Proceedings of ASME 2020 Dynamic Systems and Control Conference
DSCC 2020
October 4-7, 2020, Pittsburgh, Pennsylvania, USA

DSCC2020-3201

MULTI-CAR CONVEX FEASIBLE SET ALGORITHM IN TRAJECTORY PLANNING

Jing Huang
Intelligent Control Laboratory
The Robotics Institute
Carnegie Mellon University
Email: jingh2@andrew.cmu.edu

Changliu Liu
Intelligent Control Laboratory
The Robotics Institute
Carnegie Mellon University
Email: cliu6@andrew.cmu.edu

ABSTRACT

Trajectory planning is an essential module for autonomous driving. To deal with multi-vehicle interactions, existing methods follow the prediction-then-plan approaches which first predict the trajectories of others then plan the trajectory for the ego vehicle given the predictions. However, since the true trajectories of others may deviate from the predictions, frequent re-planning for the ego vehicle is needed, which may cause many issues such as instability or deadlock. These issues can be overcome if all vehicles can form a consensus by solving the same multi-vehicle trajectory planning problem. Then the major challenge is how to efficiently solve the multi-vehicle trajectory planning problem in real time under the curse of dimensionality. We introduce a novel planner for multi-vehicle trajectory planning based on the convex feasible set (CFS) algorithm. The planning problem is formulated as a non-convex optimization. A novel convexification method to obtain the maximal convex feasible set is proposed, which transforms the problem into a quadratic programming. Simulations in multiple typical on-road driving situations are conducted to demonstrate the effectiveness of the proposed planning algorithm in terms of completeness and optimality.

1 INTRODUCTION

Autonomous driving has been viewed as a promising technology to transform today's transportation systems. One of the key challenges for trajectory planning for autonomous vehicles is to handle the interactions among vehicles.

1.1 Multi-Vehicle Interactions

To handle the interactions among vehicles, existing solutions seek to actively predict other vehicles and incorporate the prediction in the planning for the ego vehicle [1, 2]. However, without perfect information about other vehicles, the prediction may deviates from the actual behaviors of the others. As a result, the motion plans of the ego vehicle need to be regenerated from time to time in order to make more informed decisions when new information of others is received in real time [3]. However, this replanning scheme may cause issues in terms of closed-loop stability. For example, in a four-way intersection, if vehicles from the four directions arrive at the stop sign at the same time, no vehicle has the priority to cross the intersection. Suppose the behaviors of the vehicles are designed in such a way that 1) they will inch forward to test the response of others; 2) they predict that once a surrounding vehicle yields, it should keep yielding; 3) they will then cross the intersection if others yield and stop if other do not. However, when all vehicles are doing so, this prediction is wrong and the system may be trapped into a inch-stop-inch-stop loop, which is either unstable or converging to a deadlock. The underlying problem can be attributed to information asymmetry and the lack of consensus among vehicles, which leads to prediction errors and sub-optimal motion plans. The resulting behavior of the multi-vehicle system is undesired.

To make the autonomous transportation system more efficient and safe, we may eliminate the information asymmetry by allowing active communication among vehicles [4]. However, limited by the bandwidth of the communication, only high-level information can be shared among vehicles. The motion planning

problem is still non-trivial under multi-vehicle interactions. This paper considers the case that all vehicles know the cost functions of one another (high-level information known *a priori* or through communication), and every vehicle solves in real time a multi-vehicle trajectory planning problem by minimizing all these costs. When all vehicles have the same understanding of the cost functions (no information asymmetry), they should generate the same trajectory plans for all vehicles (called as a consensus). An individual vehicle only needs to execute its own trajectory under the consensus. When all vehicles act according to the consensus, there should not be any prediction error, hence no need for replanning. This approach solves the prediction and planning problem at the same time through multi-vehicle trajectory planning, hence properly handles the multi-vehicle interactions. If the communication of the cost functions can be perfectly handled, the only challenge is to develop efficient algorithms to solve the multi-vehicle trajectory planning problem in real time.

1.2 Multi-Vehicle Trajectory Planning

The multi-vehicle trajectory planning problem has received many research interests since the 1980s [5–8]. There are centralized approaches as well as decentralized approaches to solve the problem. Note the distinction between centralized and decentralized approaches in this paper is considered only regarding the solution approach, not how the system is executed in the real world. For example, the scenario we discussed earlier that all vehicles independently solve the multi-vehicle trajectory planning problem is a decentralized system in the real world. If an individual vehicle only uses one computation unit to solve the problem, then we call it a centralized solution approach. On the other hand, it is possible that there exists a central traffic manager for the road network which handles multi-vehicle trajectory planning, in which way the system is centralized in the real world. The central traffic manager may solve the problem in a decentralized approach using multiple computation units. In the following discussion, the vehicles are called agents when referring to solution approaches. Decentralized approaches call multiple computation units to handle different sub problems (e.g., plans for individual agents) and design unique coordination mechanisms to achieve the global optima. Representative methods include alternating direction method of multipliers (ADMM) [9] and distributed conflict resolution [10]. Centralized approaches only use one computation unit to solve the problem, which treat multiple agents as a single composite agent and directly plan for the composite agent. The advantage of centralized approaches is that there is no coordination overhead, while the disadvantage is the curse of dimensionality when we compose multiple agents together [11]. The objective of this paper is to develop a centralized approach to efficiently solve the multi-vehicle trajectory planning problem.

A centralized multi-vehicle trajectory planning problem has an equivalent mathematical form as a trajectory planning problem for a single vehicle. Typical trajectory planning approaches can be divided into three categories: graph search based algorithms, sampling based algorithms, and optimization based algorithms. A representative graph search based algorithm is the A* algorithm [12], which takes a heuristic function and outputs an optimal trajectory that has the lowest cost to traverse the graph. Rapidly exploring random tree (RRT) [13] is one typical sampling based algorithm. It samples nodes in the configuration space and plans a feasible trajectory once the destination is connected to the start. However, both A* and RRT highly depend on the resolution of the graph or the tree and can easily result in non-smooth trajectories [14], which are sub-optimal and not ideal for autonomous driving. In many cases, these trajectories need to go through an additional smoothing step before execution.

Optimization based trajectory planner treats a trajectory as the decision variable to be optimized and can generate trajectories that are much smoother than the other techniques. However, the drawbacks are that optimization based algorithms are less computationally efficient [15] as the optimization problem is usually nonlinear and non-convex. In addition, it is non-trivial to mathematically specify complex interaction constraints (e.g., collision avoidance) for multi-vehicle trajectory planning, while the constraints are coupled with the non-holonomic vehicle dynamics. For example, some approaches [16] define uniform distance margins between different vehicles along all directions. This homogeneous constraint is inefficient. As the future positions of a vehicle are constrained by its non-holonomic dynamics, the safety margin should be larger in the longitudinal direction than the lateral direction in the vehicle body frame.

In summary, the main challenge in optimization-based planning is to properly specify the optimization problem as well as to efficiently solve the specified optimization problem. Problem specification and problem solving are coupled together. Indeed, we need to properly define the constraints to handle multi-vehicle interactions and design the cost function to allow a faster computation.

1.3 Contributions

This paper leverages the Convex feasible set (CFS) algorithm [17], an optimization based motion planning algorithm, and extends it to solve multi-vehicle trajectory planning problems in typical driving scenarios. The key steps to apply CFS on trajectory planning is to 1) formulate a reasonable quadratic cost function that penalizes the magnitude of acceleration and speed of the target trajectory as well as the difference from the target trajectory to a reference trajectory; 2) transform the planning problem with non-convex collision avoidance constraints into a convex optimization by finding a convex feasible set around the reference trajectory; and 3) solve the convex optimization problem in the convex feasible set iteratively until convergence. By

directly using the prior knowledge of the problem in the convexification step, the computation time for CFS is an order of magnitude faster than other generic nonlinear optimization solvers such as SQP [14]. Nonetheless, since an improper convexification of the constraint will inadvertently shrink the size of the feasible sets, it is critical to optimize the approach for the convexification step. In this paper, we extend the CFS algorithm to multi-vehicle trajectory planning problems and propose a novel way to extract the maximal convex feasible set out of a non-convex set. The new algorithm is called multi-car convex feasible set (MCCFS) algorithm. The extensive simulation results demonstrate that the proposed method can generate feasible and smooth trajectories in real-time.

The remainder of this paper is organized as follows. Section 2 reviews the CFS algorithm and formulates the problem for multi-vehicle trajectory planning. Section 3 introduce the core of MCCFS: convexification of the constraints for multi-vehicle interaction. Section 4 presents the simulation results for several typical multi-vehicle driving scenarios. Section 6 concludes the paper.

2 Problem Formulation

This section first introduces the benchmark problem for optimization based trajectory planning, then describes how the CFS algorithm solves the benchmark problem, followed by the specification of the multi-vehicle trajectory planning problem.

2.1 Benchmark Problem

Denote $\mathbf{x}$ as the optimization variable consists of a series of trajectory points $\mathbf{x} := [\mathbf{x}_1; \mathbf{x}_2; \mathbf{x}_3 \ ... \ \mathbf{x}_h]$.The benchmark optimization problem for trajectory planning is:

$$\begin{aligned} \min_{\mathbf{x}} \quad & J(\mathbf{x}) \\ \text{s.t.} \quad & \phi_i(\mathbf{x}) \geq 0 \quad i = 1,2,...l \end{aligned} \tag{1}$$

where $J(\mathbf{x})$ is a cost function, which can penalize acceleration on the trajectory and deviation from a reference trajectory. $\phi_i(\mathbf{x})$ is the ith constraint function, which can be a collision avoidance constraint or a dynamic constraint. For simplicity, the dynamic constraint is ignored in this paper. A method to handle dynamic constraints is discussed in [18]. In addition, we assume that J and ϕ_i for all i are smooth convex functions. For collision avoidance constraint, ϕ_i is chosen as a signed distance function, which is convex for convex objects such as cars. However, when ϕ_i's are convex, the optimization problem constrained on $\phi_i(\mathbf{x}) \geq 0$ is indeed non-convex. In order to efficiently solve the resulting non-convex optimization (1), convexification is needed. In the following discussion, we first review the CFS algorithm, then formulate the multi-car trajectory planning problem by specifying the design of the cost J and the constraint ϕ_i as well as the convexification of ϕ_i.

2.2 Convex Feasible Set (CFS) Algorithm

The CFS algorithm [16] solves (1) iteratively.

1. *Initialization*
   First of all, an initial reference trajectory $\mathbf{x}^{(0)}$ should be obtained. The reference trajectory can be obtained by 1) solving the planning problem using either search based or sampling based approaches [19]; 2) solving the optimization problem in (1) without constraint. In the case that the cost function is designed to penalize the difference between $\mathbf{x}$ and some reference $\mathbf{x}^{ref}$, the reference can be directly used as the initialization, i.e., $\mathbf{x}^{(0)} := \mathbf{x}^{ref}$. After the initialization, the algorithm goes through the following iterations between the convexification step and the optimization step.
2. *Convexification*
   At iteration k, given the solution obtained from the last optimization iteration $\mathbf{x}^{(k-1)}$, the non-convex feasible region $\{\mathbf{x} \mid \phi_i(\mathbf{x}) \geq 0\}$ can be convexified as:

$$F_i^k := \left\{ \mathbf{x} \mid \phi_i(\mathbf{x}^{(k-1)}) + \nabla \phi_i(\mathbf{x}^{(k-1)})^\mathsf{T}(\mathbf{x} - \mathbf{x}^{(k-1)}) \geq 0 \right\} \tag{2}$$

   where F_i^k denotes the convex feasible set of the ith constraint at iteration k. By definition (2), F_i^k is a half space that belongs to the original non-convex feasible region. The convex feasible set for problem (1) is defined as $F^k := \cap F_i^k$. Note that the convex feasible set at iteration k is found with respect to the trajectory $\mathbf{x}^{(k-1)}$ obtained through the optimization at iteration $k-1$.
3. *Optimization*
   At iteration k, once the convex feasible set F^k is obtained, the non-convex problem (1) can be reformulated as a quadratic programming:

$$\mathbf{x}^{(k)} = \underset{\mathbf{x} \in F^k}{\operatorname{argmin}} J(\mathbf{x}) \tag{3}$$

   The solution of the quadratic program is the new trajectory $\mathbf{x}^{(k)}$, which can be used to obtain the next convex feasible set.

It is proved that the above iterations between the convexification step and the optimization step will finally converge to a local optimum of (1) [16].

2.3 Multi-Vehicle Trajectory Planning Problem

Now we specify the multi-vehicle trajectory planning problem, in particular the cost function and the constraints in (1).

Denote the trajectories of n vehicles as $\mathbf{x}^1$, $\mathbf{x}^2$, $\mathbf{x}^3$ ... and $\mathbf{x}^n$. Suppose the planning horizon for each car is h, then the collection of states at time step i is denoted as $\mathbf{x}_i^1, \mathbf{x}_i^2, ..., \mathbf{x}_i^n, i = 1, 2, ..., h$, where $\mathbf{x}_b^a \in \mathbb{R}^2$ is the x, y coordinates of the vehicle a at time step b. The optimization variable $\mathbf{x}$ for the multi-vehicle system is formed by stacking the trajectory points for all vehicles at all time steps:

$$\mathbf{x} := \left[\mathbf{x}_1^1; \mathbf{x}_1^2; \ldots; \mathbf{x}_1^n; \ldots; \mathbf{x}_h^1; \mathbf{x}_h^2; \ldots; \mathbf{x}_h^n\right] \in \mathbb{R}^{2nh} \tag{4}$$

Cost Function The cost function is constructed as a weighted sum of three terms $J(\mathbf{x}) = J_o(\mathbf{x}, \mathbf{x}_{ori}) + J_v(\mathbf{x}) + J_a(\mathbf{x})$. The three terms are explained below.

1. J_o penalizes the difference between the target trajectory $\mathbf{x}$ and a reference trajectory $\mathbf{x}_{ori} \in \mathbb{R}^{2nh}$.

$$J_o = (\mathbf{x} - \mathbf{x}_{ori})^\mathsf{T} \Sigma_o (\mathbf{x} - \mathbf{x}_{ori}) = \mathbf{x}^\mathsf{T} \Sigma_o \mathbf{x} - 2\mathbf{x}_{ori}^\mathsf{T} \Sigma_o \mathbf{x} + \mathbf{x}_{ori}^\mathsf{T} \Sigma_o \mathbf{x}_{ori} \tag{5}$$

   where $\Sigma_o \in \mathbb{R}^{2nh \times 2nh}$ is a diagonal matrix that encodes the weighting factors which may vary for different vehicles at different steps. As the term $\mathbf{x}_{ori}^\mathsf{T} \Sigma_o \mathbf{x}_{ori}$ is a constant, (5) can be simplified as $J_o := \mathbf{x}^\mathsf{T} Q_o \mathbf{x} - 2\mathbf{x}_{ori}^\mathsf{T} \Sigma_o \mathbf{x}$ where $Q_o := \Sigma_o$.
2. $J_v := \mathbf{v}^\mathsf{T} \Sigma_v \mathbf{v}$ penalizes the magnitude of velocity in order to keep the average velocity in a low level. The vector $\mathbf{v} \in \mathbb{R}^{2n(h-1)}$ is the velocity vector along the trajectory, which stacks $\mathbf{v}_b^a := {}^{(\mathbf{x}_{b+1}^a - \mathbf{x}_b^a)}/_{T_s} \in \mathbb{R}^2$ for all vehicles $a = 1, \ldots, n$ at all times steps $b = 1, \ldots, h-1$ where T_s is the sample time.

$$\mathbf{v} := \left[\mathbf{v}_1^1; \mathbf{v}_1^2; \ldots; \mathbf{v}_1^n; \ldots; \mathbf{v}_{h-1}^1; \mathbf{v}_{h-1}^2; \ldots; \mathbf{v}_{h-1}^n\right] \in \mathbb{R}^{2n(h-1)} \tag{6}$$

   $\Sigma_v \in \mathbb{R}^{2n(h-1) \times 2n(h-1)}$ is a diagonal matrix that encodes the weighting factors which may vary for different vehicles at different steps. The relationship between the velocity vector $\mathbf{v}$ and the optimization variable $\mathbf{x}$ satisfies $\mathbf{v} = V\mathbf{x}$ where

$$V := \frac{1}{T_s} \begin{bmatrix} 1 & 0 & \cdots & 0 & -1 & & & & & \\ & 1 & 0 & \cdots & 0 & -1 & & & & \\ & & \ddots & \ddots & \ddots & \ddots & \ddots & & & \\ & & & 1 & 0 & \cdots & 0 & -1 & & \end{bmatrix} \in \mathbb{R}^{2n(h-1) \times (2nh)} \tag{7}$$

   Therefore, the velocity cost can be reformulated as $J_v = \mathbf{x}^\mathsf{T} Q_v \mathbf{x}$ where $Q_v := V^\mathsf{T} \Sigma_v V$.
3. $J_a := \mathbf{a}^\mathsf{T} \Sigma_a \mathbf{a}$ penalizes the magnitude of the acceleration to ensure smoothness of the planned trajectories. The vector $\mathbf{a} \in \mathbb{R}^{2n(h-1)}$ is the acceleration vector along the trajectory, which stacks $\mathbf{a}_b^a := {}^{(\mathbf{v}_{b+1}^a - \mathbf{v}_b^a)}/_{T_s} = {}^{(\mathbf{x}_{b+2}^a - 2\mathbf{x}_{b+1}^a + \mathbf{x}_b^a)}/_{T_s^2} \in \mathbb{R}^2$ for all vehicles $a = 1, \ldots, n$ at all time steps $b = 1, \ldots, h-2$.

$$\mathbf{a} = \left[\mathbf{a}_1^1; \mathbf{a}_1^2; \ldots; \mathbf{a}_1^n; \ldots; \mathbf{a}_{h-2}^1; \mathbf{a}_{h-2}^2; \ldots; \mathbf{a}_{h-2}^n\right] \in \mathbb{R}^{2n(h-2)} \tag{8}$$

   $\Sigma_a \in \mathbb{R}^{2n(h-2) \times 2n(h-2)}$ is a diagonal matrix that encodes the weighting factors which may vary for different vehicles at different steps. The acceleration $\mathbf{a}$ can also be represented using $\mathbf{x}$, i.e., $\mathbf{a} = A\mathbf{x}$ where $A \in \mathbb{R}^{2n(h-2) \times (2nh)}$ and

$$A := \frac{1}{T_s^2} \begin{bmatrix} 1 & 0 \ldots & 0 & -2 & 0 \ldots & 0 & 1 & & & \\ & 1 & 0 \ldots & 0 & -2 & 0 \ldots & 0 & 1 & & \\ & & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & & 1 & 0 \ldots & 0 & -2 & 0 \ldots & 0 & 1 \end{bmatrix} \tag{9}$$

   The acceleration cost can be reformulated as $J_v = \mathbf{x}^\mathsf{T} Q_a \mathbf{x}$ where $Q_a := A^\mathsf{T} \Sigma_a A$.

The cost function can be written as $J(\mathbf{x}) = \mathbf{x}^\mathsf{T}[Q_o + Q_v + Q_a]\mathbf{x} - 2\mathbf{x}_{ori}^\mathsf{T} \Sigma_o \mathbf{x}$. The weights in Σ_o, Σ_v and Σ_a can be customized based on users' preference. These weights encode high-level information that can be easily exchanged among vehicles. When all vehicles have the same preference, the matrices Σ_o, Σ_v and Σ_a can be reduced to three scalars.

Constraints This paper focuses on the collision avoidance constraint, which needs to be enforced between every pair of vehicles at every time step. Hence there should be $l = h\mathbb{C}_n^2 = {}^{hn(n-1)}/_2$ constraints. From now on, when there is no ambiguity, we drop the time index in the notations and use the subscript to denote different vehicles. The square distance between two vehicles $\mathbf{x}_1$ and $\mathbf{x}_2$ can be computed as:

$$D(\mathbf{x}_1, \mathbf{x}_2) := \|\mathbf{x}_1 - \mathbf{x}_2\|^2 = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix}^\mathsf{T} \begin{bmatrix} I & -I \\ -I & I \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} \tag{10}$$

To ensure safety, the distance constraint is posed as:

$$\phi(\mathbf{x}_1, \mathbf{x}_2) := D(\mathbf{x}_1, \mathbf{x}_2) - c \geq 0 \tag{11}$$

where the constant c should be chosen to be greater than or equal to the square of the safety margin between two vehicles.

Challenges The specified cost function J and the constraint function ϕ are smooth and convex that satisfy the assumptions in the benchmark problem. Hence we can apply the CFS algorithm to solve the multi-vehicle trajectory planning problem. However, there are two problems if we naively apply CFS. The first problem is that if we simply linearize the collision avoidance constraint in the convexification step, the resulting convex feasible set may be too small which eliminates many qualified solutions. The second problem is associated with the discrete time sampling of the trajectory. It is possible that two vehicles

cross each other between two time steps. Although the safety constraint is satisfied at both time steps, the trajectory is indeed infeasible as shown in Fig. 4. These problems will be addressed in the following discussions.

3 Multi-Car Convex Feasible Set Algorithm

Multi-car convex feasible set algorithm (MCCFS) is an extension of the CFS algorithm designed specifically for multi-vehicle trajectory planning. The main structure of MCCFS follows the same three steps as in the CFS algorithm. To address the challenges mentioned above, the convexification method will be improved in order to find the maximal convex feasible set; and additional priority constraints will be added to avoid trajectory crossing between two time steps.

3.1 Maximal Convex Feasible Set

The feasible set defined in (11) is non-convex as the function $D(\mathbf{x}_1,\mathbf{x}_2)$ is quadratic. Following the first order approximation defined in (2), the distance constraint can be convexified at a reference point $(\mathbf{x}_1^{ref},\mathbf{x}_2^{ref})$ as:

$$\|\mathbf{x}_1^{ref}-\mathbf{x}_2^{ref}\|^2+2\begin{bmatrix}\mathbf{x}_1^{ref}\\ \mathbf{x}_2^{ref}\end{bmatrix}^{\mathsf{T}}\begin{bmatrix}I & -I\\ -I & I\end{bmatrix}\begin{bmatrix}\mathbf{x}_1-\mathbf{x}_1^{ref}\\ \mathbf{x}_2-\mathbf{x}_2^{ref}\end{bmatrix}-c\geq 0 \quad (12)$$

In the multi-vehicle trajectory planning problem with safety constraint (11), the size of the convex feasible set heavily depends on the reference $(\mathbf{x}_1^{ref},\mathbf{x}_2^{ref})$. It is assumed that the reference never assigns two different vehicles on a same location, i.e., $\mathbf{x}_1^{ref}\neq\mathbf{x}_2^{ref}$. Define $\mathbf{e}$ to be the unit vector along the direction of $\mathbf{x}_1^{ref}-\mathbf{x}_2^{ref}$ and $\lambda:=\|\mathbf{x}_1^{ref}-\mathbf{x}_2^{ref}\|$. Hence $\mathbf{x}_1^{ref}-\mathbf{x}_2^{ref}=\lambda\mathbf{e}$. Then (12) can be reduced to $2\lambda\mathbf{e}^{\mathsf{T}}[\mathbf{x}_1-\mathbf{x}_2]-\lambda^2-c\geq 0$. Dividing λ on both sides, we get

$$2\mathbf{e}^{\mathsf{T}}[\mathbf{x}_1-\mathbf{x}_2]\geq\lambda+\frac{c}{\lambda} \quad (13)$$

Different choice of the reference $(\mathbf{x}_1^{ref},\mathbf{x}_2^{ref})$ affects λ, hence affects the size of the feasible area. When $\lambda\to 0$ or $\lambda\to\infty$, it is almost infeasible to satisfy (13) as the right-hand-side goes to infinity. The feasible area for (13) is maximized when $\lambda=\sqrt{c}$, which implies that $\phi(\mathbf{x}_1^{ref},\mathbf{x}_2^{ref})=\lambda^2-c=0$. Figure 1 also illustrates how the reference point affects the derived convex feasible set for a one dimensional constraint $g(x)\geq 0$.

Although the original convergence proof of the CFS algorithm does not depend on the size of the convex feasible set [17], a smaller convex feasible set will generate smaller improvements of the trajectories during the iterations, hence require more iterations to converge to local optima. To improve computation efficiency, we would like to maximize the size of the convex feasible set during the convexification step. In the following discussion, we will formally introduce the notion of maximal convex feasible set and how to properly select the reference $(\mathbf{x}_1^{ref},\mathbf{x}_2^{ref})$ to attain the maximal convex feasible set. Note the convergence of the CFS algorithm is only proved with respect to the situation that the convex feasible set is generated with respect to the solution from the previous iteration. Hence we should also demonstrate that the new approach in the convexification step will not affect the convergence of the CFS algorithm.

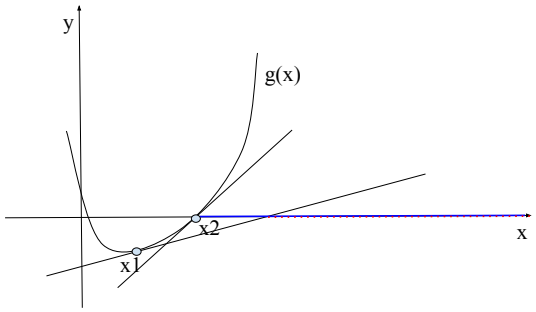


FIGURE 1: Different convex feasible sets obtained at different reference points for constraint $g(x)\geq 0$. Blue solid line is the convex feasible set obtained at reference point x_2. Red dotted line is the convex feasible set obtained at reference point x_1.

Definition 1 (Maximal Convex Feasible Set). *We say a half space F inside a nonconvex set Γ is a maximal convex feasible set with respect to G if F is touching the boundary of Γ, i.e., $\partial F\cap\partial\Gamma\neq\emptyset$.*

The notation ∂F means the boundary of the set F. The definition essentially says that a half space F is a maximal convex feasible set if and only if it cannot move along its normal direction to increase its coverage without violating the feasibility constraint $F\subset\Gamma$. The maximal convex feasible set is not unique.

Theorem 1 (Maximum Condition). *For a smooth convex function ϕ that has negative minimum, the convex feasible set computed in* (2) *using a reference $\mathbf{x}^*$ is maximal with respect to $\phi(\mathbf{x})\geq 0$ if and only if*

$$\phi\left(\mathbf{x}^*-\frac{\phi(\mathbf{x}^*)}{\|\nabla\phi(\mathbf{x}^*)\|^2}\nabla\phi(\mathbf{x}^*)\right)=0 \quad (14)$$

Proof. Considering the definition of the maximal convex feasible set, we only need to show that a reference $\mathbf{x}^*$ satisfies (14) is equivalent to the condition that there exist a point $\mathbf{x}^o$ on the boundary of both the convex feasible set (2) and the original non-

convex constraint, i.e.,

$$\phi(\mathbf{x}^*) + \nabla\phi(\mathbf{x}^*)^\mathsf{T}(\mathbf{x}^o - \mathbf{x}^*) = 0 \tag{15}$$

$$\phi(\mathbf{x}^o) = 0 \tag{16}$$

If (14) holds, then we can set $\mathbf{x}^o$ to be $\mathbf{x}^* - {\phi(\mathbf{x}^*)}/{\|\nabla\phi(\mathbf{x}^*)\|^2}\nabla\phi(\mathbf{x}^*)$. It is easy to check that (15) and (16) hold.

If there exist $\mathbf{x}^o$ that satisfies (15) and (16), then we can compute the relationship between $\mathbf{x}^o$ and $\mathbf{x}^*$ from (15). We first claim that $\nabla\phi(\mathbf{x}^*) \neq 0$. If $\nabla\phi(\mathbf{x}^*) = 0$, then $\mathbf{x}^*$ achieves the minimum of the convex function which should be negative, i.e., $\phi(\mathbf{x}^*) < 0$. In this way, the left-hand-side of (15) is negative and the equality cannot hold. Hence $\nabla\phi(\mathbf{x}^*) \neq 0$. Then by simple arithmetics on (15), we obtain that $\mathbf{x}^o = \mathbf{x}^* - {\phi(\mathbf{x}^*)}/{\|\nabla\phi(\mathbf{x}^*)\|^2}\nabla\phi(\mathbf{x}^*)$. By plugging $\mathbf{x}^o$ to (16), we get (14).

Hence the claim is verified.

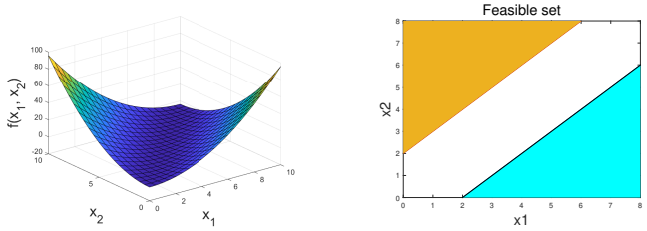
One implication of theorem 1 is that a reference $\mathbf{x}^*$ can lead to a maximal convex feasible set if $\phi(\mathbf{x}^*) = 0$, i.e., the reference $\mathbf{x}^*$ is on the boundary of the constraint. This conclusion aligns with the analysis with (13). For ϕ defined in (11) and $\mathbf{x}^* = [\mathbf{x}_1^{ref}; \mathbf{x}_2^{ref}]$, it is easy to check that (14) holds if and only if $\lambda = \sqrt{c}$, i.e., the reference is on the boundary of the constraint. Given the knowledge of what reference can lead to a maximal convex feasible set, the next question we would like to answer is: for reference points that do not satisfy (14), how can we choose a different reference point to maximize the convex feasible set while not violating the convergence conditions of the CFS algorithm? In the following discussion, we only consider the specific ϕ defined in (11).

Lemma 1 (Choice of Reference for Convexification). *For ϕ in (11) and a reference point $\mathbf{x}^* = [\mathbf{x}_1^{ref}; \mathbf{x}_2^{ref}]$ with $\mathbf{x}_1^{ref} - \mathbf{x}_2^{ref} = \lambda\mathbf{e}$ that (2) is not maximal, consider a set of new points $\mathbf{x}^o$ that satisfies $[I \ -I]\mathbf{x}^o = \sqrt{c}\mathbf{e}$. Then the convex feasible set computed using (2) with respect to $\mathbf{x}^o$ is maximal and is a superset with respect to the convex feasible set computed with respect to $\mathbf{x}^*$.*

Proof. By definition, $\phi(\mathbf{x}^o) = 0$, then the resulting convex feasible set $F(\mathbf{x}^o) = \{\mathbf{x} \mid \nabla\phi(\mathbf{x}^o)^\mathsf{T}(\mathbf{x} - \mathbf{x}^o) \geq 0\}$ is maximal according to theorem 1. Now we need to prove that $F(\mathbf{x}^*) \subseteq F(\mathbf{x}^o)$ where $F(\mathbf{x}^*) = \{\mathbf{x} \mid \phi(\mathbf{x}^*) + \nabla\phi(\mathbf{x}^*)^\mathsf{T}(\mathbf{x} - \mathbf{x}^*) \geq 0\}$ is the convex feasible set computed with respect to $\mathbf{x}^*$. Note that $\nabla\phi(\mathbf{x}^*) = 2\lambda[\mathbf{e}; -\mathbf{e}]$ and $\nabla\phi(\mathbf{x}^o) = 2\sqrt{c}[\mathbf{e}; -\mathbf{e}]$. Hence the normal directions of the two half spaces $F(\mathbf{x}^o)$ and $F(\mathbf{x}^*)$ are aligned. Define $L := [\mathbf{e}; -\mathbf{e}] \in \mathbb{R}^4$. Rewritten the two half space constraints, we get

$$F(\mathbf{x}^o) = \{\mathbf{x} \mid L^\mathsf{T}\mathbf{x} \geq L^\mathsf{T}\mathbf{x}^o = \sqrt{c}\} \tag{17}$$

$$F(\mathbf{x}^*) = \{\mathbf{x} \mid L^\mathsf{T}\mathbf{x} \geq L^\mathsf{T}\mathbf{x}^* - \frac{\lambda^2 - c}{2\lambda} = \frac{\lambda^2 + c}{2\lambda}\} \tag{18}$$



(a) The constraint function. (b) The feasible sets.

FIGURE 2: Illustration of the safety constraints and the convex feasible sets. a) Illustration of the surface of the constraint function $\phi(\mathbf{x}_1, \mathbf{x}_2)$. b) The orange area and the cyan area represent the two maximal convex feasible sets obtained from the constraint function. The original non-convex feasible set is the union of the two colored areas.

Clearly ${\lambda^2+c}/{2\lambda} \geq \sqrt{c}$, and therefore $F(\mathbf{x}^*) \subseteq F(\mathbf{x}^o)$. Hence it is proved that $\mathbf{x}^o$ results in a maximal convex feasible set and it is a superset to the convex feasible set computed with respect to $\mathbf{x}^*$.

The result $F(\mathbf{x}^*) \subseteq F(\mathbf{x}^o)$ implies that we are not loosing any feasible area by changing to a new reference in the convexification step. In this way, it still holds that the iterations between the convexification step (using the modified reference) and the optimization step in the CFS algorithm should converge to a local optimum.

It is worth noting that for single vehicle trajectory planning problems discussed in [19], the problem of getting non-maximal convex feasible set does not arise. That is due to the usage of signed distance functions for the safety constraint as opposed to squared distance functions used in this paper. Since signed distance functions are only piece-wise smooth (squared distance functions are smooth), to properly handle none smooth points, subgradient was introduced [17]. The advantage of signed distance functions is that: all points satisfy the maximum condition in theorem 1 (the gradient needs to be interpreted as the subgradient), i.e., the convex feasible sets are always maximal. We may also use signed distance function for multi-vehicle trajectory planning to avoid changing the reference in the convexification step. Nonetheless, the contribution of this paper is that we introduce 1) a systematic approach to determine whether a convex feasible set is maximal for arbitrary constraints (theorem 1); 2) a way to obtain the maximal convex feasible set with smooth constraint functions without dealing with subgradients.

Examples In one dimension space, the distance constraint becomes $f(x_1, x_2) = (x_1 - x_2)^2 - C \geq 0$. Figure 2a shows the curve of the constraint function. From Fig. 2b, we make two important observations. First, if the reference point is on the intersection line of the constraint function and the zero plane, the linear feasible set is maximal. Second, for any reference point

on the same intersection line in Fig. 2b, the maximal linear feasible set is the same. These observations align with the theoretical results presented earlier and indicate that the choice of reference points can be highly customized.

When describing distance for two vehicles in 2D, the constraint $\phi(\mathbf{x}) \geq 0$ is in four dimension space where $\mathbf{x}$ denotes the x and y positions of the two vehicles. Denote the optimization variable and the reference trajectory point for car 1 as $\mathbf{x}_1 = [x_1, x_2]^\mathsf{T}$, $\mathbf{x}_1^{ref} = [a, b]^\mathsf{T}$, respectively. Similarly, we define $\mathbf{x}_2 = [x_3, x_4]^\mathsf{T}$ and $\mathbf{x}_2^{ref} = [a + \sqrt{c}cos(\theta), b + \sqrt{c}sin(\theta)]^\mathsf{T}$ such that $\phi(\mathbf{x}_1^{ref}, \mathbf{x}_2^{ref}) = 0$ and Lemma 1 is satisfied. θ is a free variable indicating the relative angle between $\mathbf{x}_1^{ref}$ and $\mathbf{x}_2^{ref}$. Substitute the reference points into (12):

$$2 \begin{bmatrix} a \\ b \\ a + \sqrt{c}cos(\theta) \\ b + \sqrt{c}sin(\theta) \end{bmatrix}^\mathsf{T} \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 - a \\ x_2 - b \\ x_3 - a - \sqrt{c}cos(\theta) \\ x_4 - b - \sqrt{c}sin(\theta) \end{bmatrix} \geq 0 \tag{19}$$

Simplify this equation:

$$(x_3 - x_1 - \sqrt{c}cos(\theta))cos(\theta) + (x_4 - x_2 - \sqrt{c}sin(\theta))sin(\theta) \geq 0 \tag{20}$$

Equation (20) shows the feasible set of $\mathbf{x}_1$ and $\mathbf{x}_2$, which has no relation with the location of the reference point $[a, b]$, only the angle θ affects the feasible set.

The angle θ plays an important role in determining the relative location between the newly planned trajectory points of the two cars. Denote vector

$$\mathbf{v}_1 = \left[x_3 - x_1 - \sqrt{c}cos(\theta), x_4 - x_2 - \sqrt{c}sin(\theta)\right]^\mathsf{T} \tag{21}$$

$$\mathbf{v}_2 = [cos(\theta), sin(\theta)]^\mathsf{T} \tag{22}$$

Then (20) becomes $\mathbf{v}_1 \cdot \mathbf{v}_2 \geq 0$, which means the angle between $\mathbf{v}_1$ and $\mathbf{v}_2$ should be smaller than $90°$.

Figure 3 shows the relative location between the newly planned trajectory points of car 1 and car 2 with respect to θ. In the figure, E is car 1's trajectory point, G is car 2's trajectory point and F is a point that has a distance $\sqrt{c}$ from point E with an angle of θ. Hence $\overrightarrow{EF} = \sqrt{c}\mathbf{v}_2$, $\overrightarrow{FG} = \mathbf{v}_1$. As (20) requires $\mathbf{v}_1 \cdot \mathbf{v}_2 \geq 0$, then $\overrightarrow{EF} \cdot \overrightarrow{FG} \geq 0$, which indicates that car 2's point G should locate in the shadowed area once car 1's point is located at E.

3.2 Priority Constraint

To address the problem that trajectories of two vehicles may cross between two consecutive time steps, we introduce the priority constraints to specify which car can go first. As

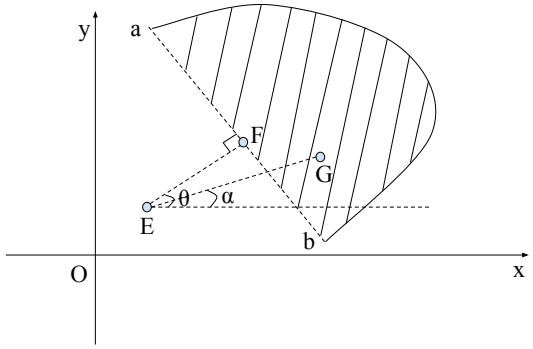


FIGURE 3: Feasible set of two cars' location points. The shadowed area represents the feasible set of car 2's location(point G) once car 1 locates in point E.

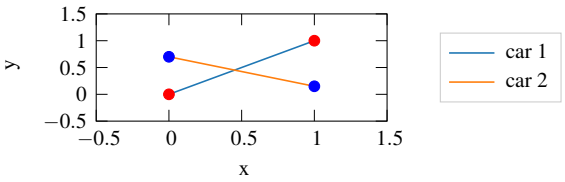


FIGURE 4: Trajectory crossing between two consecutive time steps.

shown in Fig. 4, the trajectories of car 1 and car 2 form two 2D lines $k_1\mathbf{x} + b_1 = 0$ and $k_2\mathbf{x} + b_2 = 0$ respectively. Originally, for car 2's two trajectory points $\mathbf{x}_i^2$ and $\mathbf{x}_{i+1}^2$, there would be $(k_1\mathbf{x}_i^2 + b_1) \cdot (k_1\mathbf{x}_{i+1}^2 + b_1) < 0$ as these two points lay on different sides of car 1's trajectory. So does car 1. To make the car with higher priority pass first, the other car should wait at the same side of the high priority car's trajectory. In this example, if car 1 has a higher priority, then car 1's trajectory points should still locate at both sides of car 2's trajectory:

$$(k_2\mathbf{x}_i^1 + b_2) \cdot (k_2\mathbf{x}_{i+1}^1 + b_2) < 0 \tag{23}$$

Meanwhile, car 2's trajectory points should remain on one side of car 1's trajectory:

$$(k_1\mathbf{x}_i^2 + b_1) \cdot (k_1\mathbf{x}_{i+1}^2 + b_1) > 0 \tag{24}$$

These constraints are added to the optimization problem to resolve the trajectory crossing.

4 Simulation Results

To demonstrate the effectiveness of the MCCFS algorithm, three typical driving scenarios are simulated. All the simulation

experiments and optimization computation are conducted using Thinkpad X1 Carbon with a Intel Core i7-7600U 2.80GHz (Up to 3.90GHz) Processor with Python scripts.

4.1 Intersection Scenario

In an intersection scenario, two cars cross each other. The planner gives car 2 a higher priority and plans the trajectories as shown in Fig. 5b. Without the prioritized planner, car 1 and car 2 will arrive at the middle point at the same time which will result in collision. On the contrary, our proposed planner generates the reference trajectories such that car 1 decelerates and car 2 accelerates before the middle point to enable car 2 cross first. Both car 1 and car 2 have a slight turn, which satisfies the distance constraint and decreases the acceleration cost. The computation time to find the feasible solution in this scenario only takes about 0.9 seconds.

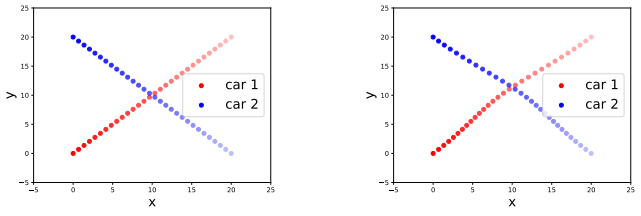
4.2 Overtake Scenario

Another typical situation that is really challenging for decentralized planner but relatively easier for centralized planner is the three car overtake scenario as shown in Fig. 6. In reality, it is difficult for car 2 and car 3 to avoid collision without communication or coordination with each other. Sudden break or acceleration might occur if the intentions of other cars are misunderstood. However, using the centralized planner, this problem is well solved as shown in Fig. 7. The speed profiles of car 2 and car 3 are shown in Fig. 8. To overtake car 1 and avoid car 3, car 2 accelerates at the beginning meanwhile car 3 decelerates. This cooperation is better than the case that only car 2 accelerates or only car 3 decelerates as it will make the total acceleration cost smaller. In this scenario, the total computation time is about 7 seconds.

4.3 Crowded Nine-Car Scenario

In urban driving, there can be a lot of cars on the road and each of them can have different intentions. Hence a 9 car driving scenarios is simulated using the MCCFS algorithm. The result is shown in Fig. 9.

As shown in Fig. 9, every car's trajectory is smooth and is able to conduct lane change together. To keep safety distance with car 2, car 9 does a slight right turn when approaches car 2. Due to the high dimension of the state variable and a large number of constraints, the optimization time in this scenario goes up to 27 seconds.



(a) Original Trajectory (b) Planned Trajectory

FIGURE 5: Planning results for the stop sign scenario. a) The original trajectories for two cars in the stop sign and they cross with each other at the mid point. b) The new planned trajectories.

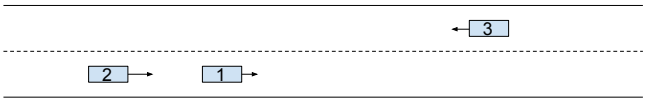


FIGURE 6: 3 car overtake scenario. Car 1 and car 2 are driving in the same direction and car 2 want to overtake car 1, meanwhile car 3 is driving toward car 2.

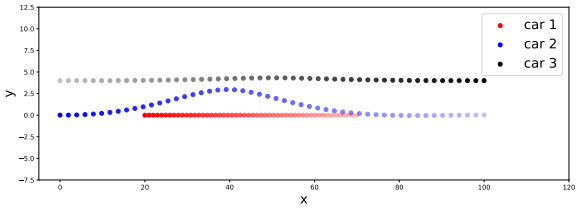


FIGURE 7: Planned trajectories of all three cars in the overtake scenario.

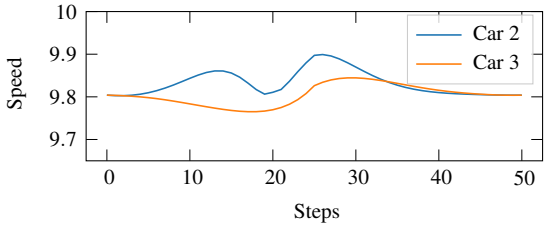


FIGURE 8: Speed profile of car 2 and car 3 in the overtake scenario.

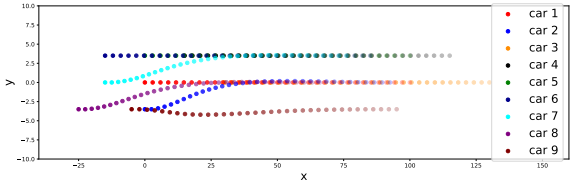


FIGURE 9: Trajectories planned in 9 car scenario.

5 Run-time Statistics

The computation time of the algorithm significantly depends on the number of cars and the number of steps of the reference trajectories. Experiments are conducted to analyze the relations among them.

1. *Computation time w.r.t number of cars*
   In this experiments, number of cars are set from 2 to 9 while each car has a reference trajectory that contains 35 steps. For each parameter set, the algorithm is run for 5 times and the average time is calculated for analysis. The result is shown in Fig. 10. It is obvious that the computation time is $O(n^2)$ with regard to the number of cars. The reason is that since the distance constraints are enforced between every pair of cars, hence given n cars, the number of constraints is proportional to $\mathbb{C}_n^2 = n(n-1)/2$, which affects the computation time quadratically.
2. *Computation time w.r.t trajectory steps*
   In this experiments, three cars are considered and the trajectory steps of them are set differently. Among those trajectory steps, some of them take 2 iterations for the planner to converge and some of them take 3 iterations. The result is shown in Fig. 11. In the cases that take the same number of iterations to converge, the computation time increases linearly with the number of trajectory steps. This is because the number of constraints increases linearly with regard to the number of trajectory steps.

6 Conclusion

This paper proposed an optimization based multi-car trajectory planning method that provides trajectories for multiple cars at the same time. It can properly handle multi-vehicle interactions and generate smooth trajectories. By linearizing the non-convex distance constraint with respect to certain reference points, we transform the non-convex multi-vehicle trajectory planning problem into a quadratic programming which is easy and efficient to solve. The most important contribution in this algorithm is that we formalize the definition of the maximal convex feasible set and propose methods to achieve the maximal convex feasible set. It is shown that the maximal convex feasible set doesn't depend on the location but only the relative angle between the reference points of two cars as long as the reference points are along the boundary of safety constraint, which provides great freedom in choosing the reference. Simulation results of three typical on-road driving situations validate the effectiveness of the proposed algorithm.

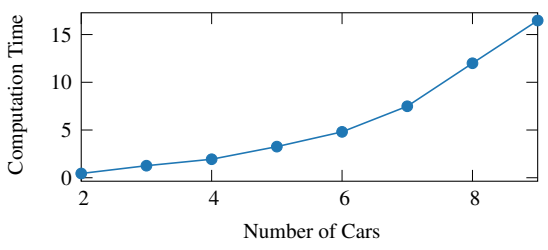


FIGURE 10: Computation time with regard to number of cars.

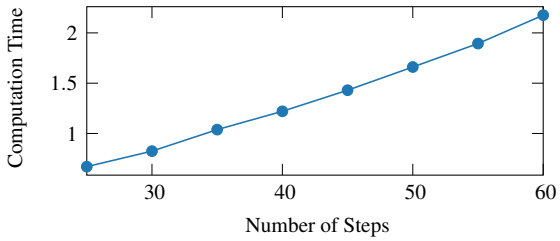


FIGURE 11: Computation time with regard to trajectory steps.

The computation time for multi-vehicle trajectory planning takes longer than for single vehicle trajectory planning as the state variable and number of constraints are significantly growing than in the single car scenario. The computation time grows linearly with respect to the number of constraints, which then grows in the order of $O(n^2)$ with respect to the number of cars and $O(h)$ with respect to the number of trajectory steps. In the future, we will eliminate constraints that are not effective and implement the algorithm in C++, which should dramatically improve the computation speed.

ACKNOWLEDGMENT

The authors would like to thank Rong Hu and Weiye Zhao for valuable suggestions to this paper.

REFERENCES

[1] Liu, C., and Tomizuka, M., 2016. "Enabling safe freeway driving for automated vehicles". In 2016 American Control Conference (ACC), IEEE, pp. 3461–3467.

[2] Hubmann, C., Becker, M., Althoff, D., Lenz, D., and Stiller, C., 2017. "Decision making for autonomous driving considering interaction and uncertain prediction of surrounding vehicles". In 2017 IEEE Intelligent Vehicles Symposium (IV), IEEE, pp. 1671–1678.

[3] Zhan, W., Liu, C., Chan, C.-Y., and Tomizuka, M., 2016. "A non-conservatively defensive strategy for urban autonomous driving". In 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC), IEEE, pp. 459–464.

[4] Liu, C., Lin, C.-W., Shiraishi, S., and Tomizuka, M., 2018. "Improving efficiency of autonomous vehicles by v2v communication". In 2018 Annual American Control Conference (ACC), IEEE, pp. 4778–4783.

[5] Pallottino, L., Scordio, V. G., Bicchi, A., and Frazzoli, E., 2007. "Decentralized cooperative policy for conflict resolution in multivehicle systems". *IEEE Transactions on Robotics,* **23**(6), pp. 1170–1176.

[6] Barraquand, J., Langlois, B., and Latombe, J.-C., 1992. "Numerical potential field techniques for robot path planning". *IEEE Transactions on Systems, Man, and Cybernetics,* **22**(2), March/April, pp. 224–241.

[7] Barraquand, J., and Latombe, J.-C., 1991. "Robot motion planning: A distributed representation approach". *The International Journal of Robotics Research,* **10**, pp. 628–649.

[8] Erdmann, M., and Lozano-Pérez, T., 1987. "On multiple moving objects". *Algorithmica*, p. 477.

[9] Van Parys, R., and Pipeleers, G., 2016. "Online distributed motion planning for multi-vehicle systems". In 2016 European Control Conference (ECC), IEEE, pp. 1580–1585.

[10] Liu, C., Lin, C.-W., Shiraishi, S., and Tomizuka, M., 2017. "Distributed conflict resolution for connected autonomous vehicles". *IEEE Transactions on Intelligent Vehicles,* **3**(1), pp. 18–29.

[11] Luna, R., and Bekris, K. E., 2011. "Efficient and complete centralized multi-robot path planning". In IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, pp. 3268–3275.

[12] Zhang, Z., and Zhao, Z., 2014. "A multiple mobile robots path planning algorithm based on a-star and dijkstra algorithm". *International Journal of Smart Home,* **8**(3), pp. 75–86.

[13] Kala, R., and Warwick, K., 2011. "Multi-vehicle planning using rrt-connect". *Paladyn, Journal of Behavioral Robotics,* **2**(3), pp. 134–144.

[14] Chen, J., Liu, C., and Tomizuka, M., 2018. "FOAD: Fast optimization-based autonomous driving motion planner". In 2018 Annual American Control Conference (ACC), IEEE, pp. 4725–4732.

[15] Borrelli, F., Subramanian, D., Raghunathan, A. U., and Biegler, L. T., 2006. "MILP and NLP techniques for centralized trajectory planning of multiple unmanned air vehicles". In 2006 American Control Conference, IEEE, p. 6.

[16] Mellinger, D., Kushleyev, A., and Kumar, V., 2012. "Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams". In 2012 IEEE International Conference on Robotics and Automation, IEEE, pp. 477–483.

[17] Liu, C., Lin, C., and Tomizuka, M., 2018. "The convex feasible set algorithm for real time optimization in motion planning". *SIAM Journal on Control and Optimization,* **56**(4), pp. 2712–2733.

[18] Liu, C., and Tomizuka, M., 2017. "Real time trajectory optimization for nonlinear robotic systems: Relaxation and convexification". *Systems & Control Letters,* **108**, pp. 56–63.

[19] Liu, C., Lin, C.-Y., Wang, Y., and Tomizuka, M., 2017. "Convex feasible set algorithm for constrained trajectory smoothing". In 2017 American Control Conference (ACC), IEEE, pp. 4177–4182.