Autonomous Driving Motion Planning with Constrained Iterative LQR

Jianyu Chen, *Student Member, IEEE,* Wei Zhan, *Student Member, IEEE,* Masayoshi Tomizuka, *Life Fellow, IEEE*

Abstract—**Motion planning is a core technique for autonomous driving. Nowadays, there still exists a lot of challenges in motion planning for autonomous driving in complicated environments due to: 1) The need of both spatial and temporal planning in highly dynamic environments; 2) Nonlinear vehicle dynamic models and non-convex collision avoidance constraints. 3) The need of high computation efficiency for real-time implementation. Iterative Linear Quadratic Regulator (ILQR) is an algorithm to solve the optimal control problem with nonlinear system very efficiently. However, it can not deal with constraints. In this paper, the Constrained Iterative LQR (CILQR) is proposed to efficiently solve the optimal control problem with nonlinear system dynamics and general form of constraints. An autonomous driving motion planning problem is then formulated and solved using CILQR. Simulation case studies show the capability of the CILQR algorithm to solve this kind of problems. And the computation efficiency of CILQR is shown to be much higher than the standard SQP solver.**

Index Terms—**Autonomous Driving, Motion Planning, Constraints, Iterative LQR**

I. INTRODUCTION

MOTION planning is a challenging area for autonomous driving. The planning module receives high-level decisions or behaviors from decision-making and behavior generation module, as well as a dynamic world model with road structure and states of all detected obstacles from perception module. The module finally generates trajectories satisfying safety and feasibility constraints with desirable driving quality.

Typically, the representation of the constraints for collision avoidance in motion planners is relatively complex. Also, the trajectories need to be generated in a spatiotemporal domain in order to deal with highly dynamic driving scenarios, such as lane change and overtaking with moving obstacles. When generating motions in a spatiotemporal domain with a relatively long horizon, the computational load is often intractable so that the autonomous vehicle cannot respond to emergencies in real time.

Therefore, autonomous vehicles need a motion planner which can 1) generate long-term motions in a spatiotemporal domain, 2) take into account complex collision avoidance constraints, as well as vehicle kinematic and dynamic models, and 3) achieve real-time computation so that emergency situations can be handled timely.

Our approach utilizes the analytical expressions in Linear Quadratic Regulator (LQR) to design a motion planner with fast computation for complicated planning problems. Iterative LQR (ILQR) can be efficiently solved by dynamic programming (DP) to deal with unconstrained problems with non-quadratic objectives and nonlinear models. Constrained Iterative LQR (CILQR) is proposed in this paper to deal with constrained problems for motion planning of autonomous vehicles, so that the aforementioned requirements can be satisfied.

The main contributions of this paper can be summarized as follows:

- **New Fast Motion Planning Solver Proposed:** We proposed a version of Constrained Iterative LQR algorithm to solve optimal control problems with nonlinear system dynamics and general form of constraints. We use logarithmic barrier function and introduce a new outer-inner loop framework to handle the constraints for Iterative LQR, achieving fast computation.
- **Convergence Property Proved:** The convergence of our proposed outer-inner loop iterative algorithm is proved in this paper, which guarantees the safety and performance of applying the algorithm.
- **Applied to Autonomous Driving Problem Settings:** The proposed algorithm is then applied to an autonomous driving motion planning problem. The performance is evaluated with some challenging simulation scenarios.

II. RELATED WORKS

A. Autonomous Driving Motion Planning

Considerable amount of efforts [1] [2] have been devoted to the area of motion planner design for autonomous driving. Spatiotemporal planning frameworks were proposed based on state-lattice search algorithm [3] and numerical optimization method [4]. The computational load was high for spatiotemporal planning frameworks with relatively long preview horizon, so that the planner had to suffer relatively long runtime [4] or use GPU to achieve real-time computation [3].

The spatiotemporal domain can be partitioned to alleviate the computational load. Typically, the domain was temporally-partitioned into path planning (without temporal dimension) and speed profile planning (with temporal dimension). Such temporally-partitioned architecture was employed in [5] and [6], in which fast computation was achieved. However, as was pointed out in [7], the temporally-partitioned architecture is not suitable for lane change and overtaking scenarios with moving obstacles. A spatially-partitioned planning architecture

was proposed in [7] with corresponding generic environmental representation, which partitioned the domain into longitudinal and lateral motion in a Frenét Frame. Desirable long-term trajectories can be generated in various kinds of scenarios in real time. The limitation is that when the difference is large between the curvature of the traffic-free reference path and that of the final path, undesirable speed profile may be generated. Therefore, if the computational load is satisfactory, spatiotemporal planning is still worth exploiting.

Despite of the viewpoint of planning architecture, the works on motion planning can also be categorized by discretized or continuous space. Methods in discretized space, such as A* search [8], D* search [9], state-lattice search [3] and Rapidly-exploring Random Tree (RRT) [10], generate non-smooth trajectories. Also, the sampling resolution significantly impacts the computational load due to the "curse of dimensionality". Methods in continuous space, such as optimization-based [4] and learning-based [11] [12] approaches, can overcome such deficiencies. Learning-based methods are flexible to learn expert or human-like driving policies and the computational load is not sensitive to the dimensionality. However, it is still challenging for learning-based methods to have theoretical guarantee for long-term safety and feasibility constraints.

The optimization-based method formulates the motion planning problem as a mathematical optimization problem. There are several advantages of optimization-based method over other methods. First, it allows continuous planning because the state and action space is a subset of the Euclidean space. Second, it can evaluate multiple objectives (such as constraints) in a uniform formulation. Also, the computation does not increase exponentially with the increase of dimension.

B. Numerical Optimization

The optimization-based method can be roughly divided into two categories: direct collocation and indirect shooting [13]. The main idea of the direct collocation method is to consider points on the trajectory as the decision variables, and improve the trajectory by directly changing the trajectory points. Numerical optimization algorithms [14] such as sequential quadratic programming (SQP) [15] are the main tools for this method. An example of this method in autonomous driving is the work by Mercedes-Benz [4], [16], which utilized SQP for autonomous driving motion planning.

An advantage for direct collocation method is the elimination of system dynamic equation constraints. All the control inputs are formulated using inverse dynamics. This elimination can significantly improve the computation efficiency. However, for problems with long preview horizon, nonlinear dynamics and non-convex constraints, the computation load is still intractable.

Efforts have been made to improve the efficiency of optimization algorithms. For example, Liu, Lin and Tomizuka proposed the convex feasible set (CFS) algorithm for efficient trajectory optimization [17]. It has been applied to robot collision avoidance and autonomous driving motion planning [18], [19]. Although CFS makes the real-time implementation possible, the use of inverse dynamics makes the resulting optimal control

input inaccurate, and a low level controller is required to track the planned trajectory. This may introduce a mismatch problem between the planned trajectory and the actual trajectory.

Although some general numerical optimization tools can handle dynamic equality constraints directly, the additional equality constraints will significantly increase the computation load as we will show in Section VIII-D.

C. Optimal Control and Iterative LQR

Another category of optimization-based motion planning is the shooting method. The main idea is to consider the control inputs as the decision variables and the trajectory can be obtained by forward simulation using the system dynamic equation. This category is closely related to the optimal control theory [20], [21]. For specific problems, optimal control theory can give an analytical solution such as the Riccati equation for linear systems. However, for problems with nonlinear systems and non-convex constraints, it is extremely difficult or even impossible to obtain an analytical solution.

Differential Dynamic Programming (DDP) [22], [23] is an efficient numerical method which can solve the unconstrained optimal control problem. Iterative Linear Quadratic Regulator (ILQR) [24], [25] is a simplified and faster version of DDP. However, only very few literatures considered constraints in ILQR, while constraints are inevitable in autonomous driving motion planning. The control-limited DDP was proposed in [26], which solves the nonlinear optimal control problems under only the control input constraints. Extended LQR [27], [28] considered collision avoidance, but it is achieved by adding a distance cost and thus can not ensure hard constraints. Furthermore, it only deals with circle obstacles. To our knowledge, there is still a lack of methods that can handle general form of constraints using ILQR, and there is no work using ILQR to deal with the general on-road autonomous driving motion planning with complex constraints.

III. PROBLEM STATEMENT

What is a motion planning problem? Imagine you are an agent, you can get observations from the world, and can execute actions to influence the world. Every time you execute an action, you will get a reward from the world. The planning problem is to find the best action sequence which will return the maximum total reward. Since physical objects are usually controlled by computer, we will consider the discrete time formulation shown below:

Problem 1 (General Motion Planning Problem).

$$x^*, u^* = \arg\min_{x,u} \left\{ \phi(x_N) + \sum_{k=0}^{N-1} L^k(x_k, u_k) \right\} \quad (1a)$$

$$s.t. \quad x_{k+1} = f^k(x_k, u_k), \quad k = 0, 1, ..., N-1 \quad (1b)$$

$$x_0 = x_{start} \quad (1c)$$

$$g(x, u) < 0 \quad (1d)$$

$$h(x, u) = 0 \quad (1e)$$

where x_k is the state vector at time step k, and u_k is the control input vector at time step k. N is the preview horizon. (1a) is the cost function, where $\phi(x_N)$ is the final stage cost and $L^k(x_k, u_k)$ is the stage cost at time step k. (1b) are the system dynamic equations, (1c) is the initial state constraint. (1d) are the inequality constraints and (1e) are the equality constraints, where $x = \begin{bmatrix} x_1^T & \cdots & x_N^T \end{bmatrix}^T$ and $u = \begin{bmatrix} u_1^T & \cdots & u_{N-1}^T \end{bmatrix}^T$ are the state vector and the control input vector for the whole trajectory, respectively.

This formulation is quite general and covers a large class of problems. It is also an optimization problem, which can be solved by numerical optimization solvers such as SQP. However, these numerical solvers are designed for solving general optimization problems, which makes it not efficient enough for real time motion planning. Actually, there are some special structures of the motion planning problem, which makes the motion planning different from the general optimization problem. Thus we change the problem formulation into the following one:

Problem 2 (Discrete-time Finite-horizon Motion Planning Problem).

$$x^*, u^* = \arg\min_{x,u} \left\{ \phi(x_N) + \sum_{k=0}^{N-1} L^k(x_k, u_k) \right\} \quad (2a)$$

$$s.t. \quad x_{k+1} = f^k(x_k, u_k), \quad k = 0, 1, ..., N-1 \quad (2b)$$

$$x_0 = x_{start} \quad (2c)$$

$$g^k(x_k, u_k) < 0, \quad k = 0, 1, ..., N-1 \quad (2d)$$

$$g^N(x_N) < 0 \quad (2e)$$

where the objective function (2a), the system dynamic equations (2b), and the initial state constraints (2c) are the same with Problem 1. The following additional assumptions are introduced:

Assumption 1. The system dynamic equation constraints are the only constraints in the problem. Here we eliminate the equality constraints such as (1e) in Problem 1.

Assumption 2. The inequality constraints (2d) and (2e) are separated to different time steps.

Assumption 3. All the functions in Problem 2 have continuous first and second order derivatives.

These assumptions are actually not very strict. Most motion planning problems naturally satisfy assumptions 1 and 2, and we can easily define the functions to have continuous first and second derivatives. Problem 2 will be considered in the remainder of this paper.

IV. ITERATIVE LQR (ILQR)

According to our discussion in section I and II, DDP and ILQR are efficient numerical tools to solve the unconstrained planning problems. Here we briefly describe the algorithm.

Problem 3 (Unconstrained Motion Planning Problem). The

Iterative LQR algorithm can deal with the following nonlinear system motion planning problem

$$x^*, u^* = \arg\min_{x,u} \left\{ \phi(x_N) + \sum_{k=0}^{N-1} L^k(x_k, u_k) \right\} \quad (3a)$$

$$s.t. \quad x_{k+1} = f^k(x_k, u_k), \quad k = 0, 1, ..., N-1 \quad (3b)$$

$$x_0 = x_{start} \quad (3c)$$

Note that compared to (2), everything remains the same except that now the constraints (2d) and (2e) are eliminated.

The theoretical foundation of DDP/ILQR is dynamic programming, specifically, the following Bellman equation:

$$V^k(x_k) = \min_{u_k} \left[L^k(x_k, u_k) + V^{k+1}(f^k(x_k, u_k)) \right] \quad (4)$$

where $V^k(\cdot)$ is the minimum cost to go starting at x_k. The algorithm works with the following steps:

Step 1 (Nominal Trajectory Definition). First a nominal trajectory $(\bar{x}, \bar{u})$ is defined. In the first iteration, a feasible initial trajectory needs to be given as the nominal trajectory.

Step 2 (Backward Pass). For the final time step, we have the final step value function $V^N(x_N) = \phi(x_N)$. Then start from step N-1, define the perturbation term for each time step as follows:

$$P^k(\delta x_k, \delta u_k) = L^k(\bar{x}_k + \delta x_k, \bar{u}_k + \delta u_k) - L^k(\bar{x}_k, \bar{u}_k) \\ + V^{k+1}(f^k(\bar{x}_k + \delta x_k, \bar{u}_k + \delta u_k)) - V^{k+1}(f^k(\bar{x}_k, \bar{u}_k)) \quad (5)$$

Approximate $P^k(\delta x_k, \delta u_k)$ by its second order Taylor expansion to get:

$$P^k(\delta x_k, \delta u_k) \approx \\ \frac{1}{2} \begin{bmatrix} 1 \\ \delta x_k \\ \delta u_k \end{bmatrix}^T \begin{bmatrix} 0 & (P_x^k)^T & (P_u^k)^T \\ P_x^k & P_{xx}^k & P_{xu}^k \\ P_u^k & P_{ux}^k & P_{uu}^k \end{bmatrix} \begin{bmatrix} 1 \\ \delta x_k \\ \delta u_k \end{bmatrix} \quad (6)$$

where

$$P_x^k = L_x^k + (f_x^k)^T V_x^{k+1} \quad (7a)$$

$$P_u^k = L_u^k + (f_u^k)^T V_x^{k+1} \quad (7b)$$

$$P_{xx}^k = L_{xx}^k + (f_x^k)^T V_{xx}^{k+1} f_x^k + V_x^{k+1} \cdot f_{xx}^k \quad (7c)$$

$$P_{uu}^k = L_{uu}^k + (f_u^k)^T V_{xx}^{k+1} f_u^k + V_x^{k+1} \cdot f_{uu}^k \quad (7d)$$

$$P_{ux}^k = L_{xx}^k + (f_x^k)^T V_{xx}^{k+1} f_x^k + V_x^{k+1} \cdot f_{ux}^k \quad (7e)$$

Then the optimal control strategy is:

$$\delta u_k^* = \arg\min_{\delta u_k} P^k(\delta x_k, \delta u_k) = -(P_{uu}^k)^{-1}(P_u^k + P_{ux}^k \delta x_k) \\ = q^k + Q^k \delta x_k \quad (8)$$

The derivatives and Hessians of the value function are:

$$V_x^k = P_x^k - P_u^k(P_{uu}^k)^{-1} P_{ux}^k \\ V_{xx}^k = P_{xx}^k - P_{xu}^k(P_{uu}^k)^{-1} P_{ux}^k \quad (9)$$

The computation is repeated until reaching the first time step.

Step 3 (Forward Pass). After the backward pass, we need to forward simulate the actual trajectory using the system dynamic equation and the optimal control strategy (8):

$$x_0 = x_{start}$$
$$u_k = \bar{u}_k + q_k + Q_k (x_k - \bar{x}_k) \qquad (10)$$
$$x_{k+1} = f^k (x_k, u_k)$$

Once an iteration from step 1 to step 3 is completed, the nominal trajectory $(\bar{x}, \bar{u})$ will be replaced by the new actual trajectory (x, u), which improves the performance. By going back and iteratively apply step 1 and step 2, the nominal trajectory will asymptotically converge to the optimal trajectory.

Note that the computation of the Hessians (7c), (7d) and (7e) can be simplified by eliminating their last terms. This simplification transforms DDP into ILQR. The avoidance of computing tensor multiplications makes ILQR generally more efficient than DDP.

Although ILQR is a very efficient algorithm to solve motion planning problems with nonlinear system dynamics, a drawback of ILQR is the disability of dealing with inequality constraints, which is essential in most robotic motion planning scenarios. Taking autonomous driving for example, there are inequality constraints for collision avoidance and actuator limits. Without the ability to deal with inequality constraints, the power of ILQR is significantly limited.

V. CONSTRAINED ITERATIVE LQR (CILQR) ALGORITHM

The proposed CILQR algorithm aims to solve Problem 2 by extending the ILQR algorithm. It is pretty hard to directly handle constraints in ILQR. For example, if we add constraints to (4), since the added constraints only apply at a certain time step, during forward pass the cumulated trajectory change may make the constraints violated. The control-limited DDP [26] added constraints to (4), but only for control inputs. However, for state constraints this can not work.

An alternative way to handle constraints is introducing penalties. The basic idea is to use a barrier function to shape the constraint functions (2d) and (2e):

$$c(x, u) = b(g(x, u)) \qquad (11)$$

Then the barrier function is added to the objective function. The ideal barrier function is the indicator function:

$$b^*(g(x, u)) = \begin{cases} \infty, & g(x, u) \geq 0 \\ 0, & g(x, u) < 0 \end{cases} \qquad (12)$$

which can represent the original constraints. However, since the indicator function is not differentiable, the augmented objective function cannot be optimized by most numerical optimization methods. Therefore, we need to find a barrier function that can approximate the indicator function (12) while being differentiable.

Our previous work [29] used an exponential barrier function:

$$b(g(x, u)) = q_1 \exp(q_2 g(x, u)) \qquad (13)$$

where q_1 and q_2 are parameters. The constraints are incorporated into the ILQR algorithm by a four-step process with shaping
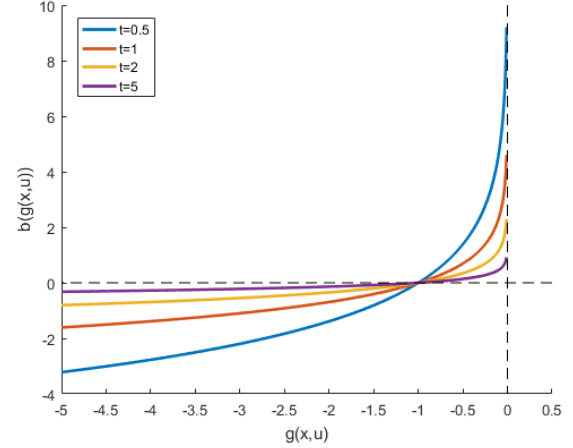

Fig. 1: The logarithmic barrier function

and linearization. However, there are some drawbacks using the exponential barrier function. First, it cannot ensure hard constraints, and the constraints may be violated. Second, it is hard to tune the parameters q_1 and q_2. If they are too small, the constraint violation may be significant; if they are too large, the gradient and Hessian will change sharply near the boundary, which may result in numerical ill conditions.

In this paper, we use a logarithmic barrier function:

$$b(g(x, u)) = -\frac{1}{t} \log(-g(x, u)) \qquad (14)$$

where $t > 0$ is a parameter. The shape of the function under different t is shown in Fig. 1. The logarithmic function has several nice properties. First, it ensures hard constraints by definition. Second, by increasing t, it will asymptotically converge to the indicator function.

Based on the logarithmic barrier function, we built the main structure of the CILQR algorithm. The basic idea is to construct both outer loop and inner loop iterations. The outer loop transforms the inequality constraints into penalties using logarithmic barrier function. The inner loop then solves the transformed unconstrained problem using ILQR. During each outer loop iteration, the parameter t will increase, making the solution converge to the optimal solution of the original constrained problem. The pseudo-code for the CILQR algorithm is shown in Algorithm 1.

The three core ingredients of the algorithm are:
Outer Loop

The algorithm starts with an initial feasible trajectory and an initial parameter $t^{(0)} > 0$. At every iteration, the constraints in (2d) and (2e) are transformed into penalties using (14). They are then added to the objective function (3a) in Problem 3. The transformed problem is then passed to the inner loop to solve. After every iteration, the parameter t will be increased by a scalar $\mu > 1$.

Inner Loop

The inner loop solves the transformed Problem 3 introduced by the outer loop. Since it is now already an unconstrained problem, the ILQR algorithm described in Section IV is utilized

Result: The planned control sequence u and the corresponding trajectory x

Given a strictly feasible initial control sequence u, $t := t^{(0)} > 0$, $\mu > 1$, $0 < \alpha < 1$;

Forward simulation to get initial trajectory $x(u)$;

repeat// *Outer loop*

    Transform the inequality constraints to the cost using logarithmic barrier function $g^k < 0 \rightarrow -\frac{1}{t} \log(-g^k)$;

    repeat// *Inner loop*

        Compute $\bar{u}$ by computing ILQR step;

        repeat// *Line search*

            $\bar{u} := \alpha\bar{u}$;

        until *constraints not violated*;

        Forward simulation to get trajectory $x^*(\bar{u})$;

    until *convergence*;

    $u := \bar{u}$;

    $x := x^*$;

    $t := \mu t$;

    // *Update*

until *convergence*;

Algorithm 1: The Constrained Iterative LQR Algorithm

to solve the problem.

Line Search

During the ILQR computation, line search is commonly used to ensure convergence. In CILQR, line search has an additional function. Although the logarithmic function ensures hard constraints by definition, the constrains may still be violated during ILQR trajectory update. To solve this issue, in the line search iteration, we continue to check if the updated trajectory violates the constraints until the constraints are satisfied.

VI. CONVERGENCE OF THE CONSTRAINED ITERATIVE LQR ALGORITHM

The proposed CILQR algorithm can be categorized as a new numerical optimization algorithm, which can be used to solve general motion planning problems for robotic systems.

When we talk about a numerical optimization algorithm, one important issue is the convergence property. Does the CILQR algorithm converge? The answer is yes; its convergence can be proved and the main result is summarized as follows:

Theorem 1 (Convergence of Algorithm 1). *Under Algorithm 1, the trajectory pair sequence $\{x^{(k)}, u^{(k)}\}$ will converge to a local optimum of Problem 2, e.g., satisfies the following Karush-Kuhn-Tucker (KKT) conditions:*

$$\nabla J + \sum_{i=0}^{N} \mu_i^T \nabla g^i + \sum_{j=0}^{N-1} \lambda_{j+1}^T \nabla(f^j - x_{j+1}) = 0 \quad (Stationarity) \tag{15a}$$

$$\left. \begin{array}{l} g^i < 0, \ for \ i = 0, ..., N \\ f^j = x_{j+1}, \ for \ j = 0, ..., N-1 \end{array} \right\} \quad (Primal \ feasibility) \tag{15b}$$

$$\mu_i \geq 0, \ for \ i = 0, ..., N \quad (Dual \ feasibility) \tag{15c}$$

$$\mu_i g^i = 0, \ for \ i = 0, ..., N \quad (Complementary \ slackness) \tag{15d}$$

where J is the objective function (2a), $\{\mu_i\}$ are the dual variables for inequality constraints, and $\{\lambda_i\}$ are the dual variables for the dynamic system equality constraints. We now show the proof of Theorem 1. Note here we ignore the trivial initial state constraint (2c).

A. Preliminary Results

In order to prove Theorem 1, we need to present some preliminary results that are useful for proving the theorem.

Proposition 1. *Using the barrier function, at each iteration in the outer loop, problem 2 is transformed to the form of problem 3.*

Proof. By eliminating the inequality constraints (2d) and (2e) of Problem 2, transforming them using the logarithmic barrier function (14), and adding them to the objective function (2a), we get the following problem:

$$x^*, u^* = \arg\min_{x,u} \left(\phi(x_N) - \frac{1}{t} \log(-g^N(x_N)) \right)$$
$$+ \sum_{k=0}^{N-1} \left(L^k(x_k, u_k) - \frac{1}{t} \log(-g^k(x_k, u_k)) \right) \tag{16}$$
$$s.t. \quad x_{k+1} = f^k(x_k, u_k), \quad k = 0, 1, ..., N-1$$

Define

$$\bar{\phi}(x_N) = \phi(x_N) - \frac{1}{t} \log(-g^N(x_N)) \tag{17}$$

and

$$\bar{L}^k(x_k, u_k) = L^k(x_k, u_k) - \frac{1}{t} \log(-g^k(x_k, u_k)) \tag{18}$$

we have

$$x^*, u^* = \arg\min_{x,u} \left(\bar{J} = \bar{\phi}(x_N) + \sum_{k=0}^{N-1} \bar{L}^k(x_k, u_k) \right) \tag{19}$$
$$s.t. \quad x_{k+1} = f^k(x_k, u_k), \quad k = 0, 1, ..., N-1$$

which is in the same form of Problem 3. $\qquad \square$

Lemma 2. *The inner loop ILQR algorithm will converge to an optimum that satisfies the KKT conditions of Problem 3.*

Proof. The ILQR algorithm can solve the Problem 3 and converge to a solution which satisfies the Pontryagin's Minimum Principle [23]. The solution satisfies:

$$f^j = x_{j+1}, \ for \ j = 0, ..., N-1 \tag{20a}$$

$$\frac{\partial(L^k + V^{k+1}(f^k))}{\partial u_k} = 0 = \frac{\partial L^k}{\partial u_k} + \left(\frac{\partial f^k}{\partial u_k} \right)^T V_x^{k+1} \tag{20b}$$

$$\frac{\partial(L^k + V^{k+1}(f^k))}{\partial x_k} = \frac{\partial L^k}{\partial x_k} + \left(\frac{\partial f^k}{\partial x_k} \right)^T V_x^{k+1} = V_x^k \tag{20c}$$

$$V^N = \phi(x_N) \tag{20d}$$

Define Hamiltonian

$$H^k = L^k + \lambda_{k+1}^T f^k \tag{21}$$

Then the Lagrangian of the transformed Problem 3 can be written as:

$$\bar{J}' = \bar{\phi} + \sum_{k=0}^{N-1} \left(\bar{L}^k + \lambda_{j+1}^T \left(f^j - x_{j+1} \right) \right)$$
$$= \bar{\phi} - \lambda_N^T x_N + H^0 + \sum_{k=1}^{N-1} \left(H^k - \lambda_k^T x_k \right) \tag{22}$$

Now, write down the variation of the Lagrangian:

$$d\bar{J}' = (\phi_{x_N} - \lambda_N)^T dx_N + \left(H_{x_0}^0 \right)^T dx_0 + \left(H_{u_0}^0 \right)^T du_0$$
$$+ \sum_{k=1}^{N-1} \left[\left(H_{x_k}^k - \lambda_k \right)^T dx_k + \left(H_{u_k}^k \right)^T du_k \right]$$
$$+ \sum_{k=1}^{N} \left(H_{\lambda_k}^{k-1} - x_k \right)^T d\lambda_k \tag{23}$$

By setting all coefficients of (23) to zero we get the corresponding KKT conditions:

$$x_{k+1} = \frac{\partial H^k}{\partial \lambda_{k+1}} = f^k \tag{24a}$$

$$0 = \frac{\partial H^k}{\partial u_k} = \frac{\partial L^k}{\partial u_k} + \left(\frac{\partial f^k}{\partial u_k} \right)^T \lambda_{k+1} \tag{24b}$$

$$\lambda_k = \frac{\partial H^k}{\partial x_k} = \frac{\partial L^k}{\partial u_k} + \left(\frac{\partial f^k}{\partial x_k} \right)^T \lambda_{k+1} \tag{24c}$$

$$\frac{\partial \phi}{\partial x_N} = \lambda_N \tag{24d}$$

Observe that by setting $\lambda_k = V_x^k$, (24a), (24b), (24c) and (24d) become same as (20a), (20b), (20c) and (20d). Therefore, the solution of the inner loop ILQR satisfies the KKT conditions of Problem 3. □

Lemma 3. *The KKT conditions of the transformed Problem 3 is a deformation of the KKT conditions of Problem 2.*

Proof. Recall the stationarity condition of Problem 3:

$$0 = \nabla \left(\bar{\phi} + \sum_{k=0}^{N-1} \bar{L}^k \right) + \sum_{j=0}^{N-1} \lambda_{j+1}^T \nabla \left(f^j - x_{j+1} \right) \tag{25}$$

Now compute the derivatives of the Lagrangian we have:

$$0 = \nabla \left(\bar{\phi} + \sum_{k=0}^{N-1} \bar{L}^k \right) + \sum_{j=0}^{N-1} \lambda_{j+1}^T \nabla \left(f^j - x_{j+1} \right)$$
$$= \nabla \left(\phi - \tfrac{1}{t} \log \left(-g^N \right) \right) + \nabla \sum_{k=0}^{N-1} \left(L^k - \tfrac{1}{t} \log \left(-g^k \right) \right)$$
$$+ \sum_{j=0}^{N-1} \lambda_{j+1}^T \nabla \left(f^j - x_{j+1} \right)$$
$$= \nabla \left(\phi + \sum_{k=0}^{N-1} L^k \right) + \sum_{k=0}^{N} \nabla \left(-\tfrac{1}{t} \log \left(-g^N \right) \right)$$
$$+ \sum_{j=0}^{N-1} \lambda_{j+1}^T \nabla \left(f^j - x_{j+1} \right)$$
$$= \nabla \left(\phi + \sum_{k=0}^{N-1} L^k \right) + \sum_{k=0}^{N} \left(-\tfrac{1}{tg^k} \right) \nabla g^k$$
$$+ \sum_{j=0}^{N-1} \lambda_{j+1}^T \nabla \left(f^j - x_{j+1} \right) \tag{26}$$

Define the dual variables:

$$\mu_k = -\frac{1}{tg^k} \tag{27}$$

where $t > 0$ is a parameter. Then we have:

$$\nabla \left(\phi + \sum_{k=0}^{N-1} L^k \right) + \sum_{k=0}^{N} \mu_k \nabla g^k + \sum_{j=0}^{N-1} \lambda_{j+1}^T \nabla \left(f^j - x_{j+1} \right) = 0 \tag{28a}$$

$$\left. \begin{array}{l} g^i < 0, \ for \ i = 0, ..., N \\ f^j = x_{j+1}, \ for \ j = 0, ..., N-1 \end{array} \right\} \tag{28b}$$

$$\mu_i = -\frac{1}{tg^i} \geq 0, \ for \ i = 0, ..., N \tag{28c}$$

$$\mu_i g^i = -\frac{1}{t}, \ for \ i = 0, ..., N \tag{28d}$$

Note that everything in (28) is same as the KKT conditions (15) of Problem 2 except the complementary slackness condition (28d) and (15d). We call (28) a deformation of the KKT conditions of Problem 2. □

Proposition 4. *The deformed KKT conditions of each outer loop iteration converge to the KKT conditions of problem 2.*

Proof. By increasing t at each outer loop iteration to infinity, we have:

$$t \to \infty \Rightarrow \mu_i g^i = -\frac{1}{t} \to 0, \ for \ i = 0, ..., N \tag{29}$$

which makes (28d) the complementary slackness condition (15d). Therefore (28) become the KKT conditions (15). □

B. Proof of the Main Result

With the preliminary results, the proof of the main results becomes clear:

Proof. The CILQR algorithm starts with a feasible trajectory pair $\left(x^{(0)}, u^{(0)} \right)$, parameter $t^{(0)}$ and scale μ. By Proposition 1, the original Problem 2 is transformed to the form of Problem 3. Then Problem 3 is solved by the inner loop ILQR and gives a trajectory pair $\left(x^{(1)}, u^{(1)} \right)$. By Lemma 2, this solution $\left(x^{(1)}, u^{(1)} \right)$ satisfies the KKT conditions of the Problem 3. Then by Lemma 3, this KKT conditions of Problem 3 are deformation of KKT conditions of the original Problem 2, with a different complementary slackness condition (28d). Now increase t by letting $t^{(1)} = \mu t^{(0)}$ and restart the iteration to obtain a solution $\left(x^{(2)}, u^{(2)} \right)$ which also satisfies deformation of KKT conditions of Problem 3. By induction, we can obtain a sequence of trajectory pairs $\left\{ \left(x^{(i)}, u^{(i)} \right) \right\}$. By Proposition 4, this sequence will converge to a trajectory pair that satisfies the KKT conditions of the original Problem 2. □

VII. APPLICATION ON AUTONOMOUS DRIVING MOTION PLANNING

In this section, the CILQR algorithm is applied to the motion planning problem in autonomous driving. Subsection VII-A formulates the problem in the form of Problem 2. Subsection VII-B introduces the vehicle model we use. Subsection VII-C and VII-D describe the objective function and constraints.
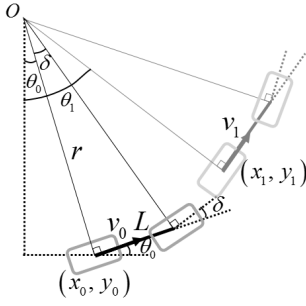
Fig. 2: The vehicle bicycle kinematic model

A. Problem Statement

The autonomous driving motion planning problem can be formulated as follows:

Problem 4 (Motion Planning Problem for Autonomous Driving)

$$x^*, u^* = \arg\min_{x,u} \phi(x_N) + \sum_{k=0}^{N-1} L^k(x_k, u_k) \quad (30a)$$

$$s.t. \quad x_{k+1} = f(x_k, u_k), \quad k = 0, 1, ..., N-1 \quad (30b)$$

$$x_0 = x_{start} \quad (30c)$$

$$d\left(x_k, O_j^k\right) > 0, \quad k = 1, 2, ..., N. \quad j = 1, 2, ..., m \quad (30d)$$

$$\underline{u} < u_k < \overline{u}, \quad k = 1, 2, ..., N-1 \quad (30e)$$

where (30b) is the system dynamic equation. Here we ignore the superscript, because the vehicle dynamics is time invariant. (30d) shows the collision avoidance constraints. There are m obstacles and O_j^k is the space occupied by the jth obstacle at time step k. $d\left(x_k, O_j^k\right)$ is the distance from the host vehicle to the jth obstacle at time step k. $\underline{u}$ and $\overline{u}$ are the lower bound and upper bound of the control inputs.

B. Vehicle Model

The model applied in this paper is the vehicle bicycle kinematic model, which is shown in Fig. 2. The current vehicle state vector includes the 2D position (x_0, y_0), the velocity v_0 and the heading θ_0. The control input vector includes the acceleration a and the steering angle δ. L is the wheel base. Assuming that the steering angle maintains the same in a sampling time T_r, the vehicle will rotate around the instant center O with rotation radius r. The distance the vehicle moves in one sampling time is $l = v_0 T_r + \frac{1}{2} a T_r^2$ and the curvature is $\kappa = \frac{\tan\delta}{L}$. Therefore, the updated vehicle state after one sampling time is:

$$
\begin{aligned}
v_1 &= v_0 + aT_r \\
\theta_1 &= \theta_0 + \int_0^l \kappa ds = \theta_0 + \kappa l \\
x_1 &= x_0 + \int_0^l \cos(\theta_0 + \kappa s) ds = x_0 + \frac{\sin(\theta_0 + \kappa l) - \sin(\theta_0)}{\kappa} \\
y_1 &= y_0 + \int_0^l \sin(\theta_0 + \kappa s) ds = x_0 + \frac{\cos(\theta_0) - \cos(\theta_0 + \kappa l)}{\kappa}
\end{aligned}
\tag{31}
$$

We can now write down the vehicle dynamic equation as:

$$
\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ v_{k+1} \\ \theta_{k+1} \end{bmatrix} = f\left(\begin{bmatrix} x_k \\ y_k \\ v_k \\ \theta_k \end{bmatrix}, \begin{bmatrix} a \\ \kappa \end{bmatrix} \right) \tag{32}
$$

Note that in this subsection, in order to describe the dynamic model more clearly, we use x_k and y_k to represent the 2D position of the vehicle at time step k. However, in previous and the continuing sections, x_k represents the state vector at time step k.

C. Objective Function

The objective function (30a) is decoupled into control costs and state costs. For control costs, there are cost for acceleration and cost for steering angle. For state costs, there are cost for reference tracking and cost for velocity tracking. We now describe how they are computed.

1) acceleration: The term:

$$c_k^{acc} = w_{acc} u_k^T \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} u_k \tag{33}$$

is the cost for longitudinal acceleration at time step k. This cost is related to the fuel consumption and passenger comfort. The weight w_{acc} is a parameter to tune.

2) Steering Angle: The term:

$$c_k^{steer} = w_{steer} u_k^T \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} u_k \tag{34}$$

is the cost for steering angle. It penalizes rapid change in vehicle direction which is related to driving safety and comfort. Tuning the weight w_{steer} can change how much we concern the above issues.

3) Velocity Tracking: The term:

$$c_k^{vel} = w_{vel} \left(\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}^T x_k - v^r \right)^T \left(\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}^T x_k - v^r \right) \tag{35}$$

is the cost for velocity tracking at time step k. v^r is the reference velocity and w_{vel} is a parameter.

4) Reference Tracking: The term c_k^{ref} is the cost to penalize the distance from the ego vehicle to the reference trajectory. A commonly used method to define reference is to set a sequence of reference points $x^r = [x_0^r, x_1^r, ..., x_N^r]$. The cost penalizes the distance from the trajectory point x_k of the ego vehicle to the reference trajectory point x_k^r:

$$c_k^{ref} = (x_k - x_k^r)^T Q_k^r (x_k - x_k^r) \tag{36}$$

However, this method may lead to some problematic scenarios. For example, as shown in Fig. 3, the vehicle is driving on a curve road and it has to stop at the line A. Suppose the reference are points on the road centerline $\{x_k^r\}$. If the cost is (36), then the planned trajectory will be similar to points $\{x_k\}$ in Fig. 3. Observe that in this case, points such as x_i, x_{i+1} and x_{i+2} are significantly deviated from the centerline because they are
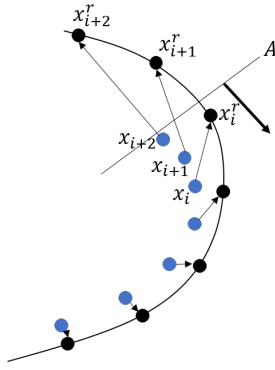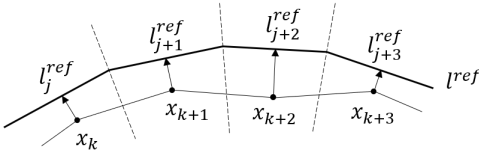
Fig. 3: Problematic Scenario on Curve Road

Fig. 4: Distance to the reference

"pulled" towards their corresponding reference points x_i^r, x_{i+1}^r and x_{i+2}^r.

The ideal cost for reference tracking penalizes the distance from a trajectory point of the ego vehicle to the entire reference trajectory. In this paper, we use polyline l^{ref} to represent the reference trajectory and penalize the distance from the ego vehicle to the polyline. The distance is denoted as $d^{ref}\left(x_k, l^{ref}\right)$. Then the cost for reference tracking is:

$$c_k^{ref} = w_{ref} d^{ref}\left(x_k, l^{ref}\right) \tag{37}$$

The distance is calculated as follows. As shown in Fig. 4, l^{ref} is the polyline reference trajectory, which has m segments. l_j^{ref}, l_{j+1}^{ref}, l_{j+2}^{ref} and l_{j+3}^{ref} are part of them. The 2D space is partitioned by the angular bisectors (denoted as dash lines in Fig. 4) between adjacent line segments. x_k, x_{k+1}, x_{k+2} and x_{k+3} are points on the planned trajectory. For a specific point such as x_k, we first localize its partition and corresponding line segment l_j^{ref}, then we have $d^{ref}\left(x_k, l^{ref}\right) = d\left(x_k, l_j^{ref}\right)$, which is the Euclidean distance between a point and a line.

D. Constraints

Besides the objectives to optimize, there are also some constraints the vehicle should not violate when driving. Three kinds of constraints are considered in this paper: the acceleration constraint, the steering angle constraint and the obstacle avoidance constraint. We now describe the details of how they are calculated.

1) Acceleration Constraint: The acceleration is bounded according to the engine force limit and the braking force limit. This constraint is formulated as:

$$a_{low} \le u_k^T \begin{bmatrix} 1 \\ 0 \end{bmatrix} \le a_{high} \tag{38}$$
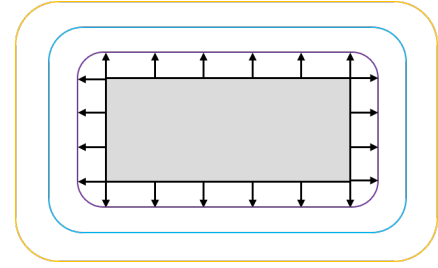

Fig. 5: Distance to a polygon

where $a_{high} > 0$ is the largest acceleration the engine can provide, and $a_{low} < 0$ is the largest deceleration the brake can provide.

2) Steering Angle Constraint: The steering angle is bounded according to the steering angle limit:

$$-\bar{s} \le u_k^T \begin{bmatrix} 0 \\ 1 \end{bmatrix} \le \bar{s} \tag{39}$$

where $\bar{s}$ is the largest steering angle value of the vehicle.

3) Obstacle Avoidance Constraint: The obstacle avoidance constraint includes avoiding the moving obstacles and static obstacles. For example, how to overtake the front vehicle safely, and how to avoid parked vehicles on the roadside. The non-convexity of the constraints makes it difficult to deal with.

In this paper the obstacles are represented by polygons, and the vehicles are represented by rectangles. The core of the collision avoidance constraints (30d) is the distance from ego vehicle to the polygon O_j, which can be calculated as:

$$d\left(x_k, O_j\right) = \min_{y \in O_j} d\left(x_k, y\right), \quad y \notin O_j \tag{40}$$

where $d\left(x_k, y\right)$ is the Euclidean distance between point x_k and point y. The distance (40) makes up a distance field, which is shown in Fig. 5. Note that here we do not consider the situation for $y \in O_j$, because given a feasible initial trajectory, our algorithm can ensure the planned trajectories strictly feasible.

VIII. CASE STUDIES

Three driving cases are used to illustrate the capability of the proposed method. In our simulation environment, the host vehicle is represented by a blue box and the surrounding vehicles are represented by yellow ones. The temporal information is represented by the depth of color, where deeper color represents later time step.

For each case, we draw the historic trajectories of the host vehicle and all surrounding vehicles. Figures of speed profile, longitudinal and lateral acceleration, as well as distance to the closest obstacle are also provided.

The sampling time of the simulation is $T_s = 0.2$ s, and the preview horizon is $N = 40$. Thus the preview time is $T = 8$ s, which is enough for long-term planning. The simulator is written in Matlab script and run on a laptop with 2.6GHz Intel Core i7-6600U.
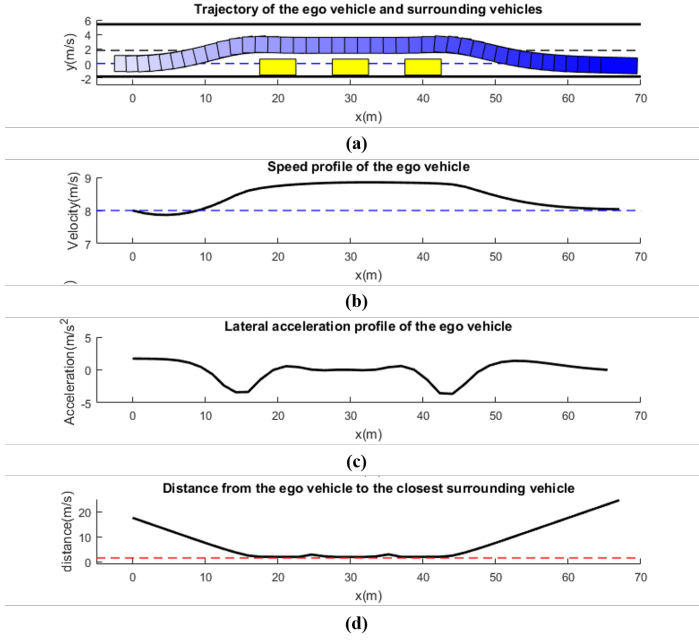
Fig. 6: Static Obstacle Avoidance

Fig. 7: The Lane Change Scenario

A. Case 1: Static Obstacle Avoidance

In this case there are several static obstacles (e.g., street parking vehicles) along the road. The scenario is shown in the first figure of Fig. 6 The three yellow boxes represent the static obstacles. The thick black lines are road boundaries. The black dash line separates the two-way road and the blue dash line represents the reference for the ego vehicle to track. Here the initial velocity of the ego vehicle is 8 m/s.

Fig. 6 (a) shows the trajectory of the ego vehicle. We can see the ego vehicle passes by these obstacles safely. Fig. 6 (b) shows the speed profile of the ego vehicle, and the blue dash line represents the desired speed, which is 8 m/s in this case. The vehicle accelerates while passing the obstacles and slows down to the desired speed after finishing passing. The reason is that the cost function has the penalty term for reference tracking defined in Section VII-C4, so that the ego vehicle tries to reduce the time it deviates from the reference. Fig. 6 (c) shows the lateral acceleration profile of the ego vehicle, which is within a comfortable level. Fig. 6 (d) shows the distance to the closest obstacle, where the red dash line represents the value of safety margin. We can see the distance is always kept above the safety margin. The average runtime in Case 1 is 0.18 s.

B. Case 2: Crowd Lane Changing

Lane changing is one of the most common maneuvers in both urban and highway driving. Usually, there is a front vehicle in the target lane and the host vehicle needs adaptive cruise control (ACC) after finishing the lane changing. However, executing lane changing is dangerous in a crowded environment where there are multiple surrounding vehicles. In such case the driver may ignore the adjacent vehicles. Thus, we create Case 2, where
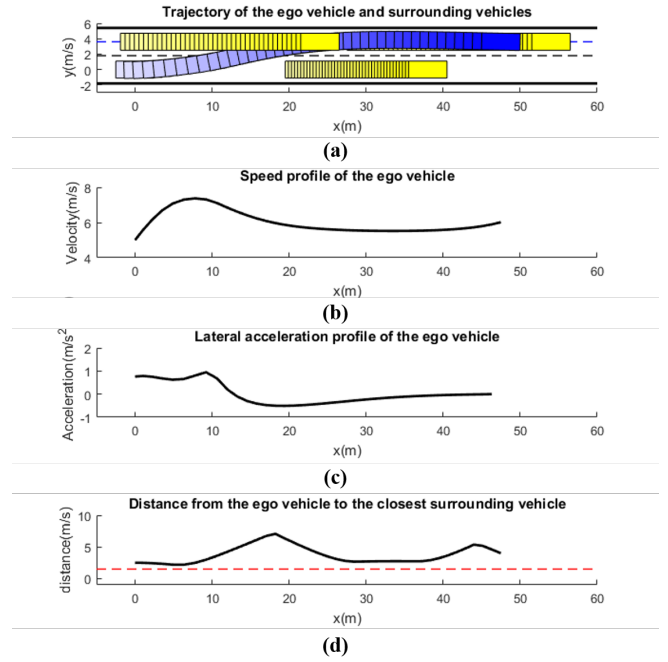
the ego vehicle plans to change to the adjacent lane because there is a slow front vehicle. There are also a vehicle adjacent to the ego vehicle in the target lane, and a vehicle in front of it. The mission for the ego vehicle is to safely change to the target lane and follow the vehicle in front of it. The initial velocity of the ego vehicle is 5 m/s, the velocity of the vehicles in the target lane is 3 m/s and the velocity of the vehicle in the original lane is 2 m/s.

The results are shown in Fig. 7. Fig. 7 (a) shows the trajectories of the vehicles. Fig. 7 (b) shows the speed profile of the ego vehicle, from which we observe that the vehicle accelerates at the beginning of the lane change. The reason is that the gap between front vehicle in the original lane and the adjacent vehicle is shrinking. Hence, it needs to accelerate to finish the lane changing before the gap becomes too small. Then at the end of the lane change, it slows done to follow its front vehicle. Fig. 7 (c) shows the lateral acceleration, and Fig. 7 (d) shows the distance to the surrounding vehicles. The average runtime in this case is 0.14 s.

C. Case 3: Two-way Road Overtaking

One of the most challenging and dangerous driving scenario is to overtake the front vehicle on a two-way road, while there is an oncoming vehicle in the adjacent lane. In this case the driver must be very careful to accelerate and steer appropriately. Otherwise, a fatal accident might happen because of the high relative speed of the vehicles.

There are two cases for this scenario. One is to accelerate to overtake the front vehicle before the oncoming vehicle passes by. We denote this case as "overtaking pass". The other is to decelerate to wait until the oncoming vehicle passes, then
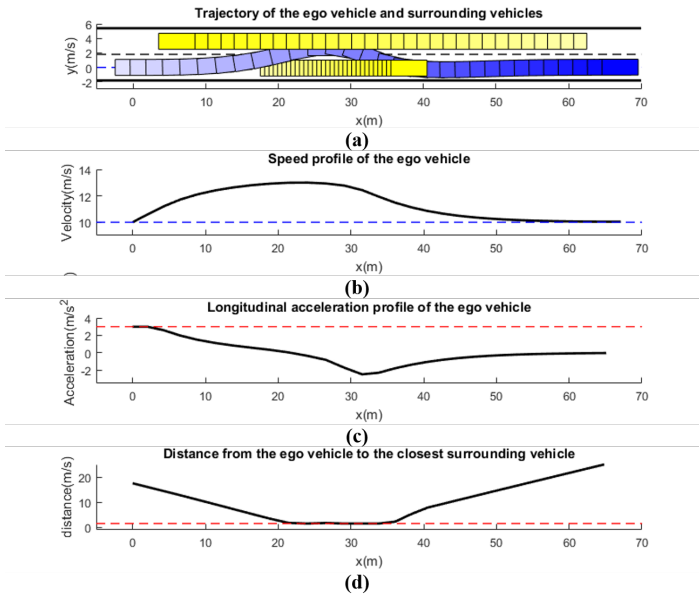
Fig. 8: Overtaking pass

Fig. 9: Overtaking yield

overtake. We denote this case as "overtaking yield". In both cases the initial velocity of the ego vehicle is 10 m/s and the velocity of its front vehicle is 3 m/s. But the velocity for the oncoming vehicle is different in the two cases.

The results for "overtaking pass" are shown in Fig. 8. In this case the velocity of the oncoming vehicle is 9 m/s. Fig. 8 (a) shows the trajectories of vehicles. Fig. 8 (b) shows the speed profile of the ego vehicle. We can observe that the vehicle first accelerates to overtake the front vehicle before the oncoming vehicle arrives. Then after finishing overtaking, it slows down to track the desired speed. Fig. 8 (c) shows the longitudinal acceleration profile, where the red dash line represents the acceleration limit. Fig. 8 (d) shows the distance to surrounding vehicles. The average runtime in this case is 0.16s.

The results for "overtaking yield" are shown in Fig. 8. In this case the velocity of the coming vehicle is 15 m/s. Fig. 8 (a) shows the trajectories of vehicles. Fig. 8 (b) shows the speed profile of the ego vehicle. We can observe that the vehicle first decelerates to wait until the oncoming vehicle passes. Then it accelerates to overtake the front vehicle and reaches its desired speed. Fig. 8 (c) shows the longitudinal acceleration profile, and Fig. 8 (d) shows the distance to surrounding vehicles. The average runtime in this case is 0.22s.

D. Runtime Analysis

To make a comparison, we implemented Case 1 using a standard SQP-based trajectory optimization "optimTraj", which is also written in Matlab script. The problem settings are exactly the same as those in Section VIII-A, including the dynamic equation constraint, inequality constraints, objective function, preview horizon and initialization. The comparison results are shown in Table I. As we can see, the CILQR algorithm is much more computationally efficient than the SQP method in this case. Besides the benefits of utilizing of dynamic
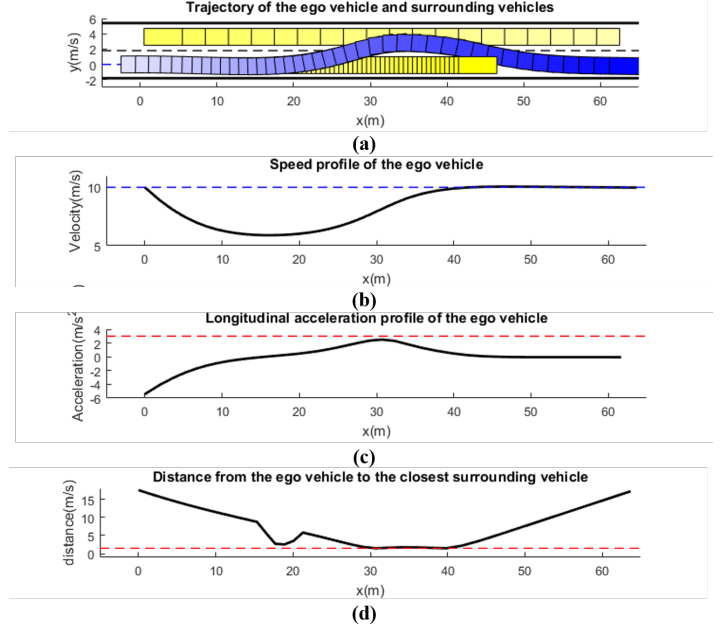
TABLE I: Runtime Comparison with SQP.

Algorithm	Time/Iters (s)	# of Iters	Total Time (s)
SQP	0.3477	43	14.95
CILQR	0.0086	21	0.18

programming, the analytical derivatives and Hessians instead of finite difference are also helpful for reducing the runtime.

IX. CONCLUSION AND DISCUSSION

In this paper, the constrained Iterative LQR algorithm was proposed to efficiently solve the optimal control problem with nonlinear system dynamics and non-convex constraints. The outer-inner loop framework enabled the usage of the efficient ILQR algorithm while guaranteeing convergence to the optimum of the constrained problem. A general on-road driving motion planning problem was then solved by the CILQR algorithm. Case studies showed that our algorithm can work properly in several on-road driving scenarios. The runtime was much faster than the standard SQP solver.

Since CILQR is basically an optimization solver, a initial trajectory must be provided. In the experiments of this paper, the initial trajectories are just some dumb trajectories such as maintaining constant velocity or staying still. The resulting planned trajectory only depends on the behavior pattern, no matter how dumb the initial trajectory is. For example, all initial trajectories that go to the left will finally converge to an optimal left side overtaking trajectory. However, the quality of the initial trajectory can influence the converge speed of the algorithm. Thus method to provide better initial trajectories can be investigated in the future. Furthermore, although this paper concentrates on the problem of autonomous driving motion planning, the general formulation of CILQR makes it possible

to be applied to motion planning of robot manipulators. In that case the contact force needs to be considered, which requires extensions and modifications of the current CILQR algorithm.

REFERENCES

[1] D. Gonzlez, J. Prez, V. Milans, and F. Nashashibi, "A Review of Motion Planning Techniques for Automated Vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 4, pp. 1135–1145, 2016.

[2] B. Paden, M. p, S. Z. Yong, D. Yershov, and E. Frazzoli, "A Survey of Motion Planning and Control Techniques for Self-Driving Urban Vehicles," *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 1, pp. 33–55, 2016.

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

[4] J. Ziegler, P. Bender, T. Dang, and C. Stiller, "Trajectory planning for Bertha - A local, continuous method," in *2014 IEEE Intelligent Vehicles Symposium Proceedings*, 2014, pp. 450–457.

[5] T. Gu, J. M. Dolan, and J.-W. Lee, "Runtime-bounded tunable motion planning for autonomous driving," in *Intelligent Vehicles Symposium (IV), 2016 IEEE*, 2016, pp. 1301–1306.

[6] X. Qian, I. Navarro, A. de La Fortelle, and F. Moutarde, "Motion planning for urban autonomous driving using Bézier curves and MPC," in *Intelligent Transportation Systems (ITSC), 2016 IEEE 19th International Conference on*, 2016, pp. 826–833.

[7] W. Zhan, J. Chen, C. Y. Chan, C. Liu, and M. Tomizuka, "Spatially-partitioned environmental representation and planning architecture for on-road autonomous driving," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, 2017, pp. 632–639.

[8] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Path Planning for Autonomous Vehicles in Unknown Semi-structured Environments," *The International Journal of Robotics Research*, vol. 29, no. 5, pp. 485–501, 2010.

[9] D. Ferguson and A. Stentz, "Field d*: An interpolation-based path planner and replanner," pp. 239–253, 2007.

[10] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 2, 2000, pp. 995–1001 vol.2.

[11] L. Sun, C. Peng, W. Zhan, and M. Tomizuka, "A fast integrated planning and control framework for autonomous driving via imitation learning," *arXiv:1707.02515 [cs]*, 2017.

[12] W. Zhan, J. Li, Y. Hu, and M. Tomizuka, "Safe and feasible motion generation for autonomous driving via constrained policy net," in *Annual Conference on IEEE Industrial Electronics Society (IECON)*, 2017, pp. 4588–4593.

[13] J. T. Betts, *Practical methods for optimal control and estimation using nonlinear programming*. SIAM, 2010.

[14] J. Nocedal and S. J. Wright, *Numerical optimization 2nd*. Springer, 2006.

[15] P. T. Boggs and J. W. Tolle, "Sequential quadratic programming," *Acta Numerica*, vol. 4, pp. 1–51, 1995.

[16] J. Ziegler *et al.*, "Making Bertha Drive #x2014;An Autonomous Journey on a Historic Route," *IEEE Intelligent Transportation Systems Magazine*, vol. 6, pp. 8–20, 2014.

[17] C. Liu, C.-Y. Lin, and M. Tomizuka, "The convex feasible set algorithm for real time optimization in motion planning," in *SIAM J. Control Optim. (2016) in review*. [Online]. Available: arXiv:1709.00627

[18] J. Chen, C. Liu, and M. Tomizuka, "Foad: Fast optimization-based autonomous driving motion planner," in *American Control Conference (ACC)*, 2018.

[19] C. Liu, W. Zhan, and M. Tomizuka, "Speed profile planning in dynamic environments via temporal optimization," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, Jun. 2017.

[20] F. L. Lewis, D. Vrabie, and V. L. Syrmos, *Optimal Control*. John Wiley & Sons, 2012.

[21] D. P. Bertsekas, D. P. Bertsekas, D. P. Bertsekas, and D. P. Bertsekas, *Dynamic programming and optimal control*. Athena scientific Belmont, MA, 1995, vol. 1, no. 2.

[22] D. H. Jacobson, "New second-order and first-order algorithms for determining optimal control: A differential dynamic programming approach," *Journal of Optimization Theory and Applications*, vol. 2, no. 6, pp. 411–440, 1968.

[23] D. H. Jacobson and D. Q. Mayne, "Differential dynamic programming," 1970.

[24] W. Li and E. Todorov, "Iterative linear quadratic regulator design for nonlinear biological movement systems." in *ICINCO (1)*, 2004, pp. 222–229.

[25] E. Todorov and W. Li, "A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems," in *American Control Conference*, 2005, pp. 300–306.

[26] Y. Tassa, N. Mansard, and E. Todorov, "Control-limited differential dynamic programming," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, 2014, pp. 1168–1175.

[27] J. van den Berg, "Iterated LQR smoothing for locally-optimal feedback control of systems with non-linear dynamics and non-quadratic cost," in *American Control Conference (ACC), 2014*, 2014, pp. 1912–1918.

[28] ——, "Extended LQR: Locally-optimal feedback control for systems with non-linear dynamics and non-quadratic cost," in *Robotics Research*. Springer, 2016, pp. 39–56.

[29] J. Chen, W. Zhan, and M. Tomizuka, "Constrained iterative lqr for on-road autonomous driving motion planning," in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, 2017.
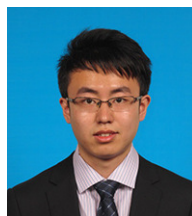
Jianyu Chen received his B.E. degree from Department of Mechanical Engineering in Tsinghua University, Beijing, China, in 2015. He is currently a Ph.D. student in Department of Mechanical Engineering at University of California, Berkeley, CA, USA. His research interests are decision-making, motion planning and behavior modeling for robotic systems, especially for autonomous driving.

Wei Zhan received his B.E. and M.S. degree from Control Science and Engineering Department in Harbin Institute of Technology, China, in 2012 and 2014, respectively. He was a Research Assistant in Department of Mechanical Engineering in The University of Hong Kong, from Feb. 2012 to Jun. 2012. He is currently a Ph.D. Candidate in Mechanical Engineering Department at University of California, Berkeley. His research interests are decision-making, motion planning, motion prediction and behavior analysis for autonomous driving.

Masayoshi Tomizuka (M'86-SM'95-F'97-LF'17) received his Ph. D. degree in Mechanical Engineering from MIT in February 1974. In 1974, he joined the faculty of the Department of Mechanical Engineering at the University of California at Berkeley, where he currently holds the Cheryl and John Neerhout, Jr., Distinguished Professorship Chair. His current research interests are optimal and adaptive control, digital control, signal processing, motion control, and control problems related to robotics, precision motion control and vehicles. He served as Program Director of the Dynamic Systems and Control Program of the Civil and Mechanical Systems Division of NSF (2002- 2004). He served as Technical Editor of the ASME Journal of Dynamic Systems, Measurement and Control, J-DSMC (1988-93), and Editor-in-Chief of the IEEE/ASME Transactions on Mechatronics (1997-99). Prof. Tomizuka is a Fellow of the ASME, IEEE and IFAC. He is the recipient of the Charles Russ Richards Memorial Award (ASME, 1997), the Rufus Oldenburger Medal (ASME, 2002) and the John R. Ragazzini Award (2006).