978-1-5386-5428-6/$31.00 ©2018 AACC

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).

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

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 $\mathbf{x}^{(k)}$: $\mathcal{F}(\mathbf{x}^{(k)}) \subset \Gamma$. 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

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

Note that $\mathcal{F}_i(\mathbf{x}^{(k)})$ is actually a polytope and is convex. Thus, $\mathcal{F}(\mathbf{x}^{(k)}) = \cap_i \mathcal{F}_i(\mathbf{x}^{(k)})$ 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 $\mathbf{x}^{(k+1)}$ is calculated as the optimal value in the convex feasible set $\mathcal{F}(\mathbf{x}^{(k)})$:

$$\mathbf{x}^{(k+1)} = \arg\min_{\mathbf{x} \in \mathcal{F}(\mathbf{x}^{(k)})} J(\mathbf{x}) \tag{3}$$

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., $\|\mathbf{x}^{(n)} - \mathbf{x}^{(n-1)}\| < \varepsilon$ or $|J(\mathbf{x}^{(n)}) - J(\mathbf{x}^{(n-1)})| < \sigma$.

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).

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

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 $\mathbf{z} = [\mathbf{x}^T, \mathbf{s}^T]^T$ and the augmented distance function $\varphi_i(\mathbf{z}) = \phi_i(\mathbf{x}) + \mathbf{s}$:

$$\begin{aligned} \mathcal{F}_i(\mathbf{z}^{(k)}) &= \{\mathbf{z} : \varphi_i(\mathbf{z}^{(k)}) + \nabla\varphi_i(\mathbf{z}^{(k)})(\mathbf{z} - \mathbf{z}^{(k)}) \geq 0\} \\ &= \phi_i(\mathbf{x}^{(k)}) + \mathbf{s}^{(k)} \\ &+ \nabla\phi_i(\mathbf{x}^{(k)}) \mathbf{1}^T \begin{bmatrix} \mathbf{x} \\ \mathbf{s} \end{bmatrix} - \begin{bmatrix} \mathbf{x}^{(k)} \\ \mathbf{s}^{(k)} \end{bmatrix} \geq 0 \\ &= \{\mathbf{z} : L_i\mathbf{z} + S_i \leq 0\} \end{aligned} \tag{5}$$

Then we have:

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

where $L = [L_1^T \; L_2^T \; \cdots]^T$, $S = [S_1^T \; S_2^T \; \cdots]^T$.

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

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

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: $\|\mathbf{z}^{(n)} - \mathbf{z}^{(n-1)}\| < \varepsilon$ or $|R(\mathbf{z}^{(n)}) - R(\mathbf{z}^{(n-1)})| < \sigma$.

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

Fig. 5. The polygon representation of the obstacle

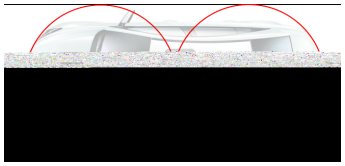

Fig. 6. The vehicle discs containing the ego vehicle body

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 $\overrightarrow{x_k x_{k+1}}$. 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 $\mathsf{y} = F\mathsf{x} \in R^{2(H+1)}$, 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 $\phi_i(\mathsf{x}) = d_i(\mathsf{y}) : R^{2(H+1)} \rightarrow R^{2(H+1)}$ is a vector function whose image is the vector of the signed distances at each time step. For example, the $2k$th value of $\phi_i(\mathsf{x})$ 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):

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

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

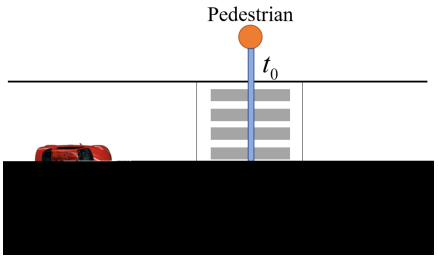

Fig. 7. The crossroad scenario with pedestrian

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 $K = ceil\ \ \frac{t_1}{T_s}$, then the related distance function is $\tau(\mathsf{x}) = d(\mathsf{y}) : R^{2(H+1)} \rightarrow R^{2(K+1)}$, where the $2k$th value of $\tau(\mathsf{x})$ 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 $\tau(\mathsf{x}) = d(\mathsf{y}) : R^{2(H+1)} \rightarrow R^{2(H+1)}$, where the $2k$th value of $\tau(\mathsf{x})$ 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 $\tau(\mathsf{x}) = d(\mathsf{y}) : R^{2(H+1)} \rightarrow R^{2(H+1\ \ K)}$ where the $2k$th value of $\tau(\mathsf{x})$ 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.

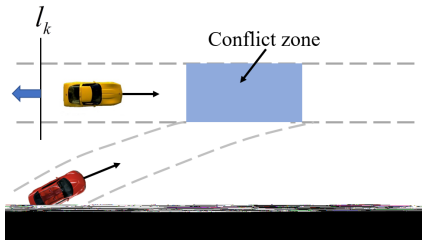

Fig. 8. The merging scenario

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:

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

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

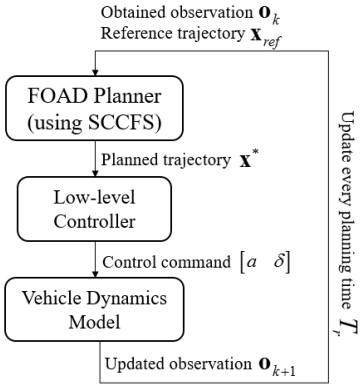

Fig. 10. The simulator scheme

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 $a \quad \delta$ 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