Model Predictive Control for Autonomous Driving Path-following Control Design

Kyle Guan

***Abstract*—Using a simple kinematic vehicle model, we show how model predictive control (MPC) framework can be applied to the automatic control of steering, acceleration, and brake, thus enabling path-following for a self-driving vehicle.**

I. Introduction

Self-driving vehicles are essentially autonomous decision making platforms that are equipped with plethora of sensors and computational resources. The observations from sensors (e.g., camera, radar, LIDAR, GPS/INS, and odometry), coupled with prior information about the road, traffic, and vehicle dynamics, are used to provide an estimate of the state of the vehicle and its surrounding environment. These estimates are then fed into a decision making system, which automatically outputs control commands (e.g., steering and acceleration) to accomplish the driving task. For efficiently streamlining the decision process, the decision making system of a typical self-driving vehicle is often partitioned into a hierarchical architectures, with each layer dedicating to a set of specific tasks [1]. Figure 1 shows the lowest three layers of the decision making hierarchy, which are the subject of this paper. The top of these three layers is the environment prediction and behavior planning module, which takes the input of route selection (provided by modules in the higher layer, not shown in Fig. 1) and environmental information via sensor fusion (e.g., perceived behavior of other vehicles, road conditions, and traffic signals) and outputs an appropriate driving behavior or a motion specification, such as turn-right or change-lane. The layer below is path planning, which is responsible for translating the driving behavior into a reference path or trajectory that is feasible, collision-free, and comfortable to the passengers. The bottom layer is the local control layer, which takes the reference path and the state of the vehicle as inputs and selects the appropriate actuator outputs to follow the reference path and to correct tracking errors based on the feedbacks.

In this paper, we focus on the local feedback control layer. Specifically, we show how the model predictive control (MPC) framework can be used for a self-driving vehicle to automatically track the reference trajectory provided by the path planning layer. MPC is well suited for solving the multiple-input and multiple-output control problems that involve complex state, input, and systems constraints, such as the path-following problem. The main concept of MPC is to use a *model* to predict the future evolution of the system (a self driving vehicle in our case). At each time step, an open-loop control optimization problem of finite time horizon is solved. The optimal control output is applied to the system only to the following time interval. At this next time step, the optimal control problem based on new measurements of the state is solved and the output is applied again (to generate the new initial state). The process continues over the duration of the control task. In the following, we illustrate the implementation of MPC based control by using a kinematic bicycle model and formulating a nonlinear optimization problem that minimizes the quadratic tracking errors accumulated over a finite time horizon, subject to system operating constraints and initial state input. Using numerical analysis and simulation, we shown that MPC controller provides satisfactory control performances.

The rest of the paper is organized as following. In Section II, we present the vehicle model. In Section III, we first review the frameworks of dynamic optimization and MPC. We then provide a problem formulation for the path-following control. In Section IV, we provide both numerical and simulation results to illustrate the effectiveness of MPC framework.

II. Vehicle Model

For control and planning algorithms, vehicle models [1] are developed, often with a balance between fidelity and complexity, to approximate a vehicle's behavior in response to control actions. A high-fidelity model may realistically and accurately reflect the response of the vehicle, but the details could increase the computational complexity for the planning and control algorithms. Since the focus of this work is to illustrate how MPC is applied to the vehicle path-following, we use kinematic bicycle model, which is often considered as the simplest model for practical use [1].

A. Kinematic Bicycle Model

Kinematic models are simplified models that ignore detailed features, such as tire forces, gravity, and vehicle mass, etc. The bicycle model assumes that two front (rear) wheels act together, thus can be represented by one single wheel. The front and rear wheels are connected by a rigid link and the movement is restricted in $X-Y$ plane, as illustrated in Fig. 2. It is assumed that both wheels can rotate freely about their axes of rotation. In addition, the front wheel can rotate about an axis that is perpendicular to the plane of the movement, to model the effect of steering. For both front and rear wheels, the effects of slipping at the point of contact with the ground are ignored. We denote by x and y the global map coordinates

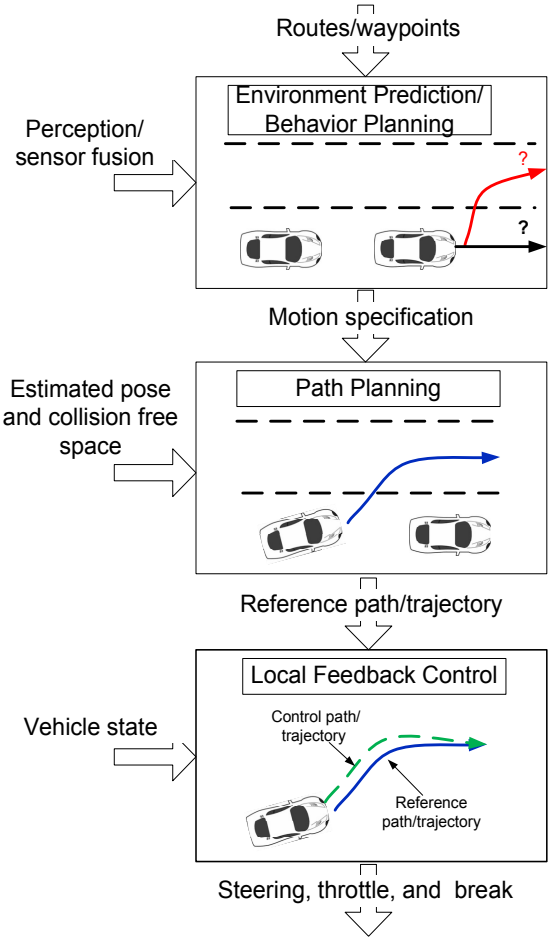

Fig. 1. The lowest three layers of decision making hierarchy.

of the center of the mass of the vehicle, v the velocity, ψ the heading, and L_f the distance between the front wheel and the center of mass. With reference to the bicycle model, the vehicle control inputs are the front wheel[1] steering angle δ and the acceleration of the center of mass a (in the same direction as v). Note that a positive acceleration and a negative acceleration indicate accelerator and brake, respectively. The movement of the vehicle is governed by the following set of nonlinear continuous time equations:

$$\dot{x} = v\cos(\psi) \quad (1)$$
$$\dot{y} = v\sin(\psi) \quad (2)$$
$$\dot{\psi} = \frac{v}{L}\delta \quad (3)$$
$$\dot{v} = a \quad (4)$$

[1]There is no steering for the rear wheel.

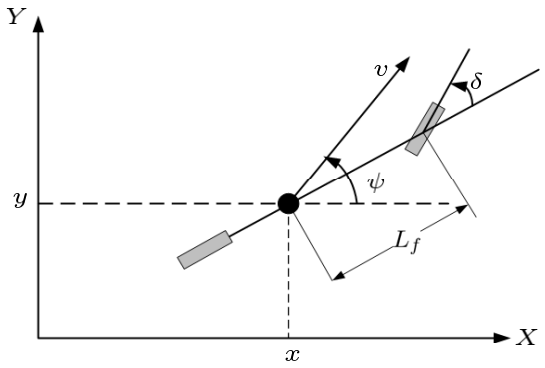

Fig. 2. Kinematic bicycle model.

For control tasks, we discretize the system dynamics with an interval dt:

$$x_{t+1} = x_t + v_t\cos(\psi_t)dt \quad (5)$$
$$y_{t+1} = y_t + v_t\sin(\psi_t)dt \quad (6)$$
$$\psi_{t+1} = \psi_t + \frac{v_t}{L_f}\delta_t dt \quad (7)$$
$$v_{t+1} = v_t + a_t dt \quad (8)$$

B. Errors

For path-following, we consider two types of errors: cross track error and orientation error. The cross track error, denoted as cte, is the Y-distance between the reference position and the current vehicle position. Assuming that the current coordinates of the vehicle is (x_t, y_t) and the the reference path is specified by a polynomial function $f(x)$, the cross track error is $f(x_t) - y_t$. The orientation error, denoted as $e\psi$, is the difference between the referenced orientation and the current orientation. The referenced orientation ψ_{ref} can be obtained by taking the first derivative of $f(x)$ at x_t, i.e., $\psi_{\text{ref}} = \tan^{-1}(f'(x_t))$. The cross track error evolves according to the following discrete time equations:

$$cte_{t+1} = cte_t + v_t\sin(e\psi_t)dt \quad (9)$$
$$= f(x_t) - y_t + v_t\sin(e\psi_t)dt \quad (10)$$

The orientation error evolves according to the following discrete time equations:

$$e\psi_{t+1} = e\psi_t + \frac{v_t}{L_f}\delta_t dt \quad (11)$$
$$= \tan^{-1}(f^{'}(x_t)) - \psi_t + \frac{v_t}{L_f}\delta_t dt \quad (12)$$

In the following, we use a vector $\mathbf{s}_t = [x_t, y_t, \psi_t, v_t, cte_t, e\psi_t]$ and a vector $\mathbf{u}_t = [\delta_t, a_t]$ to represent the state and control input at time step t, respectively.

III. Model Predictive Controller

A. Dynamic optimization and MPC

Dynamic optimization has become a standard tool for decision making in a wide range of areas. Center to these dynamic optimization problems is a (general) dynamic system

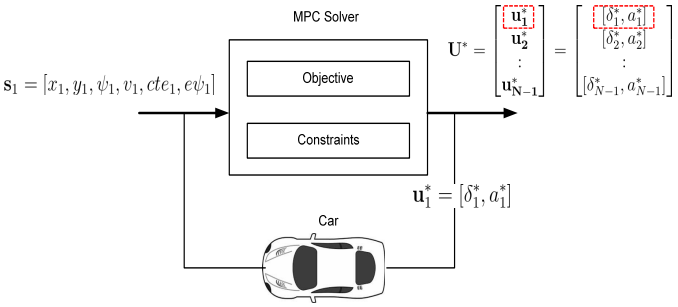

Fig. 3. The main concept of MPC.

model $g()$ that describes the evolution of the state $\mathbf{s}_t$ with time, starting from the initial state $\mathbf{s}_1$, as it is impacted by the control input $\mathbf{u}_t$:

$$\mathbf{s}_{t+1} = g(\mathbf{s}_t, \mathbf{u}_t), \tag{13}$$

with $\mathbf{s}_1$ given. Note that bicycle model equations (5)-(12) provide an example for the general $g(\mathbf{s}, \mathbf{u})$ function.

The goal of the dynamic optimization procedure is to find the vector of control inputs $\mathbf{U} = [\mathbf{u}_1, ..., \mathbf{u}_{N-1}]$, such that the objective function is optimized over some time horizon N:

$$\min_{\mathbf{U}} \sum_{t=1}^{N} q(\mathbf{s}_t, \mathbf{u}_t) + p(\mathbf{s}_N) \tag{14}$$

The terms $q(\mathbf{s}, \mathbf{u})$ and $p(\mathbf{s})$ represent the per stage cost and terminal cost, respectively [5]. In practice, there are various algorithms exploiting the structure of the particular problem, e.g., linearity and convexity, so that even large problems described by complex models and involving many control variables can be solved efficiently and reliably.

The sequence of $\mathbf{U}^* = [\mathbf{u}_1^*, ..., \mathbf{u}_{N-1}^*]$ obtained from the optimization cannot be simply applied (in sequence over the next $N-1$ time steps). The system model that predicts its evolution could be inaccurate. In addition, the system may be affected by external disturbances, causing the actual path to deviate significantly from the predicted one. Instead, it is a common practice to apply only the control solution for the current step $\mathbf{u}_1^*$. After some finite interval, say one time step, we measure the (updated) state and use it as the new initial condition to solve again the same dynamic optimization problem (with a shifted N time horizon). Upon obtaining the new $\mathbf{U}^*$, only $\mathbf{u}_1^*$ is applied to the system and then the state is measured again at the next time step. The process continues until the control task is accomplished. This procedure forms the essence of MPC. The effectiveness of MPC lies in the fact that the feedback of the measurement information feeding to the optimization enables a robustness which is typical for closed-loop systems. In addition, the resulting operating strategy takes into accounts of all the system and problem details, including interactions and constraints, which could be very difficult to accomplish with other control frameworks.

B. MPC Optimization Problem Formulation

Specific to the car path-following problem, with initial state $\mathbf{s}_1 = [x_1, y_1, \psi_1, v_1, cte_1, e\psi_1]$ MPC solves the following optimization problem with N time steps and obtains the solutions of control inputs $[\delta_1^*, a_1^*]$, ..., $[\delta_{N-1}^*, a_{N-1}^*]$, as shown in Fig. 3. Using the bicycle model, we can solve the states $\mathbf{s}_2$, ..., $\mathbf{s}_N$, as illustrated by the dashed green line in Fig. 4 (a). Only the control input $[\delta_1^*, a_1^*]$ is used to drive the vehicle to state $\mathbf{s}_2$ at the next time step, as shown in Fig. 3 and Fig. 4 (b). The updated state $\mathbf{s}_2$ becomes the new initial state to be fed into the optimization problem. The process continues until the driving task is accomplished. Let $\mathbf{S} = [\mathbf{s}_1, ..., \mathbf{s}_N]$ and $\mathbf{U} = [\mathbf{u}_1, ..., \mathbf{u}_{N-1}] = [[\delta_1, a_1], ..., [\delta_{N-1}, a_{N-1}]]$. The formulation is provided as following:

$$\min_{\mathbf{S},\mathbf{U}}: \quad \sum_{t=1}^{N} [\alpha_1 cte_t^2 + \alpha_2 e\psi_t^2 + \alpha_3 (v_t - v_{ref})^2] \tag{15}$$

$$+ \sum_{t=1}^{N-1} (\beta_1 a_t^2 + \beta_2 \delta_t^2) \tag{16}$$

$$+ \sum_{t=1}^{N-2} [\gamma_1 (a_{t+1} - a_t)^2 + \gamma_2 (\delta_{t+1} - \delta_t)^2] \tag{17}$$

$$\text{s.t.}: \quad x_{t+1} = x_t + v_t \cos(\psi_t) dt \tag{18}$$

$$y_{t+1} = y_t + v_t \sin(\psi_t) dt \tag{19}$$

$$\psi_{t+1} = \psi_t + \frac{v_t}{L_f} \delta_t dt \tag{20}$$

$$v_{t+1} = v_t + a dt \tag{21}$$

$$cte_{t+1} = f(x_t) - y_t + v_t \sin(e\psi_t) dt \tag{22}$$

$$e\psi_{t+1} = \tan^{-1}(f^{'}(x_t)) - \psi_t + \frac{v_t}{L_f} \delta_t dt \tag{23}$$

$$\delta_{\min} \leq \delta_t \leq \delta_{\max} \tag{24}$$

$$a_{\min} \leq a_t \leq a_{\max} \tag{25}$$

$$t = 1, \ ..., \ N-1$$

$$\mathbf{s}_1 = [x_1, y_1, \psi_1, v_1, cte_1, e\psi_1] \text{ is given.}$$

The cost (objective) function consists of three parts, which account for the cost associated with reference state, use of actuators, and the gap between sequential actuations. Terms in Eq. (15) represent the sum of squared cross track error, orientation, and the gap between current velocity and referenced velocity (denoted as v_{ref}). The two terms in Eq. (16) penalize large acceleration and steering values, respectively. The two terms in Eq. (17) penalize the large value gaps between sequential acceleration and steering angles, respectively. The coefficients α_1, α_2, α_3, β_1, β_2, γ_1, and γ_2 are weights associated with each term, and can be treated as parameters to be fine-tuned. In the constraints, Eq. (18) - (23) describe the state evolutions according to the bicycle model. Equations (24) and (25) ensure that steering and acceleration at each time step falls into the allowed range.

IV. Results and Discussions

In this section, we present the simulation results of MPC trajectory control for different scenarios, with a focus of

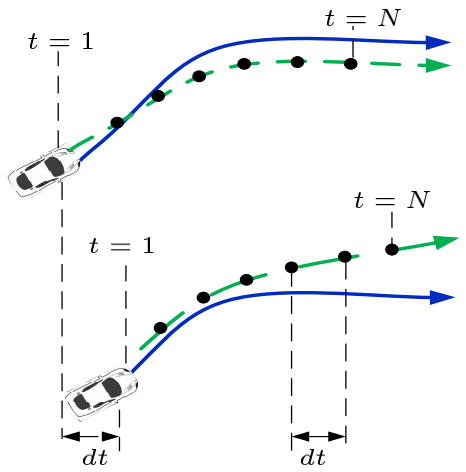

Fig. 4. MPC for vehicle trajectory control. The reference trajectory and the control trajectory are in solid blue and dashed green, respectively.

evaluating the impact of design parameters on the control performances. The MPC algorithms are implemented in C++. The open source software package Ipopt (Interior Point OPTimizer)[6] is used to solve the nonlinear programming problem formulated in the previous section.

A. Track a straight path

We first consider tracking a straight path to illustrate the effectiveness of MPC. The reference line is $y = -1$. Let $\mathbf{s}_1 = [x_1 = 0, y_1 = 10, v_1 = 10, \psi_1 = 0, cte_1 = -11, e\psi_1 = 0]$, $L_f = 2.67$, $v_{ref} = 15$, $N = 25$, $dt = 0.05$, $\delta_{\max} = 25°$, $\delta_{\min} = -25°$, $a_{\max} = 1$, and $a_{\min} = -1$. In the cost function, we first set all the weights α_1, α_2, α_3, β_1, β_2, γ_1, and γ_2 to one. Fig. 5 (a)-(f) show the results of trajectory, cte, $e\psi$, δ, a, and v, respectively. Figure 5 (b) and (c) show that around time step 48, both cte and $e\psi$ start approach zero – the vehicle successfully tracks the reference line $y = -1$. We also note in Fig.5 (e) that the acceleration starts at the maximally allowed value $a_{\max} = 1$ and gradually decreases to zero, as the velocity approaches the the reference value (Fig.5 (f)). Figure 5 (d) shows that the steering angle starts out at a minimal value -25 degrees and then rises to 25 degrees almost instantly, producing a "spike". The fall from 25 degrees to 0 degrees is more gradual but still sudden. To produce a smoother steering transition, we increase the value of the weight γ_2 from one to 500 to penalize "spiky" transition between successive steering angles. The new results (with the values of other parameters unchanged) are shown in Fig. 6. As shown in Fig. 6 (d), the steering transition becomes much smoother. We also note that as a result of smoother steering control, the maximal $|e\psi|$ is reduced from around $80°$ to $60°$, as shown in Fig. 5 (c) and Fig. 6 (c). In addition, the vehicle tracks the straight line around time step 60 (cf. Fig. 6 (b) and (c)) – 10 time steps later than the previous case where γ_2 is one (cf. Fig. 5 (b) and (c)).

B. Track with simulator

Using the simulator provided by Udacity [7], we implement MPC controller in C++ to drive the vehicle along the track. At each iteration, the simulator provides reference points (waypoints) (x_w, y_w) and the car position (x, y), heading ψ, and speed v, all in reference to the global map coordinates. In the simulator, the waypoints are extrapolated into a reference trajectory (shown by the yellow line in Fig. 7). The MPC controller takes these values as inputs, solves the nonlinear optimization problem (of N time horizon) as formulated in Eq. (15)-(25), and outputs a sequence of $N-1$ steering angles and acceleration values, which give rise to a control trajectory (as shown by the green line Fig. 7.) Specific to the simulator, both reference and control trajectory projections are displayed in reference to the vehicle's coordinate system, in which X axis always points to the heading of the car and the Y axis points to the left of the car. Thus, we need to transform the waypoints from global coordinate (x_w, y_w) into car (local) coordinate (x_c, y_c): coordinate:

$$x_c = (x_w - x)\cos\psi + (y_w - y)\sin\psi, \quad (26)$$

$$y_c = -(x_w - x)\sin\psi + (y_w - y)\cos\psi. \quad (27)$$

In car coordinate, the position, heading, and velocity are simply $(0, 0)$, zero, and v, respectively.

The reference speed is set at 45mph. To ensure stable control outputs and smooth trajectories, the values of N and dt need to be carefully selected. The product of N and dt determines the duration over which future predictions are made. In addition, N determines the number of variables to be optimized by MPC. A large N not only increases the computational complexity of the solver, but also reduces the quality of optimization solution. The time step interval dt determines the time elapse between actuations. A larger value of dt results in less frequent actuations and could make it difficult to accurately approximate a continuous reference trajectory. For our implementation, we find that $N = 10$ and $dt = 0.05$ second can strike a good balance between solving time and solution quality. The simulator also introduces a latency of 100 millisecond for an actuation to take effect. To account for this latency, the state at 100 millisecond into the future is fed into the solver as $\mathbf{s_1}$. After selecting appropriate values for N and dt and taking account of the latency, the car successfully tracks the waypoints with a reference speed of 45mph in the simulation. The video of the simulation can be found at https://youtu.be/Qpo5SFfWp5k.

V. Conclusions

We designed an MPC controller to accomplish the path-following task in autonomous driving. With carefully selected design parameters, the controller provided satisfactory control performances under various scenarios.

References

[1] B. Paden, M. Cap, S. Z. Yong, D. Yershov, and E. Frazzle, "A survey of motion planning and control techniques for self-driving urban vehicles," arXiv:1604.07446, last accessed July 2017.

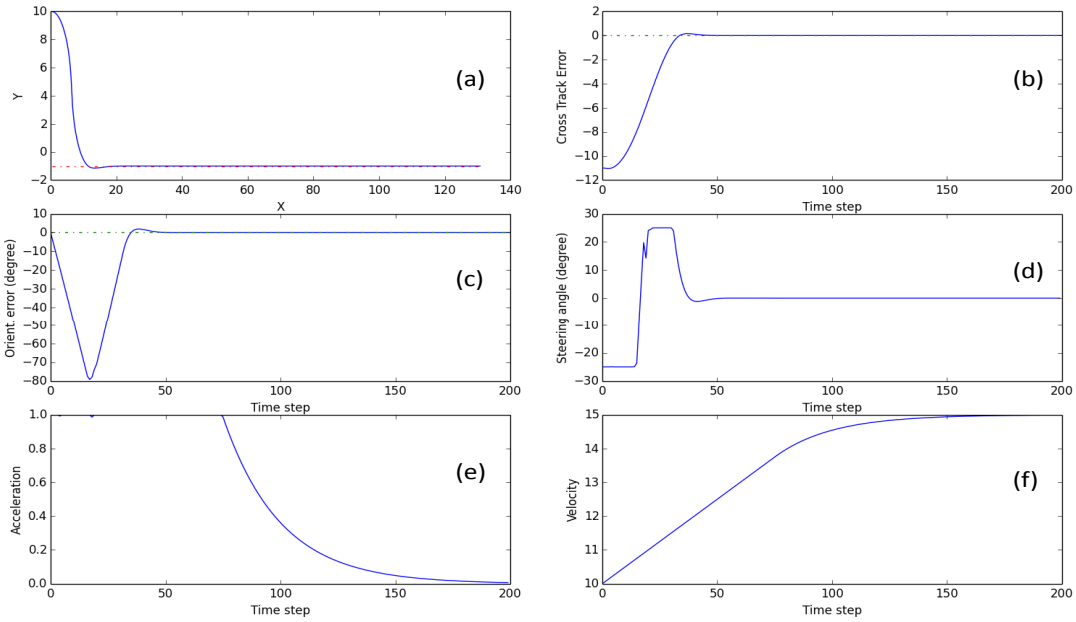

Fig. 5. Subplots (a) -(f) show the respective results of trajectory, cte, $e\psi$, δ, a, and v, with $alpha_1$, α_2, α_3, β_1, β_2, γ_1, and γ_2 all set to one.

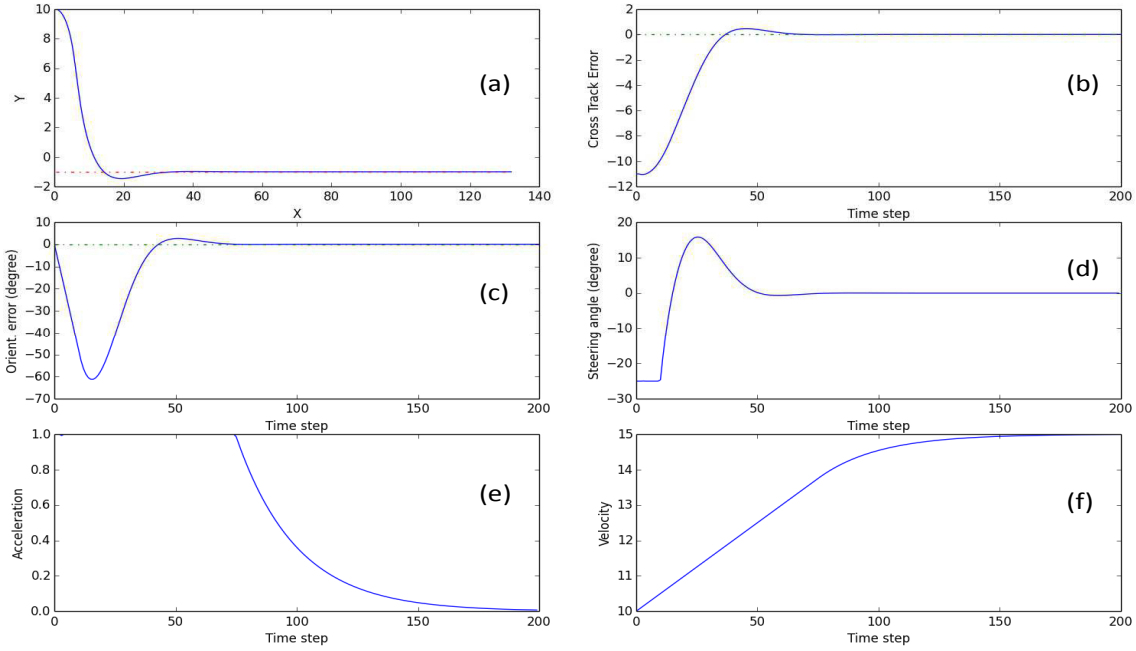

Fig. 6. Subplots (a) -(f) show the respective results of trajectory, cte, $e\psi$, δ, a, and v, with $alpha_1$, α_2, α_3, β_1, β_2, and γ_1, all set to one and γ_2=500.

Fig. 7. Screen shot of the simulator. The yellow line is the reference trajectory; the green line is the control trajectory.

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

[3] P. Falcone, F. Borelli, J. Asgari, H. E. Tsung, and D. Provat, "Predictive active steering control for autonomous vehicle systems," *IEEE Transactions of Control Systems Technology*, vol. 15, no. 3, pp. 566-580, Jan. 2007.

[4] J. Kong, M. Pfeiffer, G. Schildbach, and F. Borelli, "Kinematic and dynamic vehicle models for autonomous driving control," *IEEE Intelligent Vehicles Symposium*, pp. 1094 -1099, June 2015.

[5] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, vol. 1, 2nd Edition, Belmont, MA: Athena Scientific, 1995.

[6] Ipopt (Interior Point OPTimizer) software package, https://projects.coin-or.org/Ipopt, last accessed July 2017.

[7] Udacity Self-driving Car Nano-degree (SDCND) Term 2 Simulator v.1.45, https://github.com/udacity/self-driving-car-sim/releases, last accessed July 2017.