2012 IEEE International Conference on Robotics and Automation
RiverCentre, Saint Paul, Minnesota, USA
May 14-18, 2012

A Real-Time Motion Planner with Trajectory Optimization for Autonomous Vehicles

Wenda Xu, Junqing Wei, John M. Dolan, Huijing Zhao and Hongbin Zha

Abstract— **In this paper, an efficient real-time autonomous driving motion planner with trajectory optimization is proposed. The planner first discretizes the plan space and searches for the best trajectory based on a set of cost functions. Then an iterative optimization is applied to both the path and speed of the resultant trajectory. The post-optimization is of low computational complexity and is able to converge to a higher-quality solution within a few iterations. Compared with the planner without optimization, this framework can reduce the planning time by 52% and improve the trajectory quality. The proposed motion planner is implemented and tested both in simulation and on a real autonomous vehicle in three different scenarios. Experiments show that the planner outputs high-quality trajectories and performs intelligent driving behaviors.**

I. INTRODUCTION

A. Background

In the last few decades, researchers have put considerable effort into autonomous driving. Autonomous vehicles have great potential to improve the performance and safety of the transportation system. They can also free people from the task of driving, which could save commuters considerable time daily. To achieve this objective without affecting existing human drivers on the road, autonomous vehicles need to have human-acceptable driving performance. The planner also needs to meet strict real-time requirements to react fast enough in emergency situations. In summary, it is important, but difficult, to develop a practical high-performance real-time motion planner for on-road driving.

B. Related work

Autonomous driving Systems: The NAVLAB project at Carnegie Mellon University (CMU) has built a series of experimental platforms which are able to run autonomously on freeways [1]. In 2007, the DARPA Urban Challenge provided researchers a practical scenario in which to test the latest sensors, computer technologies and artificial intelligence algorithms [2]. Basic interaction between autonomous vehicles and human-driven vehicles was proven in low-density, low-speed traffic. Most planners in the competition were designed aggressively to win the race, rather than being focused on the trajectory's quality and human acceptance.

In recent years, commercial driving safety assistance systems such as adaptive cruise control and lane assist systems have been widely used in high-end volume-produced cars. These systems are helpful in reducing accidents caused by distractions, drowsiness or driver error. However, they cannot perform complex driving behavior such as dealing with merging vehicles, circumventing other cars, or responding intelligently to unexpected dynamic obstacles. Also, these systems still need constant human supervision.

Trajectory generation: Trajectory generation for autonomous vehicles in road scenarios needs to consider three constraints: kinematic, dynamic, and road shape. More specifically, the rate of change of curvature and acceleration should be continuous in the commanded trajectory to make sure the car can execute it. The paths should also conform to the road shape.

Kelly and Nagy ([3], [4]) propose an inverse path generation method, using curvature polynomials to ensure continuous rate of change of curvature. Based on Kelly and Nagy's method, McNaughton et al. ([5], [6]) present a planner that first samples endpoints along the road and then connects them using curvature polynomials to make all paths conform to the road shape. After that, a set of trajectories is generated by specifying different acceleration profiles for each path. Paths generated in [6] can be tracked very well by an autonomous vehicle. But because the acceleration profile is not continuous, it is hard for the vehicle to follow the profile accurately and smoothly.

Werling et al. [7] deal with this problem using an alternative method. They generate the lateral and longitudinal trajectory using quintic polynomials versus time, which ensures continuous acceleration. In addition, they use a Frenet Frame referenced to the road center line to combine lateral and longitudinal motion. This makes the trajectory longitudinal coincide with the road shape. However, the curvature of every point on each trajectory needs to be computed and verified, which is computationally expensive. Additionally, although the curvature is continuous, the sign of the first derivative of curvature (moving direction of the steering wheel) changes very frequently, which leads to jerky steering wheel movement.

Search algorithm: After trajectories are generated, search algorithms are often applied to find the optimal result. Searches in state lattice planners are usually based on heuristics (e.g. A* and ARA* [8]) or sampling (e.g. RRT [9]). For heuristic-based algorithms, a good estimate of cost

This work was supported by NSF Grant CNS1035813 and NSFC Grants No.90920304 and No.60975061.

Wenda Xu, Huijing and Hongbin Zha are with School of Electronics Engineering and Computer Science, Peking University, Beijing, China {xuwenda, zhaohj, zha}@cis.pku.edu.cn

Junqing Wei is with the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213, USA junqingw@cmu.edu

John M. Dolan is with the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA jmd@cs.cmu.edu

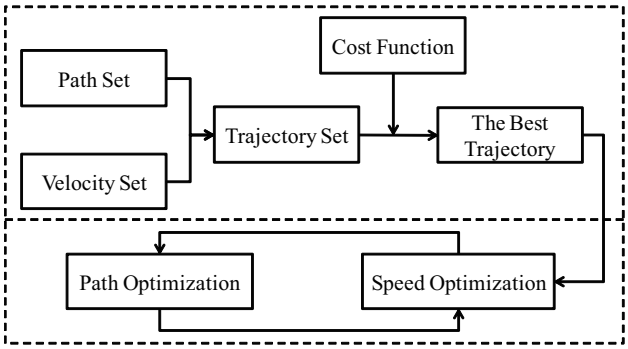


Fig. 1: Two-part trajectory planning framework.

from any vertex to the goal is essential. But the planning problem for autonomous driving is complicated. Especially when there are dynamic obstacles, it is very difficult to find an appropriate heuristic that is suitable for all scenarios.

Some incremental replanning algorithms (e.g. D* Lite [10]) are widely used in robotics navigation, but they only work for typical planning problems with one fixed goal, while in on-road scenarios the goal is changing all the time. For sampling-based methods (e.g. RRT [9]) the results are often not smooth enough for the car to execute.

In addition, because of the existence of dynamic obstacles, time needs to be included in the search state space as an additional dimension. This makes the search space exponentially increase, which leads to a low efficiency of the search-based algorithm. Therefore, in this paper we apply a discretization of the state space, then simply apply a straightforward and fast exhaustive search.

Optimization method for planning: Dolgov et al. [11] present a conjugate gradient method to smooth the prior result, which is the path. However, the speed of the vehicle is not planned or optimized. Therefore, this algorithm does not work well in on-road environments, where speed is very important for driving, especially dealing with dynamic obstacles. Zucker et al. [12] propose a gradient optimization method for motion planning to better avoid static obstacles. However, their method also does not take the time dimension or speed into account.

C. Contribution

Based on related work, a practical real-time motion planner is proposed in this paper. The planner has the following features: trajectories generated by the planner are smooth and continuous, and therefore kinematically feasible for the vehicle to execute; the search for an optimal trajectory is accelerated by efficient path and speed discretization; the performance sacrificed by discretization is compensated by a post-optimization process; the real-time performance is improved by iteratively optimizating in both path and speed space.

The algorithm framework is introduced in Section II. Section II-A describes the trajectory generation method. A set of cost functions that leads the planner to perform reasonable and good driving behavior is introduced in Section II-B. The iterative trajectory optimization mechanism is described and evaluated in Section II-C and Section II-D. Implementation details of the motion planner in a real autonomous vehicle are given in Section III. Section IV focuses on motion planner performance evaluation in both simulation and on-road testing.

II. Algorithm Framework

The framework of the proposed real-time motion planner is shown in Fig. 1. It consists of two parts, trajectory planning and trajectory optimization. In the first step, the path edges are generated using the method described in [3]. Speed sets are then generated for each path edge. After that, a set of cost functions is applied to each trajectory and the best trajectory is selected. The resultant trajectory is then passed to the optimization module, where the path and speed are iteratively optimized using a randomly-oriented simplex optimization algorithm [13].

A. Trajectory generation

Path and speed edges are generated separately. Then, by combining the path edges with the speed edges, a trajectory set is obtained.

1) Path generation: Paths are generated by connecting sampled endpoints using different kinds of curvature polynomials. The sampling method in our planner is the same as that in [5]. However, instead of cubic, quartic curvature polynomials are used to ensure that the curvature change rate at the start point of each planning cycle is continuous.

a) Endpoint sampling: To generate the vertices (endpoints) for each path, a sampling mechanism is implemented. The function for the road center line is given as:

$$\vec{r}(\ell) = [x_r(\ell)\ y_r(\ell)\ \theta_r(\ell)\ \kappa_r(\ell)] \tag{1}$$

In (1), ℓ is the longitudinal offset, also called station. We can define a point $\vec{p}(\ell, d)$ away from the road center at a given longitudinal offset ℓ and lateral offset d as $\vec{p}(\ell, d) = [x_p(\ell, d)\ y_p(\ell, d)\ \theta_p(\ell, d)\ \kappa_p(\ell, d)]$, where

$$\begin{aligned} x_p(\ell, d) &= x_r(\ell) + d\cos(\theta_r(\ell) + \tfrac{\pi}{2}) \\ y_p(\ell, d) &= y_r(\ell) + d\sin(\theta_r(\ell) + \tfrac{\pi}{2}) \\ \theta_p(\ell, d) &= \theta_r(\ell) \\ \kappa_p(\ell, d) &= (\kappa_r(\ell)^{-1} - d)^{-1} \end{aligned} \tag{2}$$

Using (2), for each layer we sample N_{path} endpoints perpendicular to the center line.

b) Path model: Paths are generated by connecting: 1) pairs of sampled endpoints; 2) endpoints and the current vehicle pose. There are two kinds of path models used by this motion planner: cubic and quartic curvature polynomials. For cubic curvature polynomials, the curvature of the path is a cubic polynomial function of arc length.

$$k(s) = \gamma_0 + \gamma_1 s + \gamma_2 s^2 + \gamma_3 s^3 \tag{3}$$

So the problem becomes to find the parameters satisfying the endpoint constraints. There are 5 parameters in the function: γ_0, γ_1, γ_2, γ_3, and s. The corresponding five constraints are relative x,y motion and orientation difference from the start

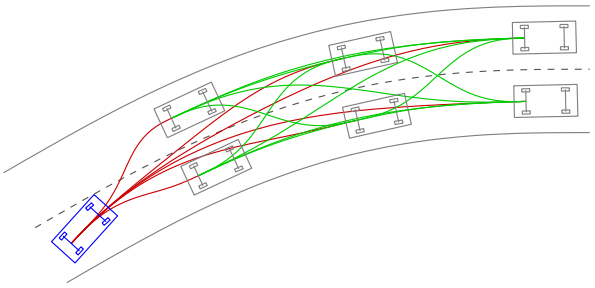


Fig. 2: Path set. The blue vehicle represents the current vehicle pose, and grey vehicles represent sampled endpoints. The red paths are quartic curvature polynomials and the green paths are cubic ones.

point to the end point, the start point curvature and the end point curvature. Following [3], we use a gradient descent algorithm to solve this problem.

In the previous work [5], only cubic curvature polynomials are used for common scenarios. This method works well in any individual planning cycle, where the rate of change of curvature is always continuous. However, a discontinuity may occur at the junction point of two plan cycles. The motion planner is replanning at a very high frequency. As a result, the curvature of the actual planning result may be far from smooth. Fig. 3 shows that a typical result using quartic curvature polynomials (red solid line) is much smoother than that using cubic polynomials (blue dashed line).

Hence, for the paths between the current vehicle pose and endpoints, a new constraint needs to be added, i.e., the first derivative of curvature at the current vehicle pose point. To satisfy this additional constraint, the polynomial needs to be quartic rather than cubic. This improvement leads to a smooth path even when we are doing frequent replanning as shown in Fig. 2, in which red paths represent quartic curvature polynomials and green paths are cubic ones. However, quartic polynomials take longer to generate. Therefore, to limit computation time, quartic polynomials are only used for trajectory segments starting from the current vehicle pose.

2) Speed generation: After the path edges are generated, candidate speed profiles are built for each individual path. Unlike many previous works, such as [5] and [14], that use a forward method to generate speed profiles, our work uses an inverse method. That is, the speed space is first discretized, and then a polynomial is generated to satisfy

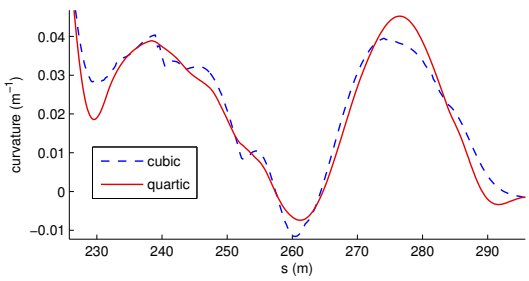


Fig. 3: Quartic curvature polynomial (red solid) vs. cubic curvature polynomial (blue dashed).

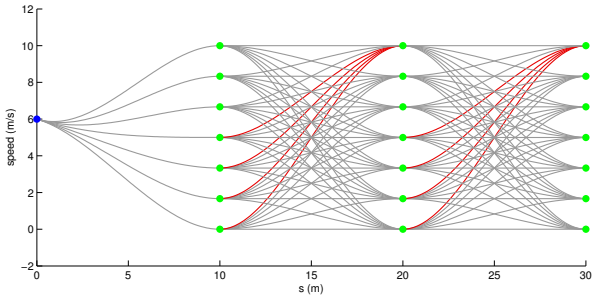


Fig. 4: Speed set. The green points are sampled speeds, the red curves are beyond the acceleration limit, and the grey curves are valid ones.

the vertex constraints. Our method is also different from the inverse method proposed in [15] and [7], which use polynomial functions of time to generate speed profiles. In the implementation, we use polynomial functions of arc length, which is more consistent with our path generation method. The polynomial equation is:

$$v(s) = \rho_0 + \rho_1 s + \rho_2 s^2 + \rho_3 s^3 \tag{4}$$

Speed state is defined as $\mathcal{Q} = (s, v, a)$, where s is arc length, v is speed, and a is acceleration. For each path, we choose the start speed state $\mathcal{Q}_{init} = (s_0, v_0, a_0)$ from the start point of the path, and end speed state $\mathcal{Q}_{goal} = (s_1, v_1, a_1)$ from the end point of the path. In (4), there are four unknown parameters ρ_0, ρ_1, ρ_2 and ρ_3, and the corresponding four constraints are v_0, a_0, v_1 and a_1. For all vertices, v_0 and v_1 are from corresponding discretized vertices, and a_0 and a_1 are set to 0. The vehicle current state (v_0, a_0) is special in that it is obtained from real vehicle sensors. Denoting maximum and minimum speed as v_{max} and v_{min}, respectively, and the number of discretized speeds as N_{speed}, the discretized speeds can be found as follows:

$$v_i = v_{min} + \frac{v_{max} - v_{min}}{N_{speed} - 1} i, \;\; i = 0, 1, \ldots, n_v - 1 \tag{5}$$

By setting $s_0 = 0$, the parameters can be obtained:

$$\begin{aligned} \rho_0 &= v_0 \\ \rho_1 &= a_0 \\ \rho_2 &= -\frac{2a_0}{s_1} - \frac{a_1}{s_1} - \frac{3v_0}{s_1^2} + \frac{3v_1}{s_1^2} \\ \rho_3 &= \frac{a_0}{s_1^2} + \frac{a_1}{s_1^2} + \frac{2v_0}{s_1^3} - \frac{2v_1}{s_1^3} \end{aligned} \tag{6}$$

The discretization of speed and candidate speed profiles is shown in Fig. 4, where the green points are sampled speeds, the red curves are beyond the acceleration limit, and the grey curves are valid ones. This speed generation method ensures continuous acceleration.

B. Cost function set

For each trajectory, we define both static cost and dynamic cost to evaluate the safety, comfort, efficiency, energy consumption and behavior. We extract n points from each

TABLE I: Static Costs

Cost	Formula	Physical Interpretation	Impact
c_ℓ	ℓ	ℓ is path length	*efficiency*
c_k	$\sum_{i=0}^{n} \lvert k_i \rvert$	k_i is curvature	*comfort*
c_{dk}	$\sum_{i=0}^{n} \lvert \dot{k}_i \rvert$	$\dot{k}_i$ is rate of change of curvature	*comfort*
c_o	$\sum_{i=0}^{n} \lvert o_i \rvert$	o_i is lateral offset with the closest center line	*behavior*
c_{obs}^s	$\sum_{i=0}^{n} s_i$	s_i is the transformed distance to static obstacles (see (7))	*safety*

TABLE II: Dynamic Costs

Cost	Formula	Physical Interpretation	Impact
c_t	t	t is time duration of a trajectory	*efficiency*
c_e	$\sum_{i=0}^{n} v_i^2$	v_i is speed	*energy*
c_a	$\sum_{i=0}^{n} a_i^2$	a_i is acceleration	*comfort*
c_j	$\sum_{i=0}^{n} j_i^2$	j_i is jerk (the rate of change of acceleration)	*comfort*
c_{ca}	$\sum_{i=0}^{n} v_i^2 k_i$	c_{ca} is centripetal acceleration	*comfort*
c_{obs}^d	$\sum_{i=0}^{n} d_i$	d_i is transformed distance to dynamic obstacles (see (8))	*safety*

trajectory to represent the cost. The formulas for static and dynamic costs are shown in Table I and Table II, respectively.

The static and dynamic obstacle cost c_{obs}^s and c_{obs}^d are also used to perform a collision check for each candidate trajectory. Based on the method proposed by [16], we use M_{cir} circles to cover the area of the vehicle. If the distance from the circle to the obstacle is smaller than a threshold $d_{minAllowed}$, then the cost is infinite. Otherwise, the cost is computed using the equation below, where λ_{obs}^s and λ_{obs}^d are the bandwidth of exponential cost functions for static and dynamic obstacles, respectively, and g_j is the distance between the obstacle and j^{th} circle covering the car.

$$s_i = \sum_{j=1}^{M_{cir}} \exp(-\frac{1}{\lambda_{obs}^s} g_j) \tag{7}$$

$$d_i = \sum_{j=1}^{M_{cir}} \exp(-\frac{1}{\lambda_{obs}^d} g_j) \tag{8}$$

The total cost for one trajectory is the weighted sum of all terms:

$$c_{total} = w_\ell c_\ell + w_k c_k + \cdots + w_{ca} c_{ca} + w_{obs}^d c_{obs}^d \tag{9}$$

C. Trajectory optimization

For most lattice planners, proper discretization is necessary to ensure real-time performance. However, this affects the optimality of the planning result. To make autonomous driving perform like human drivers, the optimality and quality of the trajectory are important, so we seek to post-optimize the trajectory to improve performance.

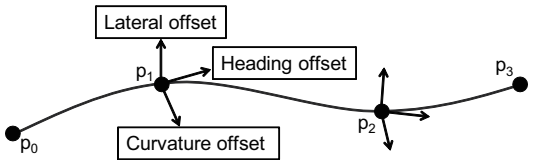


Fig. 5: Path optimization process. Relaxing lateral offset, heading and curvature for in-between sampled endpoints (p_1 and p_2), and generating new paths between the new endpoints.

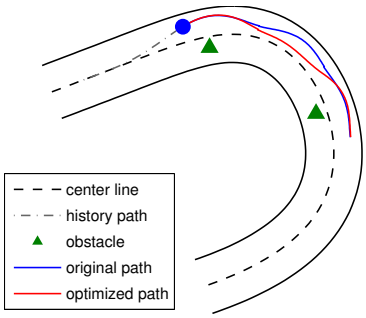


Fig. 6: Result for path optimization.

A straightforward approach is to optimize the trajectory path and speed simultaneously. However, this is time-consuming for a real-time application. The computation expense is $O(opt(M+N))$, where M is the number of path parameters, N is the number of speed parameters, and $opt()$ is the computational complexity of the optimization algorithm, which is usually $O(N^2)$. As the dimension increases, it is also harder for the optimizer to find the global optimal solution.

Therefore, we propose an iterative trajectory optimization mechanism. Its computational complexity is $O(opt(M)) + O(opt(N))$. If the trajectory found in the planning phase is close to a real optimum, using this mechanism allows us to converge to this real optimum in a few iterations.

1) Path optimization: The path discretization limits the optimality of the path. For example, the lateral offset, heading and curvature for the sampled endpoints are fixed and have some relationship to the center line. Therefore, relaxing these constraints (see Fig. 5) and generating new paths between the new endpoints allows the trajectory quality to be improved. Since the gradient for cost versus lateral offset and heading is very hard to compute, a non-derivative optimization algorithm, the Simplex algorithm [17], is used for path optimization. As shown in Fig. 6, the path after path optimization (red line) is smoother.

2) Speed optimization: The speed discretization and the constraint of the acceleration at endpoints also limit the optimality of the speed profile. Similar to path optimization, parameters of the speed profile nodes are being optimized. For the nodes that generate the current lowest-cost speed profile, we optimize their values of speeds and accelerations. Therefore, connecting the new nodes allows the speed profile to remain smooth.

In this optimization, the speed changes at nodes will affect

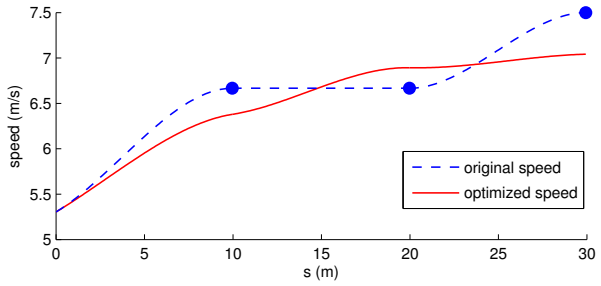


Fig. 7: Result for speed optimization.

the time of each point on the trajectory, which changes the position of dynamic obstacles on the trajectory and prevents the use of the gradient method. Therefore, the non-derivative Simplex algorithm is used for speed optimization, too. The speed optimization result is shown in Fig. 7. Because the constraints for endpoints' acceleration are relaxed, the speed change is smoother and the amplitude of the speed at endpoints are not disceitized any more.

D. Optimization performance evaluation

In this section, we analyze the performance of the proposed iterative trajectory optimization algorithm. There are two performance evaluation matrics: average trajectory quality C_{ave} (represented by the cost defined in Section II-B), and total planning time T_{total}, which is the sum of planning time and optimization time. Table III shows the performance comparison and the weights for static and dynamic costs defined in Table IV. The tests are implemented on an x86 PC with Intel i5 760 (quad core 2.8GHz) and 4GB memory.

Tests #1 to #4 show the performance using different path and speed discretization granularity, in which N_{path} and N_{speed} are the number of samples in each layer, amd C_{std} is the average cost of test #1. We can see that when the granularity is finer (N_{path} or N_{speed} is larger), the cost is lower, which means the quality of trajectory is improved. But at the same time, the planning time increases dramatically.

In tests #5 to #8, iterative optimization is applied with 1 to 4 iterations, respectively. The results show that, using iterative optimization, within a few iterations, the trajectory quality can be improved a lot more than by using finer discretization. The optimization converges within 3 iterations. The additional optimization time is proportional to the number of iterations, which is more acceptable.

In test #9, instead of iteratively optimizing the path and speed, we optimize them at the same time. The result shows that the quality of trajectory is better than without optimization. But it is not as efficient as iterative optimization. That's becasue this optimization is of higher computational complexity $O((M+N)^2)$. The larger optimization space with both speed and path in it also has many more local optima, which are difficult for the optimzer to overcome.

In summary, experiments show that the proposed planner with an iterative optimization framework is very promising. It is able to generate a higher-quality trajectory within much shorter time compared to alternative planning mechanisms or configurations. Compared with non-optimization result from test #2, the iterative optimization result from test #7 is of 52% reduction in time and 10% improvement in quality. In our implementation, we use the configuration of test #7 based on the result from Table III. The planner is running at 8Hz.

TABLE III: Performance Comparison

Num	Iteration	N_{path}	N_{speed}	C_{ave}/C_{std}	T_{total} (ms)
#1	0	7	5	1.000	85.60
#2	0	11	5	0.949	233.00
#3	0	7	9	0.997	301.05
#4	0	11	9	0.937	888.20
#5	1	7	5	0.988	92.33
#6	2	7	5	0.856	108.21
#7	3	7	5	0.853	112.08
#8	4	7	5	0.853	136.13
#9	non-iterative	7	5	0.867	162.00

TABLE IV: Weights

Static costs	Weight	Dynamic costs	Weight
w_l	1	w_t	10
w_k	10	w_e	1
w_{dk}	10	w_a	0.1
w_o	10	w_j	0.1
w^s_{obs}	0.01	w_{ca}	0.1
		w^d_{obs}	0.1

III. System Implementation

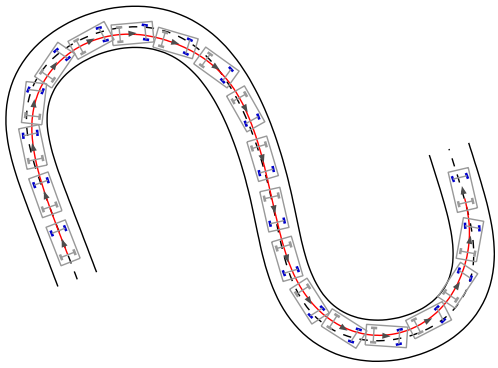
A. Interfacing with Autonomous Vehicle

An autonomous vehicle from Carnegie Mellon University's autonomous driving laboratory was used to test the motion planner. The vehicle is equipped with a high-fidelity localization system. It also uses lidar and radar to perceive the real-time surrounding environment. The vehicle's lower-level control is designed to perform high-accuracy trajectory tracking. Therefore, the motion planner only needs to generate an executable trajectory with both a path and speed profile.

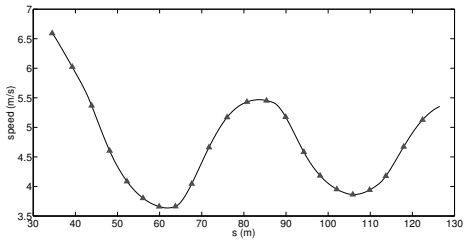
The interface between planner and lower-level controller is built so that during each plan cycle, the first partition of the trajectory, which is 8-50 meters depending on the vehicle speed, is sent. This gives the lower-level controller enough look-ahead to perform a predictive control algorithm. It also increases the reliability of the system, because the lower-level controller always has a relatively long trajectory to execute, even if the higher-level computer stops working for a few cycles.

B. Robust replan mechanism

To react to a dynamically changing environment in the real world, the motion planner needs to replan continually. If the planner starts to plan from the current vehicle state, then when the planning is finished, usually around 100 milliseconds later, the vehicle will be at a different position, and the original plan will no longer be valid. To solve this problem, [18] propose a PMP (partial motion planning) scheme. In short, their approach is to plan from the future

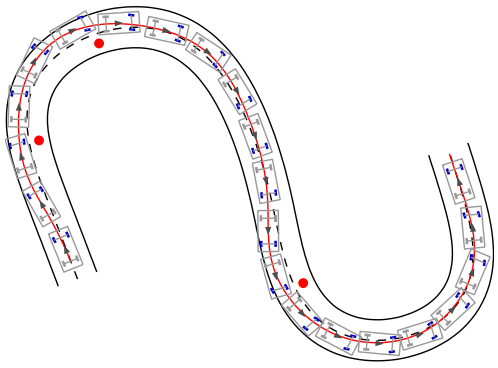


(a)

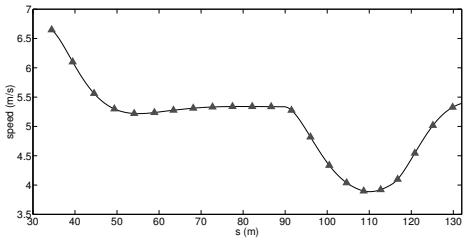


(b)

Fig. 8: Result for lane driving.



(a)



(b)

Fig. 9: Result for lane driving with static obstacles.

state. When the planner is running, the vehicle executes a trajectory from the last plan cycle.

Based on [18], a queue in ascending order of arc length is implemented to preserve the trajectory from the last plan cycle. For every cycle, first a point P_{close} with arc length s_c which is closest to the vehicle's position is found in the queue. Then the planner looks for point P_{future} with arc length s_f in the queue which is at a later position compared to P_{close}. The equation to calculate s_f is:

$$s_f = s_c + v_{curr} t_{span} + \frac{1}{2} a_{max} t_{span}^2 \tag{10}$$

where v_{curr} is the vehicle's current speed, a_{max} is the maximum acceleration the vehicle can reach, and t_{span} is the basic time span between the current time and the start time of next replan, which should be longer than the replan interval. This equation ensures the planning is finished before its result is needed. Finally, the planner replans from P_{future} and updates the trajectory queue using the latest planned trajectory.

A special case is that when the vehicle is in manual driving mode, instead of planning from a future position P_{future}, it replans from its current state (position and speed). This ensures that the vehicle always has a feasible trajectory to execute when switching from manual to autonomous mode, even when the vehicle is moving. The smoothness of transition between autonomous and manual mode is also satisfactory in real-world tests.

IV. Experimental Results

To test the performance of the motion planner, we implemented three different test categories: lane driving, static obstacles and dynamic obstacles. The proposed motion planner was tested both in simulation for all three scenarios, and on the real vehicle except for dynamic obstacle scenario.

A. Lane driving

The lane driving test was performed on an S-shaped curved road. This test was implemented to verify the motion planner's ability to deal with sinuous driving, where human drivers are often able to select a shorter and smoother route. The result is shown in Fig. 8. The autonomous vehicle moves on the inner part of the road to get a shorter path length, and it slows down when entering and speeds up when exiting curves. This lane driving test was also implemented on the real vehicle running on the exact same map. The autonomous vehicle was able to track the trajectory generated by the planner with an average cross track error less than $10cm$, maximum tracking error of around $40cm$ and average speed error around 0.5m/s. The road test verifies that the performance of the whole autonomous driving system. It also shows that the trajectory generated by the planner is feasible for the car to execute.

B. Static obstacles

In a road environment, autonomous drivers need to deal with multiple static obstacles, including curbs, parked cars, and road blockages. In this planner, undrivable areas on the road, e.g. broken pavement, are also modeled as static obstacles. Here we test the planner's performance of dealing with a curved road and multiple static obstacles together. The result is shown in Fig. 9, and the red points are static obstacles. To avoid the static obstacles, instead of doing short-cuts on curves, the autonomous vehicle selected a longer, but still smooth, path.

C. Dynamic obstacles

For autonomous vehicles, dynamic obstacles are usually moving obstacles such as cars, pedestrians, bicyclists or motorcyclists. As shown in Fig. 10, in this test the vehicle encounters a slower car in its preferred (upper) lane. Because the slower vehicle in front of it limits the autonomous vehicles progress, the planner selects a lane change trajectory to circumvent it. The autonomous vehicle speeds up at the beginning of the pass, then keeps approximately uniform speed while passing the slower car, and speeds up after the pass. Because of the lateral distance cost, when it is overtaking the slower car, it also tends to keep in the center of the circumventing lane, which is reasonable.

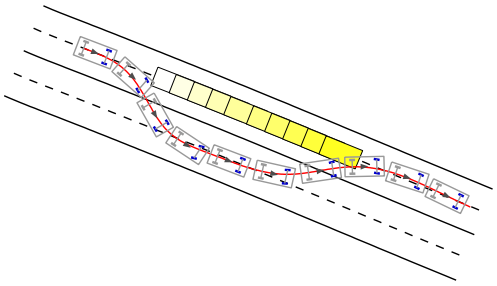
V. Conclusion

In this paper, a practical real-time autonomous driving motion planner with trajectory optimization is proposed and implemented. The trajectories generated by the planner are smooth and continuous so that the autonomous vehicle is able to execute with very small path and speed tracking error. A proper discretization is applied to the speed and path space, which makes the search for an optimal trajectory faster. To further improve the quality of the trajectory, an iterative optimization on the speed and path state space is designed and implemented. Experiments show that, with the iterative optimization framework, the performance of the resultant trajectory is improved by 10% and the planning time is reduced by more than 50%. The planner has been tested in simulation and a real vehicle in three scenarios. Its ability to deal with a sinuous road with sharp turns, avoid multiple on-road static obstacles, and perform lane changing and circumvention of slower cars was verified.

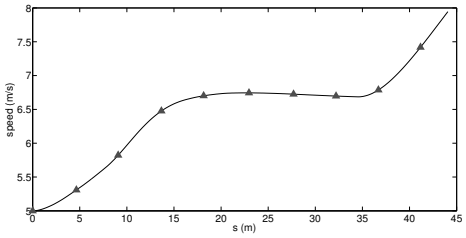
For future work, more on-road experiments need to be done to verify the planner's performance in dealing with complicated real traffic scenarios. The performance of the planner can also potentially be improved by applying more efficient pruning in both the speed and path space. Further, though the cost functions determine the choice of the final trajectory, it is difficult and subjective to find the appropriate form and weight for cost functions manually. Therefore, future studies will also focus on learning cost functions from human driver demonstrations.

VI. Acknowledgments

The authors gratefully acknowledge the help of Jarrod Snider, Tianyu Gu and Matthew McNaughton.



(a)



(b)

Fig. 10: Result for circumventing a slower car.

References

[1] D. Pomerleau, "ALVINN: An autonomous land vehicle in a neural network," *Advances in Neural Information Processing Systems I*, vol. 1, p. 305, 1989.

[2] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. Clark, J. Dolan, D. Duggins, *et al.*, "Autonomous driving in urban environments: Boss and the urban challenge," *Journal of Field Robotics Special Issue on the 2007 DARPA Urban Challenge, Part I*, vol. 25, pp. 425–466, June 2008.

[3] A. Kelly and B. Nagy, "Reactive nonholonomic trajectory generation via parametric optimal control," *The International Journal of Robotics Research*, vol. 22, no. 7-8, p. 583, 2003.

[4] B. Nagy and A. Kelly, "Trajectory generation for car-like robots using cubic curvature polynomials," *Field and Service Robots*, vol. 11, 2001.

[5] M. McNaughton, C. Urmson, J. Dolan, and J. Lee, "Motion planning for autonomous driving with a conformal spatiotemporal lattice," in *Robotics and Automation (ICRA), IEEE International Conference on*, vol. 1, pp. 4889–4895, 2011.

[6] M. McNaughton, *Parallel Algorithms for Real-time Motion Planning*. PhD thesis, Robotics Institute, Carnegie Mellon University, July 2011.

[7] M. Werling, J. Ziegler, S. Kammel, and S. Thrun, "Optimal trajectory generation for dynamic street scenarios in a frenet frame," in *Robotics and Automation (ICRA), IEEE International Conference on*, pp. 987–993, 2010.

[8] M. Likhachev, G. Gordon, and S. Thrun, "ARA*: Anytime A* with provable bounds on sub-optimality," *Advances in Neural Information Processing Systems (NIPS)*, vol. 16, 2004.

[9] S. LaValle and J. Kuffner Jr, "Randomized kinodynamic planning," in *Robotics and Automation (ICRA), IEEE International Conference on*, vol. 1, pp. 473–479, 1999.

[10] S. Koenig and M. Likhachev, "Improved fast replanning for robot navigation in unknown terrain," in *Robotics and Automation (ICRA), IEEE International Conference on*, vol. 1, pp. 968–975, 2002.

[11] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Practical search techniques in path planning for autonomous driving," in *Proceedings of the First International Symposium on Search Techniques in Artificial Intelligence and Robotics (STAIR-08)*, (Chicago, USA), AAAI, June 2008.

[12] M. Zucker, J. Bagnell, C. Atkeson, and J. Kuffner, "An optimization approach to rough terrain locomotion," in *Robotics and Automation (ICRA), IEEE International Conference on*, pp. 3589–3595, 2010.

[13] B. Gough, *GNU Scientific Library Reference Manual*. Network Theory Ltd., 2009.

[14] J. van den Berg and M. Overmars, "Roadmap-based motion planning in dynamic environments," *Robotics, IEEE Transactions on*, vol. 21, no. 5, pp. 885–897, 2005.

[15] J. Ziegler and C. Stiller, "Spatiotemporal state lattices for fast trajectory planning in dynamic on-road driving scenarios," in *Intelligent Robots and Systems (IROS), IEEE/RSJ International Conference on*, pp. 1879–1884, 2009.

[16] J. Ziegler and C. Stiller, "Fast collision checking for intelligent vehicle motion planning," in *Intelligent Vehicles Symposium (IV), IEEE International Conference on*, pp. 518–522, 2010.

[17] J. Nelder and R. Mead, "A simplex method for function minimization," *The computer journal*, vol. 7, no. 4, p. 308, 1965.

[18] S. Petti and T. Fraichard, "Safe motion planning in dynamic environments," in *Intelligent Robots and Systems (IROS), IEEE/RSJ International Conference on*, pp. 2210–2215, 2005.