2018 IEEE Intelligent Vehicles Symposium (IV)
Changshu, Suzhou, China, June 26-30, 2018

Model-based Decision Making with Imagination for Autonomous Parking*

Ziyue Feng[1], Shitao Chen[2], Yu Chen[3] and Nanning Zheng[4], Fellow, IEEE

Abstract— **Autonomous parking technology is a key concept within autonomous driving research. This paper will propose an imaginative autonomous parking algorithm to solve issues concerned with parking. The proposed algorithm consists of three parts: an imaginative model for anticipating results before parking, an improved rapid-exploring random tree (RRT) for planning a feasible trajectory from a given start point to a parking lot, and a path smoothing module for optimizing the efficiency of parking tasks. Our algorithm is based on a real kinematic vehicle model; which makes it more suitable for algorithm application on real autonomous cars. Furthermore, due to the introduction of the imagination mechanism, the processing speed of our algorithm is ten times faster than that of traditional methods, permitting the realization of real-time planning simultaneously. In order to evaluate the algorithm's effectiveness, we have compared our algorithm with traditional RRT ,within three different parking scenarios. Ultimately, results show that our algorithm is more stable than traditional RRT and performs better in terms of efficiency and quality.**

I. INTRODUCTION

Self-parking is one of the key technologies to achieve autonomous driving. The Society of Automotive Engineers (SAE) J3016 standard defines five distinct levels of vehicle automation, including the parking assist technologies. The first three are progressions of assisted parking, i.e., driver assistance, partial automation and conditional automation. The fourth level is a near-automated technology called high automation; while the last one is totally automated, which is defined as full automation. Currently, the parking capabilities of a number of vehicles on the road nowadays are among level 1 and level 2. It is commonly recognized that autonomous parking is logically the first step towards commercial applications of self-driving, with fully automated vehicles. At present we are not yet at a stage where autonomous parking is completely achievable. However, many researchers are attempting to reach this stage through continuous research. Most car manufacturers and technology companies in the automation industry are expecting fully automated vehicles to be available by 2020, with self-parking as a primary feature.

For the realization of self-parking, the parking-planning task is that, when given a parking lot, we need to find the optimal collision free trajectory, under vehicle kinematic constraints to maneuver a car into a target place. Yet, there are several difficulties involved in this task. The combination of both moving forward actions and backward in parking planning means this process is more complex than on-road planning. Sometimes moving backwards slightly can make it easier for the car to maneuver to a narrower place. However, a negative result of this action is that it could lead to a decreased human-like trajectory performance, as well as a uncomfortable passenger experience. Moreover, the turning angle of the parking trajectory is typically bigger than on-road trajectory, therefore the corridor is much more narrow, which makes the influence caused by the difference of radius between inner wheels greater when moving forward, this is the same for the outer wheels, when moving backward.

The RRT algorithm is a popular method to solve motion planning problems. It was first proposed by Steven *et al.* [1], and has then been applied to the parking-planning task [2]. This method constructs a tree to store some accessible places from the start point, and then it randomly grows the tree to explore the whole space until the target point is close enough to the tree. Finally, a feasible trajectory will be generated. The RRT can easily take into account and handle complex environments with obstacles.

However, there is still a long way to go between the RRT and the practical parking planning tasks. When the car gets closer to the target point, a little bit of bias reduction needs to be executed with quite a few steps. This causes the car getting close to the garage quickly but then moves forward and backward over and over again in order to adjust to a perfect position. The whole parking process with RRT takes a long time and makes the final part of the trajectory complex. Bi-RRT [3] uses tree growing from target to reduce the search time, but the final part of the trajectory is still complex. We imagine that the car is already parked in the parking lot and then we try to drive the car out of the parking lot with various pre-defined driving strategies. After the operation we will obtain dozens of feasible driving-out paths, each with twenty nodes. By reversing these driving-out paths to parking-in paths, there will be a tree with dozens of paths and hundreds of nodes. Then we expand the target point, in RRT algorithm, to this well-defined model-based target tree. Once RRT tree reaches any node of the target tree, the final trajectory will be generated. Since the model-based target tree is well defined it results in the trajectory in target tree being very smooth. The car can get into the target position in one movement. With this target expansion, the RRT tree can get close to the target, with fewer steps and

*This research was partially supported by the National Natural Science Foundation of China (No. 61773312, 61790563), the Programme of Introducing Talents of Discipline to University (No. B13043).

Ziyue Feng[1], Shitao Chen[2], Yu Chen[3] are with Institute of Artificial Intelligence and Robotics in Xi'an Jiaotong University, Xi'an, Shannxi, P.R.China e-mail:{brother_yue, chenshitao, alan19960212}@stu.xjtu.edu.cn

Nanning Zheng[4] is the director of Institute of Artificial Intelligence and Robotics, Xi'an Jiaotong University, Xi'an, Shannxi, P.R.China Correspondence: nnzheng@mail.xjtu.edu.cn

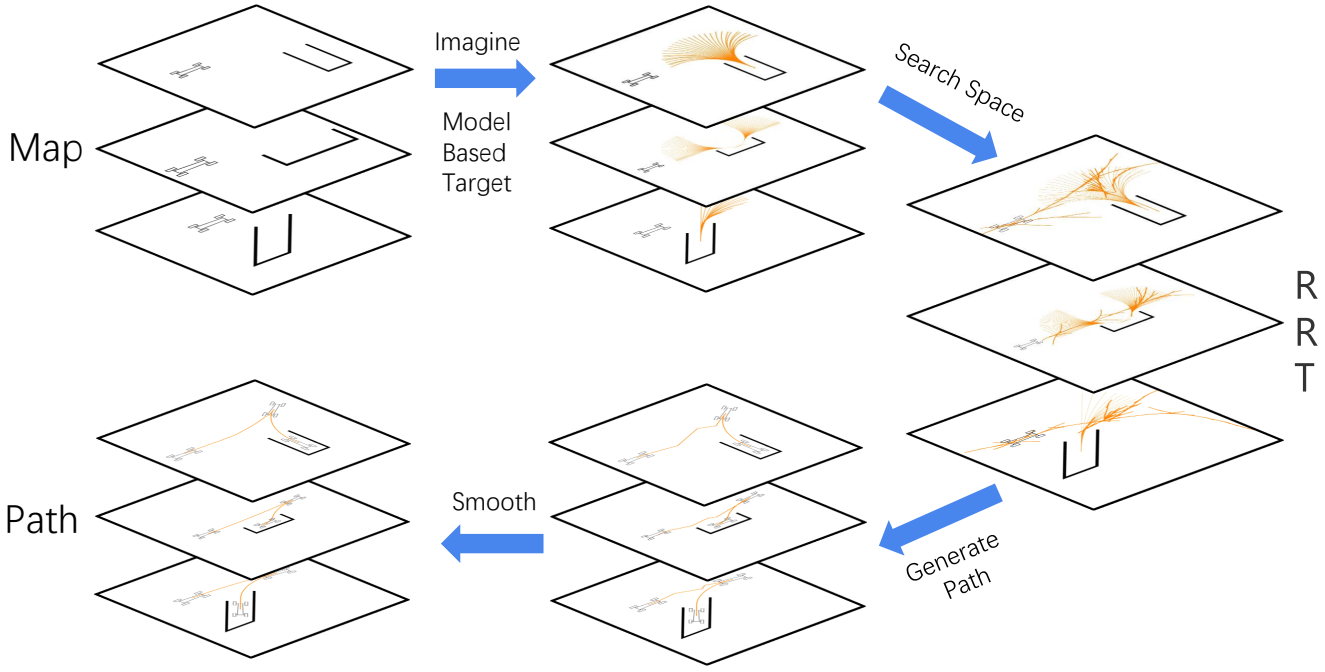


Fig. 1. The main workflow of our algorithm. The parking task has three scenarios: perpendicular, parallel and echelon. Imagine the car is already parked in the parking lot, use the algorithm that corresponds to the specific parking scenario to build a target tree, and then use the modified RRT to search the configure space and connect paths from RRT tree and target tree. After that, the trajectory is generated. Once that is completed we use our smoothing algorithm to smooth the generated trajectory under vehicle kinematic constraints.

less time cost. In fact, model-based planning is more than ten times quicker than traditional RRT. Our algorithm is not only better but is more stable than traditional RRT, in planning time and quality. RRT still has another drawback, that its trajectory can be extremely complex. This is because every edge of the tree is generated towards a random direction. There are some methods to smooth the trajectory, but they do not take into account the kinematic constraints of a real car. This may not be a problem with on-road planning, but since the parking lot is much narrower than an on-road environment, the smoothed parking trajectory may be unfeasible. We propose an algorithm to smooth the trajectory obeying the kinematic constraints. The trajectory smoothed via our algorithm has a guaranteed feasibility. With these improvements, our algorithm can be applied in actual parking planning task.

II. Related Work

Path planning for parking tasks [4] means to find a collision-free path from a given start point to a final target point in a parking lot. The very beginning stages of developing path planning methods is the car of Dubins *et al.* [5], as well as that of Reeds and Shepp *et al.* [6] in the domain of robotics. However, these two methods do not consider obstacles and continuous turning angles. Then some research have been done in the field of motion planning. Methods of motion planning for automated vehicles can be divided into four groups [7], including graph search-based planners, for example, Dijkstra algorithm [8], [9] and A-Star algorithm [10], sampling-based planners like RRT [1], deep learning-based planners like [11], [12], [13] and numerical optimization like [14]. Our method is based on RRT, which belongs to the second group. RRT is one of the sampling-based planning methods. It has many improved branches, like Bi-RRT [3], RRT* [15], CL-RRT [16] and DD-RRT [17]. It has also been applied to parking planning task in [2]. Our method has achieved many improvements to the basic RRT. We expand the target point to a model-based target tree. This expansion allows for the generated path to be more smooth and human-like. However, a downside to the sampling-based approach is the uncertainty of planning time. Our expansion can make the planing time shorter and more stable. Another drawback of every sample-based planning algorithm is that every segment of the path is generated with random direction and target, which which results in the path being more complex and with some redundant nodes. Some smoothing algorithms already exist for this particular problem, including, cubic polynomials [18], quintic polynomials [19], [20], Bezier curves [21], [22], [23], B-splines [24] and Clothoids [25]. All of these smoothing algorithms have a common drawback: they violently distort the path into a smooth curve without considering the kinematic constraints of a real car. In

this paper, we propose a new smoothing algorithm that takes the vehicle kinematic constraints into consideration. The path smoothed by our algorithm is guaranteed to be practical.

III. Method

We divide the parking tasks into three scenarios: perpendicular parking, parallel parking and echelon parking. Imagine the car is already parked in the parking lot, we define three different model-based target trees with three different scenarios. Each target tree has dozens of paths and hundreds of nodes. We then go on to expand the target point in RRT algorithm to this tree. Once the RRT tree reaches any node of the target tree the final trajectory is generated. This trajectory consists of two parts: a part from RRT tree and a part from the target tree. The model-based target tree makes the latter extremely smooth and enables a processing time of our algorithm more than ten times faster than the traditional RRT. Our algorithm is not only more efficient but more stable in planning time and quality. We use our smoothing algorithm to smooth the former part of the trajectory under vehicle kinematic constraints. Different from other smoothing algorithms, trajectories smoothed by our algorithm have guaranteed feasibility. This procedure is illustrated in Fig. 1.

A. Point Pursuit

In this paper, a basic algorithm is called 'point pursuit', which calculates the best turning angle to maneuver the car to get it closer to a particular point. Consider a vector (X, Y, θ) to indicate a car's state. X and Y is its position in a 2D surface, θ is the orientation. As shown in Fig. 2, the car's starting point is $P(0, 0, \frac{\pi}{2})$, the target point is $T(X_t, Y_t, \theta_t)$. The car will move 0.1 meters with a fixed turning angle ϕ to point $A(X_a, Y_a, \theta_a)$. Therefore, we need to find the best ϕ to make the distance between P and A as small as possible. The ϕ mentioned previously is an equivalent turning angle calculated by outer wheel turning angles ϕ_o and inner wheel turning angle ϕ_i.

$$X_a = R(1 - \cos\frac{Vt}{R}) \tag{1}$$

$$Y_a = R\sin\frac{Vt}{R} \tag{2}$$

$$\theta_a = \frac{\pi}{2} - \frac{Vt}{R} \tag{3}$$

$$R = \frac{L}{\tan\phi} \tag{4}$$

$$V = 0.1m/s,\ t = 1s \tag{5}$$

$$distance = (X_a - 0)^2 + (Y_a - 0)^2 + (\theta_a - \frac{\pi}{2})^2 \tag{6}$$

Algorithm 'point pursuit' will compute the best ϕ to minimize the distance. If this movement isn't collision-free, we will decrease ϕ to find a smaller collision-free ϕ_s and increase ϕ to find a larger collision-free ϕ_l. We will choose a better ϕ between the two ϕ which makes the point A closer to point P. This algorithm is shown in Algorithm 1.

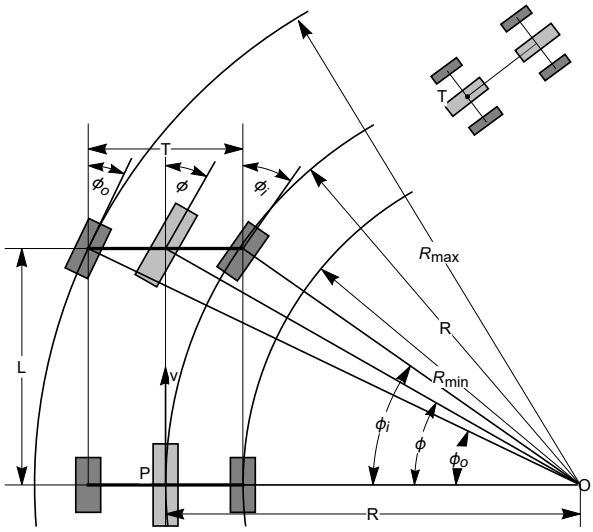


Fig. 2. Kinematic model in 'point pursuit' algorithm. It is based on Ackerman model. A car has four wheels and two equivalent wheels. The rear equivalent wheel determines the cars coordinates. In this figure, the car is positioned at the start point $(0, 0, \frac{\pi}{2})$.

B. Basic RRT

RRT was developed by Steven *et al.* [1], it is an algorithm designed to solve complex high dimensional motion planning problems by randomly building a space-filling tree. Firstly, the start point is the only point of the tree, the growing step is, we randomly sample a point P in free space and then choose the nearest point N in the tree to the sampled point P. Afterwards we find a feasible movement from point N to point A, making A and P be as close as possible. Then we add point A and the movement to the tree as a node and an edge. Repeating this process multiple times, the target point should be close enough to the tree. Then by backtracking the tree, we find a feasible path from the starting point to the target point. This algorithm is presented in Algorithm 2.

C. Model-based Target

In the basic RRT algorithm, when the distance between the car and the target is small but still larger than tolerance value θ, it takes a great amount of effort to reduce the distance even slightly. The generated trajectory will be complex, and the car will move forward and backward multiple times to adjust to a perfect position. In order to solve this problem, we put forward a model-based target, which expands the target point to a model-based tree. As shown in Fig. 3, this model-based tree has hundreds of nodes. The RRT tree needs to get close to anyone of the target tree's nodes. Then we connect the paths from RRT tree and target tree to generate the final trajectory. This expansion enables the algorithm to finish its task more easily and makes the obtained trajectory smoother. The last part of trajectory will be perfect because the target tree is generated by the well-defined model. The car will move to target position without any adjustment.

Algorithm 1 Point Pursuit

```
function POINT PURSUIT(N, P)
    movement ← argmin_distance(N, P)
    simulate_movement(movement)
    if collide then
        smaller ← movement.turning_angle
        larger ← movement.turning_angle
        while collide do
            smaller ← smaller − little
        end while
        while collide do
            larger ← larger + little
        end while
        ds ← distance(moving(smaller), P)
        dl ← distance(moving(larger), P)
        if ds < dl then
            movement.turning_angle ← smaller
        else
            movement.turning_angle ← larger
        end if
    end if
    A ← movement.destination
    E ← movement.path
    return A, E
end function
```

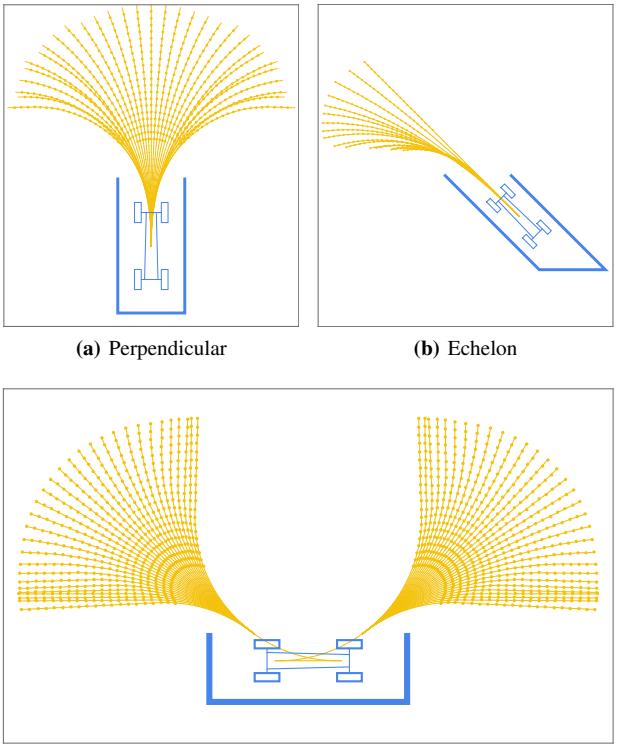
Algorithm 2 Basic RRT

```
Tree T ← startPoint
while distance(T,target_point) > tolerance_value do
    P ← random_choice( )
    N ← nearest_in_tree
    E, A ← best_movement(N,P)
    T.add (E,A)
end while
T.backtrack( )
```

To implement this algorithm, we need to generate a target tree and redefine the basic RRT's 'random choice' function. It is shown in Algorithm 3.

To generate the target tree, we divide the parking scenario into three groups: perpendicular parking, parallel parking and echelon parking. Each model is made up of a few pre-defined parking routes. To generate the models, we imagine the car is already parked within the parking lot, and then we try to drive the car out of the parking lot with various pre-defined driving strategies. After this operation, we will get dozens of feasible driving-out paths, and each path has twenty nodes. Reverse these driving-out paths to parking-in paths, there will be a tree with dozens of paths and hundreds of nodes.

- Perpendicular parking: Consider a car driving out of a perpendicular parking lot, it will go straight for sometime and then keep to a fixed turning angle. Since this angle changes from $-30°$ to $30°$ per $2°$, we will get 31 different lines. Under the premise of a collision-free parking process, we will set the straight part as short as possible. Finally, as we pick 20 points per line as nodes of the model, we will get 620 nodes in the perpendicular model.
- Parallel parking: A car is driving out of a parallel parking lot, it will move backward for a while and then keep a max turning angle until its body is half out of the parking lot. Then it will choose a fixed turning angle from $-30°$ to $30°$ per $2°$ at the rest of the path. In this part, we will also get 31 lines and 620 nodes, but considering that a car parking in a parallel parking lot would have two possible facing directions, the obtained data in our model is doubled up to 62 lines and 1240 nodes.
- Echelon parking: Similar to perpendicular parking, a car in an echelon parking lot will go straight for a while and then keep a fixed turning angle. The difference is that in this circumstance the whole movement is backward. The fixed turning angle changes from $0°$ to $30°$ per $2°$. we will finally get 16 lines and 320 nodes.



(a) Perpendicular **(b)** Echelon

(c) Parallel

Fig. 3. Model-based target tree with dozens of paths and hundreds of nodes.

D. Smoothing under Kinematic Constraint

Because of the random sampling of RRT tree nodes, there will always be some redundant nodes making the path complex. Therefore, we need an algorithm that can smooth it. There are already some smoothing algorithms for this problem, such as cubic polynomials [18], quintic polynomials [19], [20], Bezier curves [21], [22], [23], B-

Algorithm 3 Random Choice

```
r ← random(0,1)
if r < 0.5 then
    return random_point_in_free_space
else
    return nodes_in_target_tree(in proper order)
end if
```

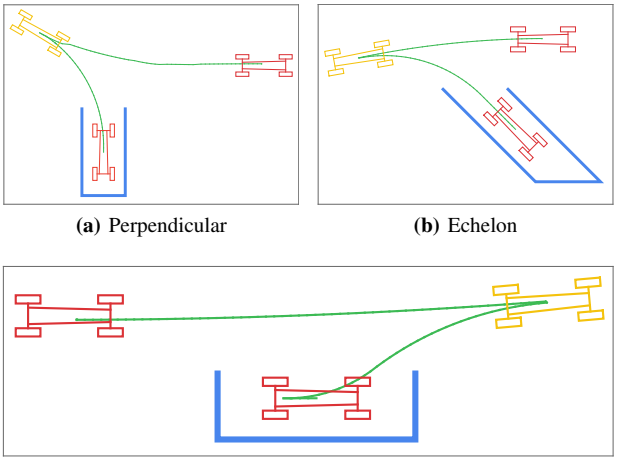
splines [24] and Clothoids [25]. Long Han *et al.* also proposed a path smoothing algorithm [2] for RRT-based parking planning. Both of these algorithms ignored the kinematic constraints of a car, which may cause the smoothed path to be unfeasible. By contrast, we have considered and worked on the vehicles kinematic constraints in our proposed smoothing algorithm, which ensures the generation of a feasible path. Our algorithm is based on the idea of Divide and Conquer. Firstly, for example, if the path has N nodes, we try to use 'point pursuit' algorithm within N steps to find a new path from start point to target point. If success, the new path is shorter than the original path. If not successful, we will divide the path into two paths from the mid-node and execute the same process to each path until success or they will have just one node. After connecting every resulting path, we will get a smoother path. The 'point pursuit' algorithm is under the car's kinematic constraints, so the generated path is feasible.

Algorithm 4 Smooth

```
function SMOOTH(path)
    if path just have one node then
        return path
    else
        new_path.add(path.start)
        while new_path is shorter than path do
            new_path.add( point_pursuit to path.end )
            if new_path reached path.end then
                return new_path
            end if
        end while
        path_1, path_2 ← divide(path)
        path_1 ← smooth(path_1)
        path_2 ← smooth(path_2)
        return connect(path_1,path_2)
    end if
end function
```

E. Parking Planning

As for parking planning, we modified the basic RRT algorithm with the model-based tree and smooth algorithm previously mentioned. Firstly, our algorithm will assign the parking task to one of three parking scenarios, and then corresponding functions are called to grow the target tree. A list of nodes in the target tree will be sent to the Algorithm 3. Finally, we will run the RRT algorithm to generate a parking path. In every step of RRT, the Algorithm 3 will choose a target point for it. RRT will check if the bias is under the tolerance value after every 1000 steps, which means that the RRT will run at least 1000 steps. If the bias is under the tolerance value, the RRT algorithm is terminated and we will start the smoothing algorithm to smooth the path before outputting it.



(a) Perpendicular **(b)** Echelon

(c) Parallel

Fig. 4. A parking path generated by our algorithm in three different parking scenarios.

Algorithm 5 Parking Planning

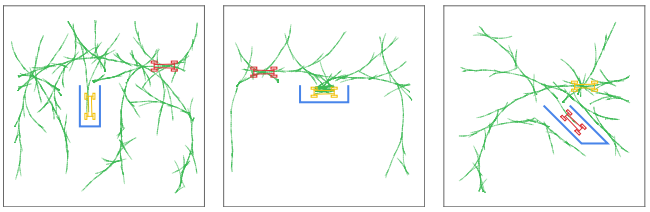
```
target_tree ← model_based_tree()
node_list ← target_tree.nodes
Tree T ← start_Point
while distance(T, target_point)> tolerance_value do
    for 1000 times do
        P ← random_choice(node_list)
        N ← nearest_node_in_tree
        E, A ← point_pursuit(N, P)
        T.add(E, A)
    end for
end while
RRT_path ← T.backtrack()
RRT_path ← smooth(RRT_path)
model_path ← target_tree.backtrack()
path ← RRT_path + model_path
return path
```

IV. Experiment

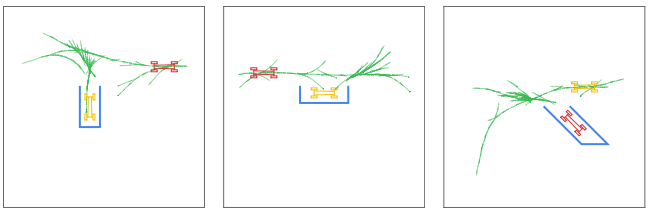
Our algorithm is programmed in C++ and executed on a personal computer with Ubuntu operating system. Our algorithm can generate the parking trajectory. We test the algorithm in three typical scenarios: perpendicular, parallel and echelon. The result is shown in Fig. 4. We will compare the result with basic RRT.

TABLE I: STEPS AND TIME COST

Scenarios	Perpendicular Parking		Parallel Parking		Echelon Parking	
Target Type	Target Point	Model-based Target	Target Point	Model-based Target	Target Point	Model-based Target
Max Steps	9000	1000	5000	1000	13000	1000
Min Steps	4000	1000	2000	1000	6000	1000
Average Steps	6250	1000	3000	1000	8750	1000
Max Time	4.85s	0.20s	2.28s	0.23s	10.30s	0.20s
Min Time	1.46s	0.12s	0.13s	0.17s	2.98s	0.17s
Average Time	2.54s	0.17s	0.92s	0.20s	5.28s	0.18s

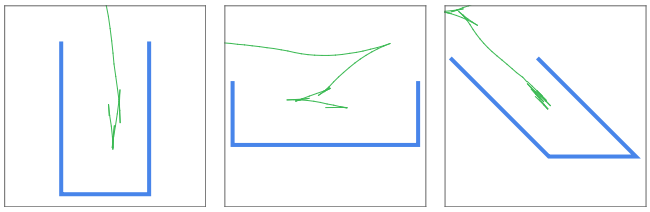


(a) Basic RRT **(b)** Basic RRT **(c)** Basic RRT

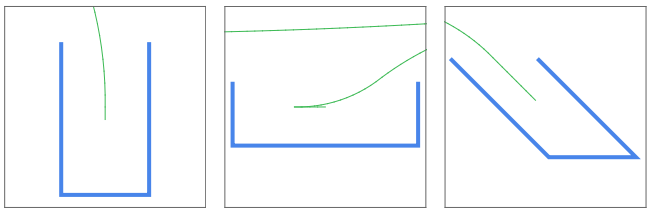


(d) Proposed method **(e)** Proposed method **(f)** Proposed method

Fig. 5. Searched space in three different parking scenarios with two algorithms. (a), (b), (c) is the searched space of basic RRT; (d), (e), (f) is the searched space of our algorithm. This searched space doesn't include the branches in target tree.



(a) Basic RRT **(b)** Basic RRT **(c)** Basic RRT



(d) Proposed method **(e)** Proposed method **(f)** Proposed method

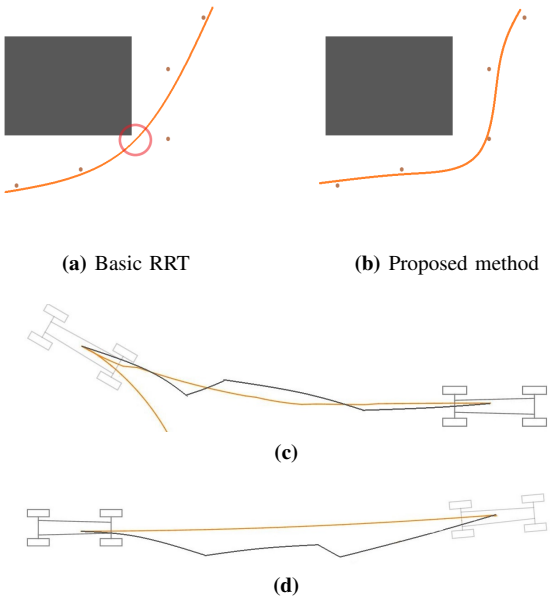
Fig. 6. The final part of the path. (a), (b), (c) is generated by basic RRT, which is very complex. The car will move forward and backward multiple times to adjust into a perfect position. (d), (e), (f) is generated by our algorithm. It is very smooth, the car can get to the target position in just one movement.

A. Model-based Target

In our experiment, there are two main advantages in the model-based target. Firstly, the target tree extends the target point out of the parking lot obstacles and expands the target point from one point to hundreds of nodes, which makes the RRT tree easier and quicker in reaching its target. Then the RRT tree needs to search a smaller space with fewer steps to get close enough to the target tree. The searched space is shown in Fig. 5. Generally, the more branches in the target tree, the smaller space needed to search, because of the reduction of search space and steps, our algorithm will have a speed boost. The steps and time cost in our algorithm compared to basic RRT is presented in Table I. After hundreds of tests, we confirmed that the basic RRT takes thousands of steps, and in some worse conditions, it takes even over ten thousands of steps. In comparison, our algorithm always takes 1000 steps and never fail. As previously mentioned, our algorithm will not stop before 1000 steps. Usually, the basic RRT costs 0.13s $\sim$ 10s for calculation, ours always takes about 0.2s.

Secondly, the final part of our parking path is more smooth and the safe margin is bigger. The reason is as follows: every RRT algorithm has a connection zone in the path. The RRT tree will connect to the target point in the connection zone. In basic RRT, the connection zone is around the target point, while in our algorithm it is the place where the RRT tree connects to the model-based target tree. The final part of parking path in basic RRT includes the connection zone, resulting in it being extremely complex. The vehicle will move forward and backward multiple times inside parking lot to adjust to a suitable position. Since it is usually narrower inside the parking lot, the adjustment may cause a collision. The final part of our algorithm is generated from the well defined model, very smooth and human-like. The vehicle will move into a perfect position in just one step.

The last part of the path is shown in Fig. 6. The connection zone of our algorithm is outside the parking lot, where it is much wider and safer. The target tree has hundreds of nodes, which makes it much easier to connect. The connection



(a) Basic RRT **(b)** Proposed method

(c)

(d)

Fig. 7. Demo of smoothing algorithm. (a) shows the path smoothed by traditional smoothing algorithm. Violently distorting the path to smoothly curve results in collision in the red zone. (b) shows the path smoothed by our algorithm. In (c), (d), the black line is the path before smooth, and the orange line is the path smoothed by our smoothing algorithm.

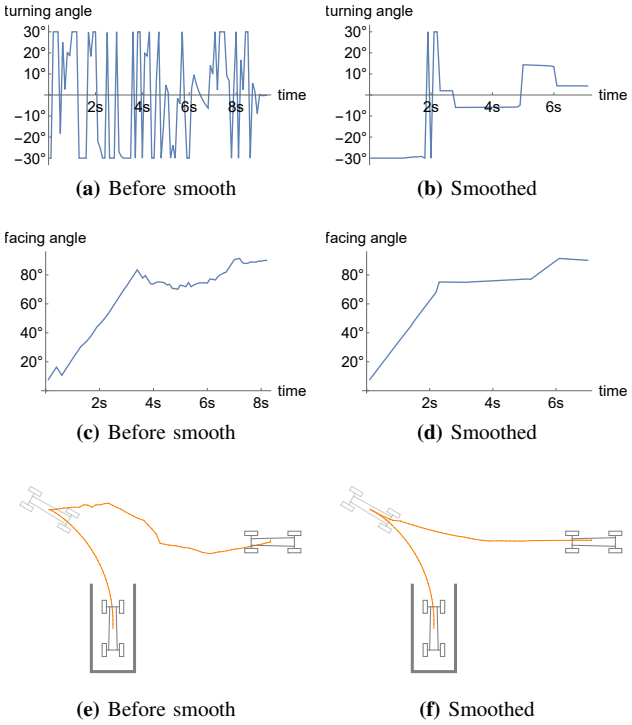
zone will also be very smooth. The connection zone of our algorithm is shown in Fig. 4.

B. Smoothing with Vehicle Kinematic Constraints

One drawback of sample-based planning algorithms is that every segment of the path is in a random direction, which makes the path complex and causes redundant nodes. As mentioned before, There are already some smoothing algorithms for it. Both of these smoothing algorithms have a common drawback, they violently distort the path into a smooth curve, while not considering the kinematic constraint. As shown in Fig. 7, the distortion can result in collision at some point. Our smoothing algorithm used the point_pursuit algorithm to re-plan a path based on the path generated by RRT. As mentioned in part III, paths smoothed by our algorithm will be feasible. The path before and after smoothing is presented in Fig. 7. We can see the path is much smoother after the procedure. From the car's moving perspective, in Fig. 8, when it is maneuvering in the pre-smoothed path, the turning angle of the front wheel will shake between $-30°$ and $30°$, the facing angle of the car will also shake. But these two angles will be far more stable and human-like in the smoothed path.

C. Complex Parking Scenarios

We have tested our algorithm in various complex parking scenarios. Including narrow slot of highly limited space. As shown in Fig. 9, our algorithm works very well in such an environment.

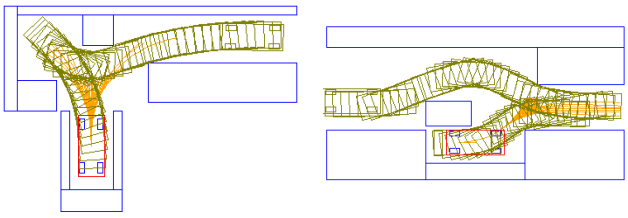


(a) Before smooth **(b)** Smoothed

(c) Before smooth **(d)** Smoothed

(e) Before smooth **(f)** Smoothed

Fig. 8. Turning angles and facing angles of the car during parking. (a), (b) show the turning angles of the front wheel before and after smoothing procedure. (c), (d) show the facing angles of the car before and after smoothing procedure. (e), (f) show the corresponding path. Note that the path generated from target tree does not need to be smoothed. We will add some constrains in future to control the abrupt change of turning angle like in (b).



(a) **(b)**

Fig. 9. This figure shows some paths planed by our algorithm in super narrow parking scenarios. The red rectangle is the destination.

V. Conclusion and Future Works

This paper introduced imagination into RRT algorithm to expand the target point to a model-based target tree, which makes the algorithm quicker and the generated trajectory smoother. We also proposed a smoothing algorithm considering vehicle kinematic constraints to generate feasible trajectory.

Our method still has some drawbacks, for example our car has to drive with a fixed speed. In the future we will use deep

reinforcement learning to generalize the vehicle speed to a continuous value and explore an unknown space to locate an available parking lot. We will also test our algorithm in the real world.

REFERENCES

[1] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," Tech. Rep., 1998.

[2] L. Han, Q. H. Do, and S. Mita, "Unified path planner for parking an autonomous vehicle based on rrt," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 5622–5627.

[3] S. M. Lavalle, J. J. Kuffner, and Jr., "Rapidly-exploring random trees: Progress and prospects," in *Algorithmic and Computational Robotics: New Directions*, 2000, pp. 293–308.

[4] Y. Song and C. Liao, "Analysis and review of state-of-the-art automatic parking assist system," in *Vehicular Electronics and Safety (ICVES), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1–6.

[5] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of mathematics*, vol. 79, no. 3, pp. 497–516, 1957.

[6] J. Reeds and L. Shepp, "Optimal paths for a car that goes both forwards and backwards," *Pacific journal of mathematics*, vol. 145, no. 2, pp. 367–393, 1990.

[7] D. González, J. Pérez, V. Milanés, and F. Nashashibi, "A review of motion planning techniques for automated vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 4, pp. 1135–1145, 2016.

[8] J. Bohren, T. Foote, J. Keller, A. Kushleyev, D. Lee, A. Stewart, P. Vernaza, J. Derenick, J. Spletzer, and B. Satterfield, "Little ben: The ben franklin racing team's entry in the 2007 darpa urban challenge," *Journal of Field Robotics*, vol. 25, no. 9, pp. 598–614, 2008.

[9] A. Bacha, C. Bauman, R. Faruque, M. Fleming, C. Terwelp, C. Reinholtz, D. Hong, A. Wicks, T. Alberi, D. Anderson *et al.*, "Odin: Team victortango's entry in the darpa urban challenge," *Journal of field Robotics*, vol. 25, no. 8, pp. 467–492, 2008.

[10] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke *et al.*, "Junior: The stanford entry in the urban challenge," *Journal of field Robotics*, vol. 25, no. 9, pp. 569–597, 2008.

[11] P. Petrov and F. Nashashibi, "Modeling and nonlinear adaptive control for autonomous vehicle overtaking," *IEEE Transactions on Intelligent Transportation Systems*, vol. 15, no. 4, pp. 1643–1656, 2014.

[12] S. Chen, S. Zhang, J. Shang, B. Chen, and N. Zheng, "Brain-inspired cognitive model with attention for self-driving cars," *IEEE Transactions on Cognitive and Developmental Systems*, 2017.

[13] S. Chen, J. Shang, S. Zhang, and N. Zheng, "Cognitive map-based model: Toward a developmental framework for self-driving cars," in *Intelligent Transportation Systems (ITSC), 2017 IEEE 20th International Conference on*. IEEE, 2017, pp. 1–8.

[14] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Path planning for autonomous vehicles in unknown semi-structured environments," *The International Journal of Robotics Research*, vol. 29, no. 5, pp. 485–501, 2010.

[15] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[16] 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 Transactions on Control Systems Technology*, vol. 17, no. 5, pp. 1105–1118, 2009.

[17] A. Yershova, L. Jaillet, T. Siméon, and S. M. LaValle, "Dynamic-domain rrts: Efficient exploration by controlling the sampling domain," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*. IEEE, 2005, pp. 3856–3861.

[18] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann *et al.*, "Stanley: The robot that won the darpa grand challenge," in *The 2005 DARPA Grand Challenge*. Springer, 2007, pp. 1–43.

[19] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer *et al.*, "Autonomous driving in urban environments: Boss and the urban challenge," in *The DARPA Urban Challenge*. Springer, 2009, pp. 1–59.

[20] E. Papadopoulos, I. Poulakakis, and I. Papadimitriou, "On path planning and obstacle avoidance for nonholonomic platforms with manipulators: A polynomial approach," *the International Journal of Robotics research*, vol. 21, no. 4, pp. 367–383, 2002.

[21] K. Yang and S. Sukkarieh, "An analytical continuous-curvature path-smoothing algorithm," *IEEE Transactions on Robotics*, vol. 26, no. 3, pp. 561–568, 2010.

[22] K. Jolly, R. S. Kumar, and R. Vijayakumar, "A bezier curve based path planning in a multi-agent robot soccer system without violating the acceleration limits," *Robotics and Autonomous Systems*, vol. 57, no. 1, pp. 23–33, 2009.

[23] K. Yang, D. Jung, and S. Sukkarieh, "Continuous curvature path-smoothing algorithm using cubic b zier spiral curves for non-holonomic robots," *Advanced Robotics*, vol. 27, no. 4, pp. 247–258, 2013.

[24] T. Maekawa, T. Noda, S. Tamura, T. Ozaki, and K.-i. Machida, "Curvature continuous path generation for autonomous vehicle using b-spline curves," *Computer-Aided Design*, vol. 42, no. 4, pp. 350–359, 2010.

[25] Y. Kanayama and B. I. Hartman, "Smooth local path planning for autonomous vehicles," in *Autonomous robot vehicles*. Springer, 1990, pp. 62–67.