, D., Gittleman, M., Harbaugh, S., Wolkowicki, Z., Ziglar, J., Bae, H., Brown, T., Demitrish, D., Sadekar, V., Zhang, W., Struble, J., Taylor, M., Darms, M., Ferguson, D.: Autonomous driving in urban environments: Boss and the urban challenge. Journal of Field Robotics: Special Issues on the 2007 DARPA Urban Challenge 25(8), 425-466 (2008)
3. LaValle, S.: Rapidly-exploring random trees: a new tool for path planning. Technical Report , Iowa State University (1998)
4. Kuwata, Y., Fiore, G., Teo, J., Frazzoli, E., How, J.: Motion planning for urban driving using RRT. In : IEEE/RSJ International Conference on Intelligent Robots and Systems, pp.1681-1686 (2008)
5. Cheng, P., LaValle, S.: Resolution complete rapidly-exploring. In : IEEE International Conference on Robotics and Automation, pp.267–272 (2002)
6. Kuffner, J., LaValle, S.: RRT-connect: an efficient approach to single-query path planning. In : IEEE International Conference on Robotics and Automation, pp.995-1001 (2000)
7. Powers, M., Wooden, D., Egerstedt, M., Christensen, H., Balch, T.: The sting racing team's entry to the urban challenge. In Rouff, C., Hinchey, M., eds. : Experience from the DARPA Urban Challenge. Springer-Verlag, London (2012) 43-66
8. Frazzoli, E., Dahleh, M., Feron, E.: Real-time motion planning for agile autonomous vehicles. In : American Control Conference, pp.43-49 (2001)
9. Bekris, K., Kavraki, L.: Greedy but safe replanning under kinodynamic constraints. In : IEEE International Conference on Robotics and Automation, pp.704-710 (2007)
10. LaValle, S., Kuffner, J.: Rapidly-exploring random trees: progress and prospects. In Donald, B., Lynch, K., Rus, D., eds. : Algorithmic and Computational Robotics: New Directions. A. K. Peters, Welessley (2001) 293-308
11. Macek, K., Becked, M., Siegwart, R.: Motion planning for car-like vehicles in dynamic urban scenarios. In : IEEE/RSJ International Conference on Intelligent Robots and Systems, pp.4375-4380 (2006)
12. Franchi, A., Freda, L., Oriolo, G., Vendittelli, M.: A Randomized Strategy for Cooperative Robot Exploration. International Conference on Robotics and Automation 768-774 (2007)
13. Chang-an, L., Jin-gang, C., Guo-dong, L., Chun-Yang, L.: Mobile robot path planning based on an improved rapidly-exploring random tree in unknown environment. International Conference on Automation and Logistics, 2375-2379 (2008)
14. LaValle, S. M., Kuffner, J. J. .: Randomized kinodynamic planning. In : IEEE International Conference on Robotics and Automation, pp.473-479 (1999)
15. Ju, T., Liu, S., Yang, J., Sun, D.: Rapidly exploring random tree algorithm-based path planning for robot-aided optical manipulation of biological cells. Transactions on Automation Science and Engineering 11(3), 649-657 (2014)