2018 Annual American Control Conference (ACC)
June 27–29, 2018. Wisconsin Center, Milwaukee, USA

FOAD: Fast Optimization-based Autonomous Driving Motion Planner

Jianyu Chen*, Changliu Liu and Masayoshi Tomizuka

Abstract— **Motion planning is one of the core modules for autonomous driving. Among the current motion planning techniques, optimization-based methods have unique advantages since they allow planning in continuous space and they can evaluate multiple objectives (such as hard constraints) in one formulation. However, it is hard to implement optimization-based methods in real-time in complicated environments due to 1) high computational complexity as the optimization problems are usually non-convex; and 2) difficulty to guarantee closed-loop performance because the low level trajectory tracking controller cannot perform perfect tracking. To solve the first challenge, convex feasible set algorithm (CFS) has been proposed for real time non-convex optimization. To solve the second challenge, a fast optimization-based autonomous driving motion planner (FOAD) is proposed in this paper which implements a soft constrained convex feasible set algorithm (SCCFS) as an enhanced version of CFS. The concept of closed-loop smoothness is defined and analyzed in this paper. Simulations and real vehicle experiments verify the efficiency and capability of the planner.**

I. Introduction

It is widely viewed that autonomous driving can significantly improve future mobility. Furthermore, it attains a number of benefits, such as freeing human drivers, easing road congestion and improving transportation safety. Motion planning is the core module in autonomous driving, which is to take the information from the environment (e.g. motions of the ego vehicle and the surrounding vehicles), and then plan a trajectory for the autonomous vehicle to execute.

Although motion planning has been investigated for decades, it is still difficult to plan a good trajectory in complex urban environment scenarios. The following characteristics need to be satisfied simultaneously: (1) computation needs to be in real-time in order for the automated vehicle to interact with the dynamic environment and deal with emergencies, (2) planning needs to be spatiotemporal for the automated vehicle to avoid dynamic obstacles, and (3) the planer needs to be able to deal with non-convex constraints such as collision avoidance.

Motion planning techniques for autonomous driving can be classified into three categories: graph-search-based, sampling-based and optimization-based methods [1], [2]. A representative search-based method is A* search [3] and the sampling-based method mainly refers to rapidly-exploring random tree (RRT) [4]. Both the graph-search-based and sampling-based methods plan in a discretized state space or action space. This will result in non-smooth trajectories and suboptimal solutions. Increase of the discretization resolution will largely lower the computation efficiency due to the “curse of dimensionality”.

The optimization-based method formulates motion planning as a mathematical optimization problem [5]. The planning is spatiotemporal which can deal with dynamic obstacles. It is performed in continuous space with infinite small resolution, which leads to better solutions than search and sampling. It is also easier to formulate the constraint in mathematical optimization, which helps us to formulate the complex environment in a uniform model. However, due to the complex non-convex constraints and the limitation of the existing numerical optimization techniques [6], the optimization-based motion planner are usually inefficient for real-time computation.

For optimization-based autonomous driving motion planning, Mercedes-Benz [7] utilized sequential quadratic programming (SQP) to solve the non-convex optimization problem, but the computation time is around 0.5s when implemented in C++, which is not fast enough for real time requirement. [8] uses optimization to solve a safe set to avoid collision, which is quite fast and reactive. But it only look for one time step ahead, which allows only limited intelligence. [9] proposed a constrained solution for the efficient iterative LQR algorithm, but the nature of indirect method [5] makes the problem easier to be trapped in local optima.

Convex feasible set (CFS) algorithm was proposed for real time trajectory optimization [10], [11]. It handles the non-convex constraints by convexification. Considering the unique geometric structure of motion planning problem, the algorithm breaks done the original problem into a sequence of easy-to-solve problem, which greatly improves the efficiency. When applied to a typical motion planning problem, it is an order of magnitude faster than SQP. The latter one is often seemed as the state-of-the-art technique for solving trajectory optimization problems.

However, the CFS algorithm is only a tool to solve optimization problems. Real-time implementation of autonomous driving motion planning needs a closed-loop MPC structure. In this paper, the fast optimization-based autonomous driving motion planner (FOAD) is established to provide a closed-loop MPC formulation for real-time implementation. Furthermore, the soft constrained convex feasible set (SCCFS) algorithm is proposed based on CFS to improve the smoothness of the motion.

The remainder of the paper is organized as follows. Section II reviews the CFS algorithm and introduces the SCCFS algorithm. Section III introduces and analyze the closed-

*This work is supported by DENSO International at America.

*J.Chen, C. Liu and M. Tomizuka are with the Department of Mechanical Engineering, University of California, Berkeley, CA 94720 USA (e-mail: jianyuchen@berkeley.edu, changliuliu@berkeley.edu, tomizuka@berkeley.edu).

loop smoothness. Section IV describes the formulation of the FOAD planner. Section V shows some simulation results. Section VI presents experimental results and section VII concludes the paper.

II. Soft Constrained Convex Feasible Set Algorithm

A. Convex Feasible Set (CFS) Algorithm

The convex feasible set algorithm was proposed by Liu, et.al to solve the following non convex optimization problem. In this subsection we briefly describe this algorithm.

Problem 1 (Optimization problem with non convex constraints).

$$\mathbf{x}^{*} = \underset{\mathbf{x} \in \Gamma \subset \mathbb{R}^n}{\arg\min}\, J(\mathbf{x}) \tag{1}$$

Here $\mathbf{x}$ is a variable of n dimension. J is the objective function, which is smooth and convex. Γ is the state space constraint, which is a subset of $\mathbb{R}^n$ and can be non convex. $\mathbf{x}^*$ is the optimal solution.

Like many other numerical optimization methods, the problem is solved iteratively. There are three steps for CFS algorithm:

1) Step 1 (Initialization): Give an initial value of the state variable $\mathbf{x}^{(0)}$. Note that $\mathbf{x}^{(0)}$ does not have to satisfy $\mathbf{x}^{(0)} \in \Gamma$.

2) Step 2 (Find the convex feasible set): Given the state variable of the last iteration $\mathbf{x}^{(k)}$, the convex feasible set is calculated corresponding to $\mathbf{x}^{(k)}$: $\mathcal{F}\left(\mathbf{x}^{(k)}\right) \subset \Gamma$. Γ is assumed to be the intersection of m constraints: $\Gamma = \cap_i \Gamma_i$. And Γ_i is defined as the space outside of the ith obstacle which can be defined as $\Gamma_i = \{\mathbf{x} : \phi_i \geq 0\}$ where ϕ_i is the signed distance function to the obstacle i.

In the original CFS algorithm, there are three cases for calculating the convex feasible set [10], [11]. Here in our autonomous driving application the main obstacles' 2D shape (bird view) is considered to be convex. Therefore, we only consider the case where the complement of Γ_i is convex. In this case the convex feasible set corresponding to obstacle i is calculated as

$$\mathcal{F}_i\left(\mathbf{x}^{(k)}\right) = \left\{\mathbf{x} : \phi_i\left(\mathbf{x}^{(k)}\right) + \nabla\phi_i\left(\mathbf{x}^{(k)}\right)\left(\mathbf{x} - \mathbf{x}^{(k)}\right) \geq 0\right\} \tag{2}$$

Note that $\mathcal{F}_i\left(\mathbf{x}^{(k)}\right)$ is actually a polytope and is convex. Thus, $\mathcal{F}\left(\mathbf{x}^{(k)}\right) = \cap_i \mathcal{F}_i\left(\mathbf{x}^{(k)}\right)$ is also a convex set. It is shown in [10] that $\mathcal{F}$ is non-empty if the obstacles are disjoint.

3) Step 3 (Solve the subproblem): The improved state variable of the next iteration $\mathbf{x}^{(\mathbf{k}+1)}$ is calculated as the optimal value in the convex feasible set $\mathcal{F}\left(\mathbf{x}^{(k)}\right)$:

$$\mathbf{x}^{(k+1)} = \underset{\mathbf{x} \in \mathcal{F}\left(\mathbf{x}^{(k)}\right)}{\arg\min}\, J(\mathbf{x}) \tag{3}$$

The algorithm works as follows: step 1 is applied to get the initial value $\mathbf{x}^{(0)}$, then step 2 and step 3 is applied iteratively to get a sequence $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(k)}, \ldots$. It is proved in [10] that this sequence will converge to a local optimum $\mathbf{x}^*$ of (1).

The iteration terminates at iteration n when some stop criteria is satisfied, e.g., $\left\|\mathbf{x}^{(n)} - \mathbf{x}^{(n-1)}\right\| < \varepsilon$ or $\left\|J\left(\mathbf{x}^{(n)}\right) - J\left(\mathbf{x}^{(n-1)}\right)\right\| < \sigma$.

B. Soft Constrained Convex Feasible Set (SCCFS) Algorithm

The convex feasible set algorithm can be extended to a modified version called soft constrained convex feasible set (SCCFS) algorithm. In this subsection, we directly describe the SCCFS algorithm. In the next section, we will define the concept of closed-loop smoothness and explain how SCCFS improves it.

Problem 2 (Soft constraint optimization problem with non convex constraints*).

$$\begin{aligned} &[\mathbf{x}^*,\ \mathbf{s}^*] = \underset{\mathbf{x} \in \Gamma(\mathbf{s})}{\arg\min}\, J(\mathbf{x}) + \omega\|\mathbf{s}\|^2 \\ &s.t. \quad \Gamma(\mathbf{s}) = \bigcap\nolimits_i \Gamma_i(\mathbf{s}) = \bigcap\nolimits_i \{\mathbf{x} :\ \phi_i(\mathbf{x}) \geq -\mathbf{s}\} \end{aligned} \tag{4}$$

where $J(\mathbf{x})$ is the objective function and $\omega\|\mathbf{s}\|^2$ is the term that penalizes the violation of the constraint.

The SCCFS algorithm uses the same three-step iteration structure. With the introduction of slack variables, however, step 2 and step 3 need to be modified. In step 2, the convex feasible set needs to be calculated corresponding to the augmented state variable $\mathbf{z} = \left[\mathbf{x}^T,\ \mathbf{s}^T\right]^T$ and the augmented distance function $\varphi_i(\mathbf{z}) = \phi_i(\mathbf{x}) + \mathbf{s}$:

$$\begin{aligned} \mathcal{F}_i\left(\mathbf{z}^{(k)}\right) &= \left\{\mathbf{z} : \varphi_i\left(\mathbf{z}^{(k)}\right) + \nabla\varphi_i\left(\mathbf{z}^{(k)}\right)\left(\mathbf{z} - \mathbf{z}^{(k)}\right) \geq 0\right\} \\ &= \Big\{\phi_i\left(\mathbf{x}^{(k)}\right) + \mathbf{s}^{(k)} \\ &+ \left[\begin{array}{cc} \nabla\phi_i\left(\mathbf{x}^{(k)}\right) & \mathbf{1}^T \end{array}\right]\left(\left[\begin{array}{c} \mathbf{x} \\ \mathbf{s} \end{array}\right] - \left[\begin{array}{c} \mathbf{x}^{(k)} \\ \mathbf{s}^{(k)} \end{array}\right]\right) \geq 0\Big\} \\ &= \{\mathbf{z} : L_i\mathbf{z} + S_i \leq 0\} \end{aligned} \tag{5}$$

Then we have:

$$\begin{aligned} \mathcal{F}\left(\mathbf{z}^{(k)}\right) &= \cap_i \mathcal{F}_i\left(\mathbf{z}^{(k)}\right) = \cap_i \{\mathbf{z} : L_i\mathbf{z} + S_i \leq 0\} \\ &= \{\mathbf{z} : L\mathbf{z} + S \leq 0\} \end{aligned} \tag{6}$$

where $L = \left[\begin{array}{ccc} L_1^T & L_2^T & \cdots \end{array}\right]^T$, $S = \left[\begin{array}{ccc} S_1^T & S_2^T & \cdots \end{array}\right]^T$.

Then in step 3 we need to solve the subproblem with the augmented state variable $\mathbf{z}$ and the augmented objective function $R(\mathbf{z})$:

$$\begin{aligned} \mathbf{z}^{(k+1)} &= \underset{L\mathbf{z}+S \leq 0,\ \mathbf{s} \geq 0}{\arg\min}\, R(\mathbf{z}) \\ &= \underset{L\mathbf{z}+S \leq 0,\ \mathbf{s} \geq 0}{\arg\min}\, J(\mathbf{x}) + w\|\mathbf{s}\|^2 \end{aligned} \tag{7}$$

Since the algorithm structure is the same as the original CFS algorithm, we can conclude that the sequence of augmented state variables $\mathbf{z}^{(1)}, \mathbf{z}^{(2)}, \ldots, \mathbf{z}^{(k)}, \ldots$ will converge to the local optimum of Problem 2. The stop criteria remains the same: $\left\|\mathbf{z}^{(n)} - \mathbf{z}^{(n-1)}\right\| < \varepsilon$ or $\left\|R\left(\mathbf{z}^{(n)}\right) - R\left(\mathbf{z}^{(n-1)}\right)\right\| < \sigma$.

*Constraints refer to the regions where the vehicle is not supposed to enter

III. Closed-loop Smoothness

A. Replanning Mechanism

In [10] and [11], only the static planning is considered, where all obstacles are static. For dynamic planning where the obstacles are moving, the trajectory should be replanned frequently to adapt to the change of the environment. Thus a closed-loop replanning mechanism needs to be established.

A naive method to do replanning is to simply solve the optimization problem based on the current state and then execute the solved trajectory. However, the computation time cannot be ignored. When the planned trajectory begins to be executed, the current state has already changed. For example, when the vehicle speed is 20m/s and the replanning time T_r[†] for trajectory planning is 0.2s, the position will change by 4 meters, which is quite a big error and cannot be ignored.

Our replanning scheme is shown in Fig.1. At time step k, we predict the position at time step $k+1$ based on the assumption of constant steering angle and constant speed. We denote the position as x_{k+1}. Then we take x_{k+1} as the current position and plan the trajectory $\mathbf{x}^*_{k+1}$.

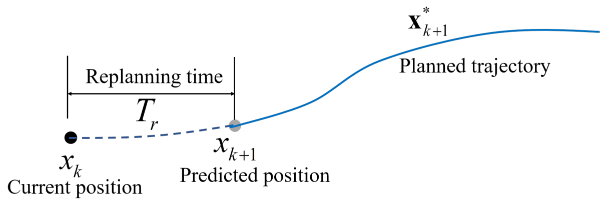

Fig. 1. The replanning scheme of the FOAD planner

B. Closed-loop Smoothness

For autonomous driving systems, there will usually be a planning-control architecture, meaning there is a lower level controller to track the planned trajectory. At time step k, the vehicle control command will be computed based on the planned trajectory $\mathbf{x}^*_k$. Denote the low level controller as a mapping from a trajectory $c_k = C(\mathbf{x}^*_k)$, where c_k is the control command at time step k and C denotes the controller. Note that the trajectory $\mathbf{x}^*_k$ needs to be transformed to the ego coordinate of the vehicle.

Intuitively, the difference between adjacent control command (e.g, $\|c_{k+1} - c_k\|$) indicates how smooth the motion is. Since the controller C is a continuous mapping, the motion smoothness can be further decided by the difference between the planned trajectories of adjacent time steps (e.g, $\left\|\mathbf{x}^*_{k+1} - \mathbf{x}^*_k\right\|$). Now we can define the closed-loop smoothness of trajectory planners.

Definition 1 (Closed-loop Smoothness) For two trajectory planners P_1 and P_2, we say P_1 is closed-loop smoother than P_2 if for the same initial states and constraints, we have:

$$\left\|\mathbf{x}^1_{k+1} - \mathbf{x}^1_k\right\| \le \left\|\mathbf{x}^2_{k+1} - \mathbf{x}^2_k\right\| \tag{8}$$

[†]The time difference between the two time steps, k to $k+1$, is called the replanning time and denoted as T_r in this paper. It is different from the sampling time, which is the time difference between two consecutive points of the planned trajectory and denoted as T_s.

for any time step k. Here $\mathbf{x}^1_k$ and $\mathbf{x}^2_k$ are the planned trajectory from P_1 and P_2 at time step k.

Although a rigorously proof to show that SCCFS is closed-loop smoother than CFS is not available, some analysis about it can be performed. First we show an intuitive example. As shown in Fig. 2, the yellow rectangle represents the obstacle, x_k and x_{k+1} are initial states of two adjacent time steps. The light and dark red/green curves are planned trajectories of two adjacent time steps from planner 1/2 (correspond to CFS/SCCFS). As we can see, the CFS algorithm (planner 1) uses hard constraint so the trajectory cannot violate the constraint. Since x_{k+1} is very close to the constraint boundary, the planned trajectory $\mathbf{x}^1_{k+1}$ changes a lot compared to $\mathbf{x}^1_k$. On the contrary, since the SCCFS allows violating the constraint a little bit, the planned trajectories $\mathbf{x}^2_{k+1}$ and $\mathbf{x}^2_k$ can stay close to minimize the cost function.

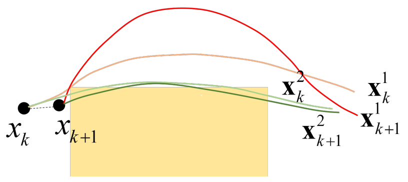

Fig. 2. The trajectory planned by CFS changes a lot near the constraint boundary

More theoretically, SCCFS can be regarded as CFS augmented with additional state dimension and constraints. As shown in Fig. 3, suppose the original state dimension is 1, denoted by x. s is the slack variable. For CFS, the planning is done in the x axis, with the one dimensional constraint $\Gamma^1_k(x)$ and cost function. The solutions of CFS for the kth and $k+1$th time steps are $\mathbf{x}^1_k$ and $\mathbf{x}^1_{k+1}$ respectively. For SCCFS, the planning is done in the x-s plane, where $\Gamma^2_k(x,s)$ represents the constraint and the ellipse represents the cost function. The solutions of SCCFS for the kth and $k+1$th time steps are $\left(\mathbf{x}^2_k, s_k\right)$ and $\left(\mathbf{x}^2_{k+1}, s_{k+1}\right)$ respectively. Their projections on x axis are $\mathbf{x}^2_k$ and $\mathbf{x}^2_{k+1}$. As we can see, due to the dimension augmentation, the trajectory change of SCCFS ($\left\|\mathbf{x}^2_{k+1} - \mathbf{x}^2_k\right\|$) becomes smaller than that of CFS ($\left\|\mathbf{x}^1_{k+1} - \mathbf{x}^1_k\right\|$), which indicates the closed-loop smoothness.

IV. Fast Optimization-based Autonomous Driving Motion Planner

A. Problem Formulation

For a robot motion planning problem, we need to first define the configuration space. In our autonomous driving motion planning problem, the configuration is the 2D position of the rear axle of the ego vehicle, denoted as $x \in \mathbb{R}^2$. Then the trajectory can be represented as $\mathbf{x} = \left[x_0^T, x_1^T, \cdots, x_H^T\right]^T \in \mathbb{R}^{2(H+1)}$, where H is the preview horizon; the subscript denotes the time step, where 0 means the current time step.

The space occupied by the obstacles can be represented by $\mathcal{O}^i_t$ which means the ith obstacle in time step t, where the superscript means the ID of the obstacle and the subscript means the time step. The traffic events (such as red traffic

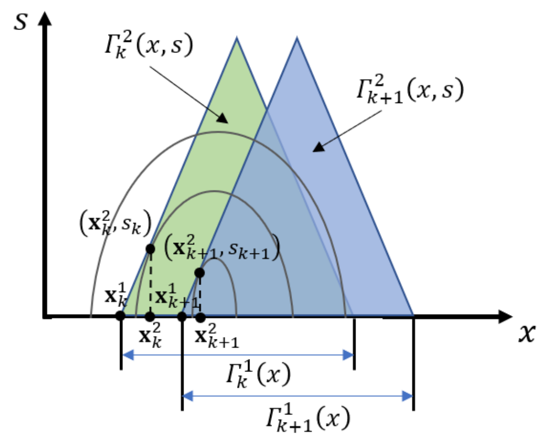

Fig. 3. Visualization of the planning process comparison for CFS and SCCFS

light) can be represented by $\mathcal{T}_t^j$. Both the obstacles and the traffic events are set as soft constraints in the optimization problem.

Problem 3 (Optimization-based Autonomous Driving Motion Planning Problem).

$$\begin{gathered}[\mathbf{x}^*,\ \mathbf{s}_\Gamma^*,\ \mathbf{s}_\mathrm{T}^*] = \underset{\mathbf{x}\in\Gamma(\mathbf{s}_\Gamma),\, \mathbf{x}\in\mathrm{T}(\mathbf{s}_\mathrm{T}),\, \mathbf{s}_\Gamma\geq 0,\, \mathbf{s}_\mathrm{T}\geq 0}{\arg\min} J(\mathbf{x}) + \omega\left(\|\mathbf{s}_\Gamma\|^2 + \|\mathbf{s}_\mathrm{T}\|^2\right) \\ s.t.\ \ \Gamma(\mathbf{s}_\Gamma) = \textstyle\bigcap_i \Gamma_i(\mathbf{s}_\Gamma) = \bigcap_i \{\mathbf{x} :\ \phi_i(\mathbf{x}) \geq -\mathbf{s}_\Gamma\} \\ \mathrm{T}(\mathbf{s}_\mathrm{T}) = \textstyle\bigcap_j \mathrm{T}_j(\mathbf{s}_\mathrm{T}) = \bigcap_j \{\mathbf{x} :\ \tau_j(\mathbf{x}) \geq -\mathbf{s}_\mathrm{T}\}\end{gathered} \tag{9}$$

where Γ is the constraint for obstacles and T is the constraint for traffic events. ϕ and τ are some distance functions relative to the obstacles and traffic events as will be discussed in Section IV-C.

Problem 3 is a special case of problem 2 and can be solved using SCCFS algorithm.

B. Objective Function

The objective function is modeled to be quadratic. Thus $J(\mathbf{x}) = \mathbf{x}^T Q\mathbf{x} + \mathbf{x}^T q + r$.

1) Offset from reference: One important object is to maintain the vehicle close to the centerline of the lane. We define a reference trajectory $\mathbf{x}_{ref} \in \mathbb{R}^{2(H+1)}$, which is of the same length as the planning trajectory and can be set as the points on the centerline of lane. Then the cost relative to the tracking accuracy is:

$$\begin{aligned} c_{off} &= \tfrac{1}{2}\omega_{off}(\mathbf{x}-\mathbf{x}_{ref})^T(\mathbf{x}-\mathbf{x}_{ref}) \\ &= \tfrac{1}{2}\omega_{off}\mathbf{x}^T\mathbf{x} - \omega_{off}\mathbf{x}^T\mathbf{x}_{ref} + \tfrac{1}{2}\omega_{off}\mathbf{x}_{ref}^T\mathbf{x}_{ref}\end{aligned} \tag{10}$$

where we penalize the distance from the planned trajectory to the reference trajectory. ω_{off} is a weighting coefficient.

2) Acceleration: Passenger comfort is another important consideration. The main term related to passenger comfort is the vehicle acceleration. When the acceleration is large, either lateral or longitudinal, passengers will feel uncomfortable. Thus, we penalize the magnitude of the acceleration by

$$c_{acc} = \frac{1}{2}\omega_{acc}(A\mathbf{x})^T(A\mathbf{x}) = \frac{1}{2}\omega_{acc}\mathbf{x}^T A^T A\mathbf{x} \tag{11}$$

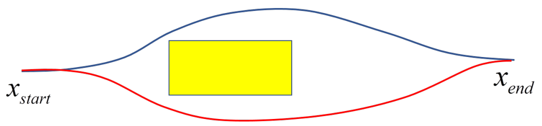

Fig. 4. Multiple local optima introduced by the rectangle obstacle

where ω_{acc} is a weighting coefficient and A is the acceleration matrix:

$$A = \frac{1}{T_s^2}\begin{bmatrix} I & -2I & I & 0 & \cdots & 0 \\ 0 & I & -2I & I & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & I & -2I & I \end{bmatrix} \tag{12}$$

where T_s is the sampling time.

3) Trajectory Change: During replanning, the trajectory between the adjacent time steps should not change too much to maintain stability and smoothness. Here we record the current vehicle position as $\mathbf{x}_{rec}$. Then at the next time step, we penalize the acceleration calculated with the three point pair $(\mathbf{x}_{rec},\ \mathbf{x}_0,\ \mathbf{x}_1)$:

$$c_{change} = \frac{1}{T_s^4}\omega_{change}(\mathbf{x}_{rec} - 2\mathbf{x}_0 + \mathbf{x}_1)^T(\mathbf{x}_{rec} - 2\mathbf{x}_0 + \mathbf{x}_1) \tag{13}$$

where $\mathbf{x}_0$ is the initial position and $\mathbf{x}_1$ is the first point in the planned trajectory.

C. Constraints

There are two kinds of constraints. One is caused by the obstacles; the body of the ego vehicle should not overlap with the body of the surrounding vehicle. The other is caused by the traffic rules; e.g., the ego vehicle should not pass the crossroad until the traffic light becomes green. The formulation of these two kinds of constraints are discussed below.

1) Obstacles: Here we consider the main obstacles: surrounding vehicles. Basically, a vehicle can be seen as a rectangle. However, a constraint of a rectangle shape might introduce multiple local optima. As shown in Fig.4, both the blue and the red trajectory can be an optimal solution. In this case, an addition module is needed to give a proper initial value, which complicates the system architecture. Also, since the SCCFS algorithm uses soft constraint, a margin is needed to guarantee safety.

To address the above problems, we build a polygon to represent the constraint. As shown in Fig.5, the new constraint is a trapezoid like polygon which contains the vehicle shape. Four more parameters are needed to define this polygon, the lateral margin l_{lat}, the front longitudinal margin l_{lonf}, the back longitudinal margin l_{lonb} and the angle θ_{ang}. The above parameters determine the shape of the margin region and they are tunable. This polygon constraint will not introduce local optima and there is only one optimal trajectory.

To consider the shape of the ego vehicle, we introduce vehicle discs. As shown in Fig.6, two discs are used to represent the vehicle body, one of which has its center at the

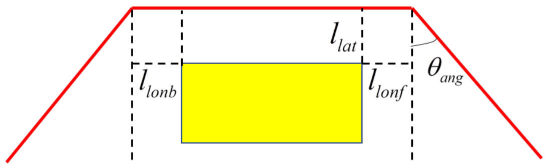

Fig. 5. The polygon representation of the obstacle

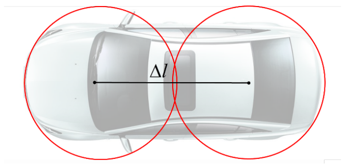

Fig. 6. The vehicle discs containing the ego vehicle body

rear axle, and the other has its center at front. As specified previously, x_k is the position of the rear axle of the ego vehicle at time step k. Here we define the vehicle direction at time step k to be aligned to $\overline{x_k x_{k+1}}$. Thus the center of the front disk can be calculated as:

$$O_{front} = x_k + \frac{x_{k+1} - x_k}{\|x_{k+1} - x_k\|} \cdot \Delta l \tag{14}$$

Thus we can define a transformation $\mathbf{y} = F\mathbf{x} \in R^{2(H+1)}$, of which y_{2k} is the center of the rear disc, and y_{2k+1} is the center of the front disc. The distance function for the ith obstacle $\phi_i(\mathbf{x}) = d_i(\mathbf{y}) : R^{2(H+1)} \to R^{2(H+1)}$ is a vector function whose image is the vector of the signed distances at each time step. For example, the $2k$th value of $\phi_i(\mathbf{x})$ means the distance from the rear disc center of the ego vehicle to the ith obstacle at time step k, and the $(2k+1)$th value means the distance from the front disc center of the ego vehicle to the ith obstacle. The distance value is positive when it is outside the obstacle and is negative when inside the obstacle. A margin r is added to the linearized constraint in (6):

$$L\mathbf{z} + S \leq -r \tag{15}$$

where r is the radius of the vehicle disc. This is to keep the vehicle body outside the obstacle, not just at the center of the vehicle disc.

2) *Traffics:* The main traffic constraints can be classified as two categories: the static event and the dynamic event. The idea is similar to one of our previous works [12].

A static event refers to the case that the position of the constraint will not change. This includes the traffic light scenario and the pedestrian crossroad scenario. Take the pedestrian crossroad as an example, as shown in Fig.7, the pedestrian is predicted to enter the crossroad at time t_0 and leave at time t_1. The area between x_0 and x_1 is the crossroad. According to the traffic rule, the vehicle should not enter the crossroad while the pedestrian is on it.

There are two discrete decisions in this scenario: the ego vehicle can decelerate to wait before the line of x_0 within t_1 (wait the pedestrian to pass), or to accelerate and exceed the line of x_1 after t_0 (pass before the pedestrian comes). The motion planner that we discuss here cannot decide which

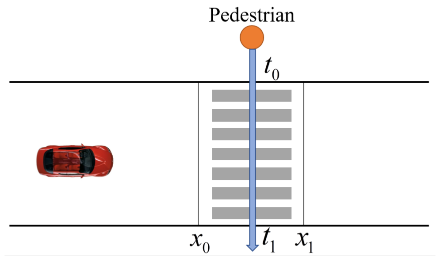

Fig. 7. The crossroad scenario with pedestrian

maneuver to execute. To make this decision, a high-level maneuver planner is needed, which is beyond the scope of this paper. Here we assume we already get the command that specifies which maneuver we should execute. Other traffic events also contain discrete maneuvers and the same assumption exists.

Suppose we decide to wait for the pedestrian to pass, then the constraint area is the area at the right of x_0, and the constraint is active before t_1. Suppose $K = ceil\left(\frac{t_1}{T_s}\right)$, then the related distance function is $\tau(\mathbf{x}) = d(\mathbf{y}) : R^{2(H+1)} \to R^{2(K+1)}$, where the $2k$th value of $\tau(\mathbf{x})$ is the signed distance from the rear center of the ego vehicle to the line of x_0, and the $2k+1$th value is the signed distance from the front center to the line.

A dynamic event means that the position of the constraint will change with time. This is related to the road participants. It can model car following, merging, intersection and roundabout scenarios. For car following, we can assume a moving line behind the leading car l with the car following distance, where l_k is the line at time step k. Then the related distance function is $\tau(\mathbf{x}) = d(\mathbf{y}) : R^{2(H+1)} \to R^{2(H+1)}$, where the $2k$th value of $\tau(\mathbf{x})$ is the signed distance from the rear center of the ego vehicle to the line of l_k.

For merging, intersection and roundabout, they can all be classified as merging scenario and have two discrete maneuvers. Take the lane merging as an example, as shown in Fig.8, the ego vehicle (red) needs to merge to the same lane with a surrounding vehicle (yellow). Similarly, there are two maneuvers, to wait the surrounding vehicle pass and then merge, or to merge before the vehicle pass. The surrounding vehicle is predicted to enter the conflict zone at time step K. Suppose the automated vehicle decides to wait for the surrounding vehicle to pass and then merge, then there will be a car following constraint after time step K. So the distance function is $\tau(\mathbf{x}) = d(\mathbf{y}) : R^{2(H+1)} \to R^{2(H+1-K)}$ where the $2k$th value of $\tau(\mathbf{x})$ is the signed distance from the rear center of the ego vehicle to the back line of the surrounding vehicle at time step $k+K$.

V. Simulations

The planner is tested on simulations for two common urban scenarios: an on-road driving scenario and a parking lot driving scenario. The planner is implemented in Matlab script and run on a laptop with 2.6GHz Intel Core i7-6600U.

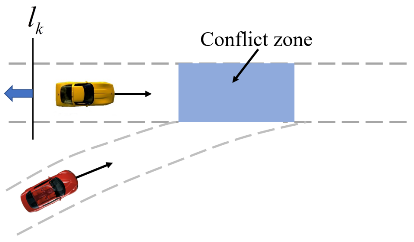

Fig. 8. The merging scenario

The performance is then analyzed by comparisons between SCCFS, CFS and SQP.

A. *The Matlab Simulator*

In order to simulate the application of the planner, a complete architecture of autonomous driving (e.g, the perception-planning-control architecture) as well as the vehicle dynamics simulation is needed. The vehicle dynamics simulation is used to simulate the real-world vehicle reaction and the perception-planning-control architecture is used to drive the simulated vehicle (The planner alone cannot drive the vehicle, only the low-level controller can drive the vehicle).

1) Vehicle Model: The model applied here is the vehicle bicycle kinematic model, as shown in Fig.9.

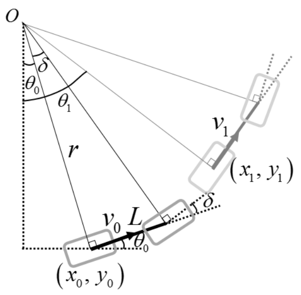

Fig. 9. The vehicle bicycle kinematic model

The current vehicle state includes the 2D position $(x_0,\ y_0)$, the velocity v_0 and the heading θ_0. The control input is the acceleration a and the steering angle δ. L is the wheel base. According to the kinematic model, and under the assumption that the steering angle maintains the same value in a replanning time, then the vehicle will rotate around the instant center O and r is the rotation radius. The distance the vehicle moves in one replanning time T_r is $l = v_0 T_r + \frac{1}{2} a T_r^2$ and the curvature $\kappa = \frac{\tan\delta}{L}$. Then the updated vehicle state after one sampling time is:

$$\begin{aligned}
&v_1 = v_0 + aT_r \\
&\theta_1 = \theta_0 + \int_0^l \kappa ds = \theta_0 + \kappa l \\
&x_1 = x_0 + \int_0^l \cos\left(\theta_0 + \kappa s\right) ds = x_0 + \tfrac{\sin(\theta_0+\kappa l) - \sin(\theta_0)}{\kappa} \\
&y_1 = y_0 + \int_0^l \sin\left(\theta_0 + \kappa s\right) ds = x_0 + \tfrac{\cos(\theta_0) - \cos(\theta_0+\kappa l)}{\kappa}
\end{aligned} \tag{16}$$

2) Simulator Scheme: In our simulator, the perception module can be removed because all the observations (e.g, position, velocity of ego and surrounding vehicles) can be

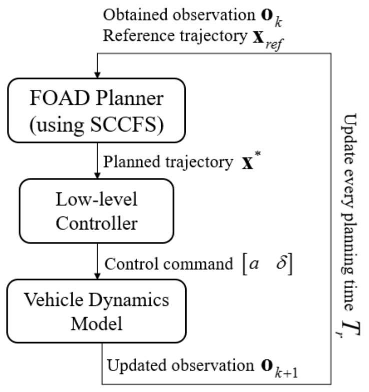
Obtained observation $\mathbf{o}_k$
Reference trajectory $\mathbf{x}_{ref}$
FOAD Planner (using SCCFS)
Planned trajectory $\mathbf{x}^*$
Low-level Controller
Control command $[a \quad \delta]$
Vehicle Dynamics Model
Updated observation $\mathbf{o}_{k+1}$
Update every planning time T_r

Fig. 10. The simulator scheme

directly acquired from the simulator. The architecture is then simplified to a planning-control scheme. The simulator scheme is shown in Fig.10.

First, the current observation (such as position and velocity of ego and surrounding vehicles) $\mathbf{o}_k$ as well as the reference trajectory $\mathbf{x}_{ref}$ are obtained and passed as the input to the FOAD planner. The planner then plans an optimal trajectory $\mathbf{x}^*$ and passes it to the low-level controller. The controller then compute the control command $\begin{bmatrix} a & \delta \end{bmatrix}$ to follow the planned trajectory. The vehicle dynamics model then takes the control command and calculate the observation for the next timestep $\mathbf{o}_{k+1}$. The iteration continues every replanning time T_r.

3) Low-level Controller: For the low-level control, we use an iterative LQR (ILQR) controller. The objective function is the difference to the planned trajectory and the acceleration. Then the nonlinear vehicle model is linearized and an LQR subproblem is solved at every iteration. The details of the ILQR algorithm are described in [13], [14]. When converged, the control input at the first time step is selected as the control command. To avoid local optima and shorten the computation time, the horizon for the ILQR controller is selected to be shorter than the horizon of the FOAD planner, e.g, half of it.

B. *Scenarios*

Two common urban scenarios are implemented in the Matlab simulator. In both scenarios, both the sampling time and the replanning time are set to be 0.2 second, which is conservative enough to guarantee the worst case computation.

1) On-road Driving: The on-road driving scenario is very common in urban driving, as most of the times the vehicle needs to drive on the road, where the vehicle is supposed to follow the centerline of the lane.

In this case, there are two lanes on the road. The ego vehicle is required to maintain in the centerline of the right lane with the reference speed v^r while keep safe with the surrounding vehicles. There are two front vehicles on the right lane and one vehicle on the left lane. The reference trajectory $\mathbf{x}_{ref}$ is set as a trajectory on the centerline of the

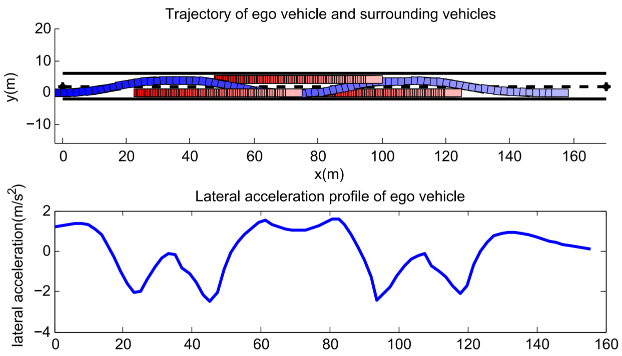

Fig. 11. Simulation for on-road driving scenario with FOAD planner using SCCFS algorithm

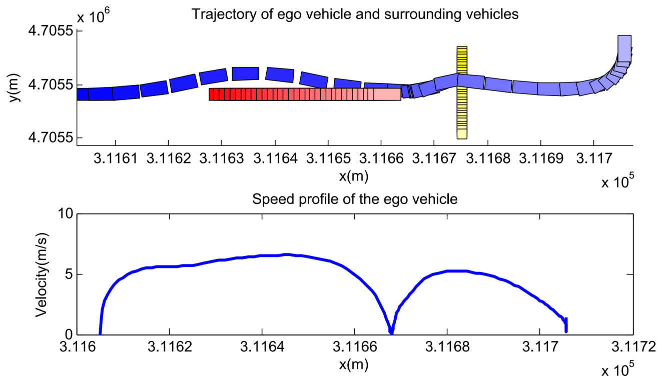

Fig. 12. Simulation for parking lot driving scenario with FOAD planner using SCCFS algorithm

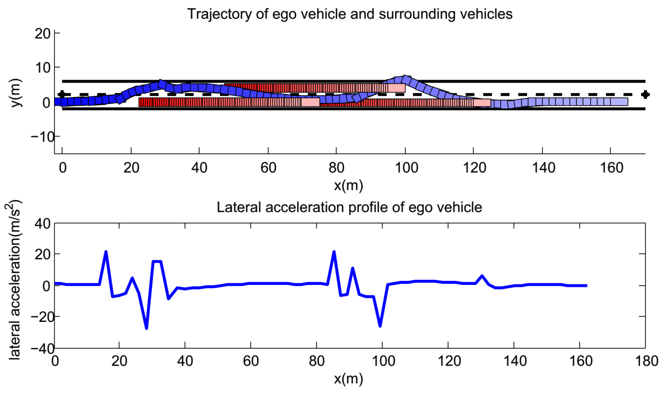

Fig. 13. Simulation for on-road driving scenario with FOAD planner using CFS algorithm

right lane, starting from the current ego vehicle position and maintain the distance gap of $v^r T_s$.

The results are shown in Fig.11. The blue rectangles represent the trajectory of the ego vehicle, and the red rectangles represent the trajectories of the three surrounding vehicles, where the shallower color represents the future timesteps. From the figure we can see that the ego vehicle successfully overtakes the three surrounding vehicles smoothly and then maintain in the centerline of the lane. The lateral acceleration profile of the ego vehicle is also plotted in the figure, where the lateral acceleration is well-bounded in the whole scenario. The maximum lateral acceleration is only about $2m/s^2$. The driving is quite smooth and comfortable.

2) Parking Lot Driving: Parking lot driving is another common scenario in urban driving. In this case, we first drove a real car in a parking lot, started from the entrance and parked into a target position, and then collect the GPS data of the trajectory. The raw data is then transformed to Universal Transverse Mercator (UTM) coordinate as ρ_{UTM} for generating the reference trajectory for the FOAD planner, and thus our simulation is also in the UTM coordinate. The reference trajectory is then generated by finding the closest point x_0 on ρ_{UTM} to the current ego vehicle position, and then pick N points from ρ_{UTM} start at x_0 which maintain the distance gap of around $v^r T_s$.

The results are shown in Fig.12. The blue rectangle represents the ego vehicle, the red rectangle is the moving front vehicle and the yellow rectangle is the pedestrian. From the figure, we can see the ego vehicle first accelerates to overtake the front vehicle, then waits for the pedestrian to pass, and finally accelerates again and parked to the target spot. One might note that the vehicle steers a little to the left during waiting for the pedestrian, this is because the vehicle is always required to go faster to maintain high efficiency, steering to the left allows it to pass the pedestrian earlier.

C. Performance Analysis

The performance analysis of the SCCFS-based FOAD planner comes with two folds: first, the comparison of computation time shows SCCFS and CFS is much more efficient than SQP, which is the state-of-the-art trajectory optimization algorithm. Second, both SCCFS and CFS are applied to FOAD planner and tested in simulation and FOAD planner with SCCFS is shown to be much better than with CFS.

1) Computation time of SCCFS, CFS and SQP: In [10], [11], the CFS algorithm is proved to be much faster than SQP while SCCFS algorithm has comparable efficiency with CFS algorithm. In our simulation, the average computation time of SCCFS and CFS for the overtaking scenario is 0.14s and 0.17s respectively. SCCFS is even more efficient than CFS in this case.

2) SCCFS and CFS applied to FOAD planner: We apply CFS algorithm to FOAD planner and test it in the same on-road driving scenario as in section V-B. The results are shown in Fig.13. From the figure, we can see the trajectory of the ego vehicle is not smooth especially when it is close to the surrounding vehicles. The lateral acceleration is horrible, where the maximum value exceeds $20m/s^2$. In simulation we only consider the 2D movement and ignore tire sliding. However in the real world, this level of lateral acceleration can cause serious roll over and drift.

VI. Vehicle Experiments

We implement the FOAD planner in a real Lincoln MKZ car equipped with GPS, camera, LIDAR and radar. The details of the hardware setup can be found in [8]. The perception module gives the information of the ego vehicle and surrounding vehicles, and the low level controller can follow the trajectory given by the FOAD planner. In this paper we only discuss details of the trajectory planning.

We did experimental tests on FT Techno of America (FTTA) at Michigan. The satellite map of the track is shown in Fig.14. In the figure, the blue box and the red box represent the initial position of the ego vehicle and a surrounding vehicle. The task for the ego vehicle is to overtake the front vehicle safely while stay close to the centerline of the lane. The test data is visualized in Fig.15. The ego

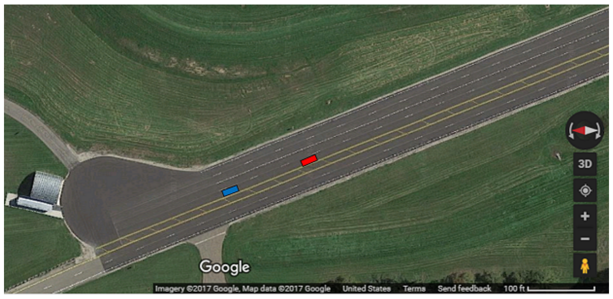
Google
Imagery ©2017 Google, Map data ©2017 Google United States Terms Send feedback 100 ft
3D

Fig. 14. The satellite map of FTTA track

vehicle accelerate from static and overtake the front vehicle smoothly. The lateral acceleration for the whole process is bounded in $1m/s^2$ and the ride is quite comfortable. We also tried to test Foad planner using the original CFS algorithm but the ego vehicle's reaction appears to be too dangerous.

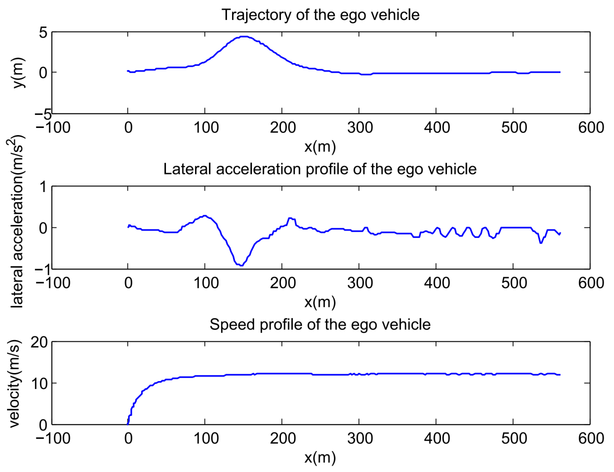

Fig. 15. Real vehicle experiments with FOAD planner using SCCFS algorithm

VII. Conclusion

In this paper, the soft constrained convex feasible set (SCCFS) algorithm was proposed based on the convex feasible set (CFS) algorithm to improve closed-loop smoothness in real time implementation. The fast optimization-based autonomous driving (FOAD) motion planner was then established based on the SCCFS algorithm, which formulated the objective functions and constraints for optimization-based motion planning problem. A Matlab simulator was established and simulations were done for on-road driving scenario and parking lot scenario. Analysis shows the FOAD planner with SCCFS algorithm performs the best comparing to CFS and SQP algorithm. Real vehicle experiment for an overtaking scenario was conducted on the FTTA track.

In the future, more work needs to be done in the decision making level, such as to yield or to pass in a roundabout.

References

[1] Gonzlez, D., Prez, J., Milans, V. and Nashashibi, F. *A review of motion planning techniques for automated vehicles*. IEEE Transactions on Intelligent Transportation Systems 17.4 (2016): 1135-1145.
[2] Paden, B., Cap, M., Yong, S.Z., Yershov, D. and Frazzoli, E. *A Survey of Motion Planning and Control Techniques for Self-driving Urban Vehicles*. IEEE Transactions on Intelligent Vehicles 1.1 (2016): 33-55.
[3] Dolgov, D., Thrun, S., Montemerlo, M. and Diebel, J. *Practical search techniques in path planning for autonomous driving*. Ann Arbor 1001 (2008): 48105.
[4] Kuffner, J.J. and LaValle, S.M. *RRT-connect: An efficient approach to single-query path planning*. Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on. Vol. 2. IEEE, 2000.
[5] J.T.Betts. *Practical methods for optimal control and estimation using nonlinear programming*. Second edition. [book review of MR2589623].[J]. Siam Review, 2011(1):183-185.
[6] Nocedal, J. and Wright S.J. *Numerical Optimization*. Second Edition[J]. 1999.
[7] Ziegler, J., Bender, P., Dang, T. and Stiller, C. *Trajectory planning for BerthaA local, continuous method*. Intelligent Vehicles Symposium Proceedings, 2014 IEEE. IEEE, 2014.
[8] Liu, C., Chen, J., Nguyen, T.D. and Tomizuka, M. *The Robustly-Safe Automated Driving System for Enhanced Active Safety*. No. 2017-01-1406. SAE Technical Paper, 2017.
[9] Chen, J., Wei, Z., and Tomizuka, M. *Constrained ilterative LQR for on-road autonomous driving motion planning*. To appear on IEEE 20th International Conference on Intelligent Transportation. 2017.
[10] Liu, C., Lin, C.Y. and Tomizuka, M. *The convex feasible set algorithm for real time optimization in motion planning*. arXiv preprint arXiv:1709.00627 (2017).
[11] Liu, C., Lin, C.Y., Wang, Y. and Tomizuka, M. *Convex feasible set algorithm for constrained trajectory smoothing*. American Control Conference, 2017 IEEE.
[12] Zhan, W., Chen, J., Chan, C.Y., Liu, C. and Tomizuka, M. *Spatially-partitioned environmental representation and planning architecture for on-road autonomous driving*. Intelligent Vehicles Symposium (IV), 2017 IEEE. IEEE, 2017.
[13] Li, W. and Todorov, E. *Iterative linear quadratic regulator design for nonlinear biological movement systems*. ICINCO (1). 2004.
[14] Todorov, E. and Li, W. *A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems*. American Control Conference, 2005. Proceedings of the 2005. IEEE, 2005.