Inverse Reinforcement Learning with Model Predictive Control

Jinxin Zhao
Baidu Research Institute
Sunnyvale, CA
`jinxinzhao@baidu.com`

Liangjun Zhang
Baidu Research Institute
Sunnyvale, CA
`liangjunzhang@baidu.com`

Abstract

Model-free learning based methods for planning and control application have been proven promising by many existing results. However, such kind of planning and control algorithms are rarely used in practical systems, due to the unpredictable outputs. On the other hand, model-based methods are largely deployed in real-life systems, with guarantee of operational safety but complaint of lacking close-to-human behavior. In this article we make an effort to leverage the benefits of model-based optimization method and model-free learning method by introducing a novel scheme of inverse reinforcement learning. We then present a framework for combining human behavior model with model predictive control. The idea is to take advantage of the feature identification capability of neural network to determine the reward function of model predictive control. Furthermore, the proposed approach is implemented to solve the practical autonomous driving longitudinal control problem. In such problem, safe execution and passenger comfort are simultaneously preferred.

1 Introduction

Selecting the most fruitful option by foreseeing the future, describes the nature of model predictive control (MPC). At each time instant, MPC aims to find the control input by solving an optimization problem. This optimization considers the costs/rewards of the future steps and the prediction is achieved by exploiting a state-space model [Mayne et al., 2000]. MPC has drawn enormous attention from both industrial and research perspectives. Industrial applications of MPC started from the field of chemical process control. In decades, many companies like Shell, Honeywell have been developing MPC packages for industrial uses [Qin and Badgwell, 2003]. Nowadays successful implementations of MPC, in addition, lie in the area of power electronics with various topics including active front end (AFE), power converters connected to resistor-inductor (RL) loads [Vazquez et al., 2014]. Apart from industrial applications, MPC has also been an on-going topic in large amount of research projects. A real-time MPC scheme is described in [Erez et al., 2013] for the control of humanoid robots, where MPC generates the trajectories for full body dynamics based on the received sub-tasks. Aerial and ground mobile robotic researchers also implement MPC algorithms to resist the dynamic environment and confront the system constrains [Shim et al., 2003, Künhe et al., 2005]. Moreover, with the surge of self-driving vehicle development in present-day, MPC is showing even more significance in such fields. [Schmied et al., 2015, Borrelli et al., 2006] describe longitudinal control approaches for adaptive cruise control and vehicle tractor control to improve the vehicle emission and fuel-consumption efficiency. Meanwhile active steering lateral vehicle controls focus on collision avoidance, which is made possible by MPC implementation with reasonable dynamic model for prediction [Borrelli et al., 2005, Falcone et al., 2008, Frasch et al., 2013].

Machine Learning for Autonomous Driving Workshop at the 33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

Despite all the advantages and successful applications of MPC, it still suffers certain drawbacks like difficulty of choosing parameters and lack of adaptivity. During the implementation phase of MPC, selection of the parameters such as prediction horizon and optimization gains require lots of trial and error in either simulation or hardware in the loop (HIL) test. In addition, previously fine-tuned MPC controller in most cases does not adapt to the changes of the system. For example, variation of the plant (under-controlled) system parameters results in the state prediction model failing to provide meaningful prediction. Also, the change of the reference trajectory pattern, which may cause the predefined cost function not able to conclude a satisfactory tracking performance anymore. Moreover, in the field of autonomous driving, another issue of model-based algorithm like MPC is that the generated behaviors do not always align with the expectation of a human, resulting into certain level of discomfort. The root cause is that the cost/reward functions are predefined and lack of variation.

2 Related Work

One option to overcome such issues is to exploit finite state machine (FSM) with MPC to update the prediction model or cost function parameters as explained in [Rodriguez et al., 2012, Arahal et al., 2009]. However, such simple approach of combining several sets of parameters, does not avoid tremendous amount of labor of parameters tuning and still requires delicately manual design of states and transitions. On the other hand, deep learning techniques have shown enormous potential in the fields such as object detection, prediction and classification [LeCun et al., 2015, Schmidhuber, 2015]. An active research topic of utilizing capability of deep learning techniques for building self-driving vehicles is end-to-end driving as discussed in [Pomerleau, 1989, Xu et al., 2017], while such methods suffer oscillating control input signals and are not friendly for real-life implementations.

On the other hand, utilizing the identification capacity of data driven method to complete the model-based optimization algorithm provides another viable perspective. Similar idea was investigated within the topic of inverse reinforcement learning (IRL) [Abbeel and Ng, 2004, Ng et al., 2000], which aims to recover the cost function by observing a desired system trajectory or policy. Whereas most problems formulated in the contexts of IRL are finite-state Markov decision process (MDP) [Zhifei and Joo, 2012] with Bayesian probability state transition model, MPC algorithm in most case deals with continuous state space systems. An interesting result is presented in [Silver et al., 2010], where the behavior model, that translates state features (observation) to cost functions, was learnt from demonstration.

In this article, we propose a framework for combining neural network (NN) model with MPC algorithm, where the NN is pre-trained to recover the cost function for MPC, with respect to the observation of the environment. One major challenge of such method is the under-determination, where the observation does not provide enough bijective mapping from observation to label. Another major challenge is that a behavior model with zero parameters also satisfies the optimality condition with observation data, but this case has to be avoided in order to provide meaningful cost/reward function. Autonomous vehicle longitudinal control problem could largely benefit from the propose framework by learning the human-intended behavior. We thus provide a demonstration of solving such problem with the proposed method.

3 Inverse Reinforcement Learning of MPC

3.1 Problem Formulation

Here we formulate the problem of inverse reinforcement learning for a system with MPC framework. Model predictive control generates the control input for the plant (under-controlled) system by solving an optimization problem. This process is repeated at each time instance and the first element of the obtained control sequence is provided to the system. Compared with the classic feedback control law, the main advantages of MPC is bi-fold. One is that the MPC control input is searched from more general space instead of just the linear combination of the state errors in classical control, another one the is that the future system states and references are also considered rather that only considering the current state and the current reference signal.

A MPC algorithm is commonly formulated as follows, providing the current state is x_k and prediction horizon is N,

$$\begin{aligned} \text{minimize}_{u_{k:k+N-1}} \quad & \sum_{i=k}^{k+N-1} C_\theta(x_i, u_i) + F(x_{k+N}) \\ \text{subject to} \quad & x_{i+1} = f(x_i, u_i) \\ & \underline{u} \leq u_i \leq \bar{u} \end{aligned} \tag{1}$$

where $x_i \in \mathbb{R}^n$, $u_i \in \mathbb{R}^m$; C_θ is a cost function and θ represents the function parameters; F represents a final cost function.

In this article, we propose a novel framework of making use of neural network to predict the appropriate cost function for the MPC algorithm, i.e.,

$$C_\theta = g_\psi(y_k), \tag{2}$$

where y_k is the observed information at step k.

As mentioned above, the benefit of such architect is that the neural network model is capable of provide suitable and various cost functions according to the observation of the environment. However, it also introduces the difficulty for the training process. The objective is to train this neural network model with favorable recorded trajectories, so that the entire algorithm is capable of generating desired behavior while guaranteeing optimality in later use. The problem is formulated as follows.

Problem 1 *Given the MPC algorithm described in* (1) *and the neural network structure in* (2)*, design the training process with pre-recorded desired trajectories. So that the cost function can be reconstructed and the output of the algorithm shares similar behavior with compared with the recorded trajectory .*

3.2 Approach

In a normal imitation learning problem, the recorded data can explicitly provide the corresponding relation between the input/observation and output/label of a model. However, this is not the case for Problem 1, since the direct outputs of the MPC cost function C_θ can not be explicitly known or recorded.

We provide the solution to Problem 1 here by presenting how to enable model training process through recorded trajectory data. The idea is to exploit the Karush–Kuhn–Tucker (KKT) condition, so that the a bijective mapping between observations and labels can be constructed.

Consider the MPC system configuration in (1), we further restrict the cost function to have a quadratic form and omit the final state cost function $F(\cdot)$,

$$C_\theta(x_i, u_i) = \frac{1}{2}(x_i^T Q_k x_i + u_i^T R_k u_i); \tag{3}$$

following this format, the output of the neural network should be the matrices Q_i and R_i, that is

$$\theta = (Q_k, R_k) = g_\psi(y_k), \tag{4}$$

where ψ represents the parameters of the neural network $g_\psi(\cdot)$. Along the dimension of time, the under-controlled system physically locates at step k, and the cost function parameters (Q_k, R_k) remain constant for the prediction steps i in equation (1). After the control input is generated and applied to the actual system and the actual system proceeds to step $k+1$, the cost function parameters will then re-adjust and the optimization problem will be solved again for step $k+1$.

Now, the original Problem 1 has been further concretized that we need to introduce a mechanism of using neural network to predict the parameters (Q_k, R_k) of cost function based on the observation sequence y_k. Given the observed samples of data sequence, our goal is to train a neural network such that it varies the MPC cost function parameters according to the observation to imitate the behavior encoded inside the data. Here we assume the hard constraint is always not active, which means the inequality constraint has always been satisfied before applying it. Consider the optimization described in (1) with cost function defined in (3), the Lagrangian is written as

$$L(\mathbf{X}, \mathbf{U}, \lambda) = \frac{1}{2} \sum_{i=k}^{k+N-1} (x_i^T Q_k x_i + u_i^T R_k u_i) + \lambda^T(k)\mathbf{F}(\mathbf{X}, \mathbf{U}) \tag{5}$$

where the variables are defined as follows,

$$\mathbf{X} = \text{col}(x_k,\ x_{k+1},\ ...,\ x_{k+N-1}) \qquad \mathbf{U} = \text{col}(u_k,\ u_{k+1},\ ...,\ u_{k+N-1})$$

$$\lambda(k) = \text{col}(\lambda_0(k),\ \lambda_1(k),\ ...,\ \lambda_{N-1}(k)), \quad \text{and} \quad \lambda_i(k) \in \mathbb{R}^n$$

$$\mathbf{F}(\mathbf{X}, \mathbf{U}) = \text{col}(f(x_k, u_k) - x_{k+1},\ ...,\ f(x_{k+N-1}, u_{k+N-1}) - x_{k+N}).$$

The necessary and sufficient conditions for optimality of solution to the problem (1) is the KKT-conditions [Boyd and Vandenberghe, 2004] as follows,

$$\frac{\partial L}{\partial \mathbf{X}} = \mathbf{Q}\mathbf{X} + \frac{\partial \mathbf{F}^T(\mathbf{X}, \mathbf{U})}{\partial \mathbf{X}}\lambda(k) = 0, \qquad \frac{\partial L}{\partial \mathbf{U}} = \mathbf{R}\mathbf{U} + \frac{\partial \mathbf{F}^T(\mathbf{X}, \mathbf{U})}{\partial \mathbf{U}}\lambda(k) = 0, \tag{6}$$

where

$$\mathbf{Q} = \text{diag}(Q_k, Q_k, ..., Q_k), \qquad \mathbf{R} = \text{diag}(R_k, R_k, ..., R_k).$$

Upon solving the MPC optimization problem at each time instant, a sequence of control inputs are generated, whereas only the first one is actually fed to and executed by the system. Hence in the recorded trajectory, each control input data point only represents the first element of the solution to the optimization problem (1) at each time step.

Suppose $\tilde{\mathbf{X}} = \text{col}(\tilde{x}_0, \tilde{x}_1, ..., \tilde{x}_l)$ is the recorded sequence of the system trajectories and $\tilde{\mathbf{U}} = \text{col}(\tilde{u}_0, \tilde{u}_1, ..., \tilde{u}_l)$ is the recorded sequence of the recorded system control inputs. Assuming those recorded trajectories are generated through a MPC solver, then each data pair needs to satisfy the following condition

$$Q_k\tilde{x}_k + \frac{\partial f^T(x, u)}{\partial x}|_{x=\tilde{x}_k, u=\tilde{u}_k}\lambda_0(k) = 0 \qquad R_k\tilde{u}_k + \frac{\partial f^T(x, u)}{\partial u}|_{x=\tilde{x}_k, u=\tilde{u}_k}\lambda_0(k) = 0. \tag{7}$$

At this point, we can see the first challenge of solving problem 1. Even given known system dynamics $f(x, u)$, one data pair $(\tilde{x}_k, \tilde{u}_k)$ is not enough to recover the matrices of Q_k and R_k.

We further restrict the form of the matrices Q_k and R_k to be diagonal, which means

$$Q_k = \text{diag}(q_k), \qquad R_k = \text{diag}(r_k),$$

where $\text{col}(q, r)$ should be generated by the prediction neural network model. The diagonal form of these matrices is commonly used in MPC applications. Thus the neural network model is represented as follows,

$$\text{col}(q_k, r_k) = g_\psi(\tilde{y}_k).$$

3.3 Model Training

Given a sequence of data pairs $((\tilde{y}_1, \tilde{x}_1, \tilde{u}_1),\ (\tilde{y}_2, \tilde{x}_2, \tilde{u}_2),\ ...,\ (\tilde{y}_l, \tilde{x}_l, \tilde{u}_l))$, towards satisfying the optimality condition described in (3.2), the loss function $\mathcal{L}$ is chosen as

$$\mathcal{L} = \sum_{k=0}^{l} J(\lambda_0(k)^*), \quad \lambda_0(k)^* = \text{argmin} J(\lambda_0(k)), \quad J(\lambda_0(k)) = \|G\lambda_0(k) + H\text{col}(q_k, r_k)\|, \tag{8}$$

where the matrices G and H are defined as

$$G = \begin{bmatrix} \frac{\partial f^T}{\partial x}|_{x=\tilde{x}_k, u=\tilde{u}_k} \\ \frac{\partial f^T}{\partial u}|_{x=\tilde{x}_k, u=\tilde{u}_k} \end{bmatrix} \qquad H = \text{diag}(\tilde{x}_k, \tilde{u}_k).$$

In this case, matrix G can be obtained from the system dynamics and matrix H can be constructed from the recorded trajectory data.

Here it can be seen another challenge of solving the proposed problem, which is

$$\lambda_0(k)^* = 0, \qquad \text{col}(q_k, r_k) = 0$$

is an optimal solution to the minimization problem (8).

To avoid the optimal solution being trapped at zero, we introduce the following procedure for updating the parameters ψ of the neural network $g_\psi(\cdot)$. The loss function in (8) is re-written as

$$J(\lambda_0(k)) = \|Uv\|, \qquad U = [G \quad H], \qquad v = \mathrm{col}(\lambda_0(k), q_k, r_k). \tag{9}$$

The row dimension of U is $n+m$ and the column dimension of U is $2n+m$, where n is the dimension of system state space and m is the dimension of the system input space. From (9), it can be seen that as long as v lies inside the null space of U, J is minimized. Now let the columns of a matrix W span the null space of U, i.e.,

$$\mathrm{Null}(U) = \mathrm{span}(w_1, w_2, ...)$$

where w_1, w_2,... are columns of matrix W. Hence, for any vector η, let

$$v = W\eta.$$

Then v is an optimal solution to the problem (8).

In this article, we exploit the idea of expectation-maximization(EM) algorithm for the training process. First of all, η is initialized with $\eta = \eta_0$. Then, given the known system dynamics and data pair, matrix W is calculated. Following that the guess value of the neural network output can be computed as $\tilde{o} = W\eta$. Finally, the loss of the neural network is defined as

$$\mathcal{L} = \sum \|\tilde{o} - o\|, \qquad o = g_\psi(y_k).$$

And the loss is back-propagated to update the parameters of the neural network. Meanwhile, η is updated by solving the least squares problem

$$\mathrm{minimize}_\eta \|W\eta - o\|.$$

These steps are then iterated until the neural network parameters converge. Summary of the algorithm is illustrated in algorithm 1.

4 Autonomous Vehicle Longitudinal Control

Algorithm 1: Inverse Reinforcement Learning MPC

```
Input: $\tilde{\mathbf{X}}$, $\tilde{\mathbf{U}}$ and $\tilde{\mathbf{Y}}$
Output: $g_\psi(\cdot)$
1 $\eta = \eta_0$, $\mathcal{L} = \mathcal{L}_0$ and $\bar{\mathcal{L}}$;
2 while $\mathcal{L} > \bar{\mathcal{L}}$ do
3     forward propagation $o = g_\psi(\tilde{\mathbf{Y}})$;
4     construct matrix $U$ ;
5     compute the null space matrix $W$ ;
6     update $\eta$ as $\eta = \mathrm{argmin}_\eta \|W\eta - o\|$ calculate
        the loss $\mathcal{L} = \sum \|W\eta - o\|$ ;
7     back propagation to update network
        parameters $g_\psi(\cdot)$ ;
8 end
```

In this section, the proposed approach is applied for the design of a longitudinal controller for autonomous vehicle. First a data-set generated through simulation is used for training and then a publicly available data-set is exploited for performance comparison.

4.1 Algorithm formulation

Model-based optimization method such as linear-quadratic-regulator (LQR) and MPC are largely developed and deployed for autonomous longitudinal vehicle control. However, substantial amount of complaints are reported because of the inconsistency between the algorithm generated behavior and human-expected behavior. One interesting example is how to approach a static vehicle stopping ahead of the plant (under-controlled) vehicle. Human driver may speed down far away and approach to the static vehicle at a lower speed; on the contrary, optimization based method usually commands the plant (under-controlled) vehicle to approach the static vehicle rather fast followed by a late braking to stop behind the static vehicle. The reason is that the cost function penalizes on the arrival time so that the vehicle travels as fast as possible to reduce the cost. Adjusting the cost function could be an option to improve the passenger experience, but manual parameters-tuning takes lots of effort and the same cost function may not necessarily improve the driving performance for every driving scenario.

The proposed scheme in this article on the other hands aims to alleviate such drawback by imitating the behavior of human through a recorded data set in the training phase and reconstruct the cost function based on the observation later in the deployment phase.

For such longitudinal control problem, we define the under-controlled autonomous vehicle as "ego vehicle" and call the vehicle in front as "leading vehicle". Furthermore, the states of the system and the system update function can be described as

$$x_i = \begin{bmatrix} d_l - d_d \\ v_l - v_e \\ v_d - v_e \end{bmatrix}, \qquad x_{i+1} = Ax_i + Bu_i, \qquad A = \begin{bmatrix} 1 & dt & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \qquad B = \begin{bmatrix} 0 \\ -dt \\ -dt \end{bmatrix};$$

where dt is the time step or sampling time of the system and the physical meanings of the variables are defined as $v_l \in \mathbb{R}$ represents the velocity of the leading vehicle; $v_e \in \mathbb{R}$ represents the velocity of the ego vehicle; $d_l \in \mathbb{R}$ represents the distance between ego vehicle and leading vehicle; $d_d \in \mathbb{R}$ represents a desired distance value between ego vehicle and leading vehicle; $v_d \in \mathbb{R}$ represents a desired speed of ego vehicle; $u_i \in \mathbb{R}$ represents the acceleration of the ego vehicle, which is also the control command generated by the algorithm.

As for the prediction model $g_\psi(y_k)$, in this example, we choose the observation y_k as the trajectory history of the previous 10 frames, i.e.,

$$y_k = \text{col}(x_k,\ x_{k-1},\ \dots,\ x_{k-9}).$$

We take use of a four-layer fully connected neural network model wit number of nodes shown in Figure 1. The activation functions are chosen as $Tanh(\cdot)$ for the first three layers and $Sigmoid(\cdot)$ for the last output layer to guarantee that Q_k and R_k are positive semi-definite. We train the model with batch size of 100 and learning rate of $1e-4$. The size of the training data varies between different experiments.

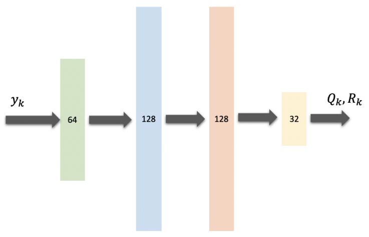
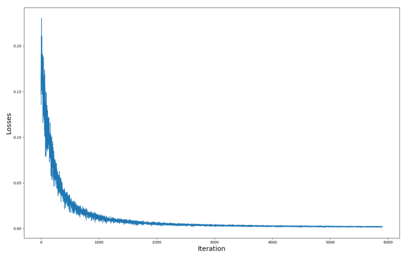


Figure 1: Neural Network Layout

4.2 Simulation result

In this part, the training data is generated through a simulation. The simulation scenario is defined as follows. The leading vehicle is first placed 50 meters ahead of the ego vehicle with a speed of 20 m/s. Then at the time $t = 100s$, the leading vehicle is switched to another one locates 40 meters ahead of the ego but with slower speed $15m/s$. Again at time $t = 200s$, the leading vehicle changes to one only 20 meters ahead with a speed $18m/s$. During the human driving data generation period, the leading vehicle speed deviates from the initial speed by a normal distribution. The ego vehicle starts with a speed of 25 m/s.



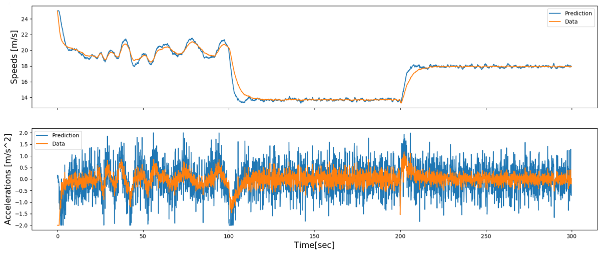


Figure 2: Simulation example result. Left: The loss history during the training phase; Right: Result comparison between simulated data and algorithm generated command. The upper figure shows the speed comparison and the lower figure shows the acceleration command comparison.

The history of the loss $\mathcal{L}$ is shown on the left hand side of Figure 2, where it can be seen that the parameters actually converge with a rather fast rate. The performance of the proposed method is demonstrated through another simulation, where the constant MPC controller is replaced by the proposed algorithm in this article. The comparison of the recorded and generated ego vehicle acceleration is shown on the right hand side of Figure 2. When confronting a similar scenario, the proposed algorithm is able to generate similar behavior with respect to the recorded trajectory.

4.3 Real-world data validation

To further demonstrate the algorithm, we use several data sets extracted from Next Generation Simulation (NGSIM) data set [US Department of Transportation, 2018]. In the NGSIM project, detailed vehicle trajectory data is collected using camera devices at specific locations including including southbound US 101 and Landershim Bounlevard, Los Angeles, CA, eastbound I-80 in Emeryville, CA and Peachtree Street in Atlanta, GA. Each trajectory provides the information of precise locations, speeds and relative distances of a vehicle with a time resolution of 0.1 second.

We extract 7 datasets from NGSIM data for the evaluation of the proposed algorithm. The training process is similar to the simulation data case. However, here the desired distance d_d is not explicitly known from the data-set, thus another prediction model is trained simultaneously for this value during the training process. Later to examine the algorithm performance, the sequences of the leading vehicle speeds and locations are reconstructed through the recorded data. A simulation is run by placing the ego vehicle at the same initial position, while it is fully controlled by the proposed control algorithm. The scheme of the implemented algorithm is shown in on the left hand side of Figure 3. On the right hand side of Figure 3, an example of the comparison between recorded data and simulation data is shown, where the simulated vehicle distance and speed are close to the ones in the recorded data.

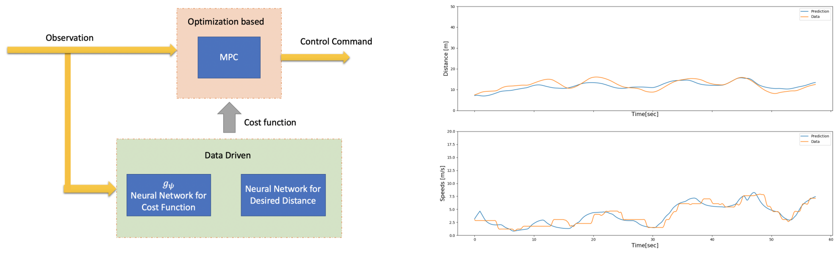


Figure 3: Real-world data result. Left: Algorithm scheme; Right: Sample result comparison

We choose the benchmark results in [Kesting and Treiber, 2008] as the baseline method, which investigates two of the popular car-following model, Intelligent Driver Model (IDM) and Velocity Difference Model (VDIFF). The IDM can be described mathematically as

$$\dot{v}_{IDM}(s, v, \Delta v) = a\big[1 - (\frac{v}{v_0})^4 - (\frac{s^*(v, \Delta v)}{s})^2\big],$$
$$s^*(v, \Delta v) = s_0 + vT + \frac{v\Delta v}{2\sqrt{ab}},$$

where v is the current vehicle velocity, v_0 is the desired velocity, Δv is the velocity difference v_0 and v, s is the current distance to the preceding vehicle and s^* is the desired distance; in this IDM model, a, b, s_0, v_0, T are parameters. Meanwhile, the VDIFF model is defined as

$$\dot{v}_{VDIFF}(s, v, \Delta v) = \frac{v_{opt}(s) - v}{\tau} - \lambda\Delta v$$
$$v_{opt}(s) = \frac{v_0}{2}\big[\tanh(\frac{s}{l_{int}} - \beta) - \tanh(-\beta)\big],$$

where definitions of v, s, Δv are same as IDM model and τ, λ, l_{int}, β are model parameters. In [Kesting and Treiber, 2008], the models are calibrated by approximately solving the nonlinear optimization problem through a generic algorithm [GOLDBERG, 2006]. In this algorithm, each new generation individual are generated by recombined two scholastically selected old generation individuals. This evolution terminates until convergence criterion reached.

We as well exploit the same performance indicators like [Kesting and Treiber, 2008] such as $relative\ error$ and $absolute\ error$ to measure the similarity between the data as follows,

$$\mathcal{F}_{rel}(s) = \sqrt{\left\langle \left(\frac{s^{sim} - s^{data}}{s^{data}}\right)^2 \right\rangle} \qquad \mathcal{F}_{abs}(s) = \sqrt{\frac{\langle (s^{sim} - s^{data})^2 \rangle}{\langle s^{data} \rangle^2}},$$

where the symbol $\langle \cdot \rangle$ represents the mean of a data sequence and s represents the position data. In addition, we define absolute difference measurements as follows,

$$\mathcal{E}_{mean}(s) = \langle |s_{sim} - s^{data}| \rangle \qquad \mathcal{E}_{var}(s) = \langle (s_{sim} - s^{data})^2 \rangle.$$

Table 1: Performance Evaluation

Dataset	$\mathcal{F}_{rel}(s)$	$\mathcal{F}_{abs}(s)$	$\mathcal{E}_{mean}(s)[m]$	$\mathcal{E}_{var}(s)[m]$	$\mathcal{E}_{mean}(v)[m/s]$	$\mathcal{E}_{var}(v)[m/s]$
1	13.9%	13.1%	1.352	1.626	0.627	0.791
2	22.3%	17.3%	3.384	4.091	1.071	1.350
3	13.1%	10.6%	2.234	2.638	0.631	0.859
4	13.6%	13.5%	3.050	3.490	0.531	0.702
5	22.1%	12.2%	3.205	3.713	0.606	0.838
6	19.3%	16.3%	2.152	2.683	0.533	0.841
7	17.5%	15.6%	2.418	2.965	0.855	1.049

The performance evaluation results are shown in Table 1. Compared with the result of [Kesting and Treiber, 2008] shown in Table 2. In the baseline result $relative\ error$ and $absolute\ error$ ranges 20% to 30%, our results show around 10% less. Besides, our result shows around 2 meters of position difference and less than $1\ m/s$ speed difference between the recorded data and simulated trajectory.

Table 2: Baseline Result

	IDM		VDIFF	
Dataset	$\mathcal{F}_{rel}(s)$	$\mathcal{F}_{abs}(s)$	$\mathcal{F}_{rel}(s)$	$\mathcal{F}_{abs}(s)$
1	24.0%	20.7%	25.5%	21.4%
2	28.7%	25.6%	29.1%	21.4%
3	18.0%	11.2%	28.2%	14.5%

5 Conclusion and Discussion

In this article, we propose a framework for inverse reinforcement learning with MPC by combining the neural network prediction with MPC control algorithm to imitate the behavior encoded inside a recorded data. The main challenge of such approach lies in the ambiguity of labeling and model parameters, where we approach problem by building a bijective mapping between the recorded data and prediction model output. Optimization and dynamic model-based autonomous vehicle longitudinal control algorithm nowadays suffers from the in-alignment with human intention. Whereas, the proposed method provides a solution. We present an implementation of such control algorithm using the proposed method. The result is compared with an existing approach and shows improved performance by reproducing similar behavior encoded within human-driven vehicle trajectory.

Current theory development faces several drawbacks. One is that the final cost function $F(\cdot)$ in (1) has to be dropped, otherwise it cannot be recovered anyway. Another one is that the hard constraints are assumed to be inactive. The main cause of such limitations is the commonly adopted design decision of MPC, that the optimization problem is solved recursively at each step and only the first part of control sequence is executed. Such set-up introduces large amount of ambiguity to solve the dual problem of optimization. Thus one of the further investigation direction is to break such setting and assume that more than one or all the generated control signals can be observed, in order to fully recover the cost functions and constraints. On the autonomous driving application side, feature extraction can be potentially much enhanced to increase the similarity of driving style between a human driver and an autonomous vehicle. One potential approach is to provide more surrounding traffic information not limited to the leading vehicle, as well as to provide local map information such as distances to lane boundaries and positions of the traffic intersections. The idea is that to make an autonomous vehicle drive like human, we may need to feed comparable observation as human receive when driving.

References

David Q Mayne, James B Rawlings, Christopher V Rao, and Pierre OM Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36(6):789–814, 2000.

S Joe Qin and Thomas A Badgwell. A survey of industrial model predictive control technology. *Control engineering practice*, 11(7):733–764, 2003.

Sergio Vazquez, Jose Leon, Leopoldo Franquelo, Jose Rodriguez, Hector A Young, Abraham Marquez, and Pericle Zanchetta. Model predictive control: A review of its applications in power electronics. *IEEE Industrial Electronics Magazine*, 8(1):16–31, 2014.

Tom Erez, Kendall Lowrey, Yuval Tassa, Vikash Kumar, Svetoslav Kolev, and Emanuel Todorov. An integrated system for real-time model predictive control of humanoid robots. In *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 292–299. IEEE, 2013.

David H Shim, H Jin Kim, and Shankar Sastry. Decentralized nonlinear model predictive control of multiple flying robots. In *42nd IEEE International Conference on Decision and Control (IEEE Cat. No. 03CH37475)*, volume 4, pages 3621–3626. IEEE, 2003.

F Künhe, J Gomes, and W Fetter. Mobile robot trajectory tracking using model predictive control. In *II IEEE latin-american robotics symposium*, volume 51, 2005.

Roman Schmied, Harald Waschl, Rien Quirynen, Moritz Diehl, and Luigi del Re. Nonlinear mpc for emission efficient cooperative adaptive cruise control. *IFAC-PapersOnLine*, 48(23):160–165, 2015.

Francesco Borrelli, Alberto Bemporad, Michael Fodor, and Davor Hrovat. An mpc/hybrid system approach to traction control. *IEEE Transactions on Control Systems Technology*, 14(3):541–552, 2006.

Francesco Borrelli, Paolo Falcone, Tamas Keviczky, Jahan Asgari, and Davor Hrovat. Mpc-based approach to active steering for autonomous vehicle systems. *International Journal of Vehicle Autonomous Systems*, 3(2):265–291, 2005.

Paolo Falcone, H Eric Tseng, Francesco Borrelli, Jahan Asgari, and Davor Hrovat. Mpc-based yaw and lateral stabilisation via active front steering and braking. *Vehicle System Dynamics*, 46(S1): 611–628, 2008.

Janick V Frasch, Andrew Gray, Mario Zanon, Hans Joachim Ferreau, Sebastian Sager, Francesco Borrelli, and Moritz Diehl. An auto-generated nonlinear mpc algorithm for real-time obstacle avoidance of ground vehicles. In *2013 European Control Conference (ECC)*, pages 4136–4141. IEEE, 2013.

Jose Rodriguez, Marian P Kazmierkowski, Jose R Espinoza, Pericle Zanchetta, Haitham Abu-Rub, Hector A Young, and Christian A Rojas. State of the art of finite control set model predictive control in power electronics. *IEEE Transactions on Industrial Informatics*, 9(2):1003–1016, 2012.

MR Arahal, F Barrero, S Toral, M Duran, and R Gregor. Multi-phase current control using finite-state model-predictive control. *Control Engineering Practice*, 17(5):579–587, 2009.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pages 305–313, 1989.

Huazhe Xu, Yang Gao, Fisher Yu, and Trevor Darrell. End-to-end learning of driving models from large-scale video datasets. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2174–2182, 2017.

Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1. ACM, 2004.

Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2, 2000.

Shao Zhifei and Er Meng Joo. A review of inverse reinforcement learning theory and recent advances. In *2012 IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE, 2012.

David Silver, J Andrew Bagnell, and Anthony Stentz. Learning from demonstration for autonomous navigation in complex unstructured terrain. *The International Journal of Robotics Research*, 29 (12):1565–1592, 2010.

Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

US Department of Transportation. NGSIM – Next Generation Simulation. `http://www.ngsim.fhwa.dot.gov`, 2018.

Arne Kesting and Martin Treiber. Calibrating car-following models by using trajectory data: Methodological study. *Transportation Research Record*, 2088(1):148–156, 2008.

DD GOLDBERG. Genetic algorithms in search, optimization, and machine learning genetic algorithms in search, optimization, and machine learning, 1989. *IEEJ Transactions on Electronics, Information and Systems*, 126(7):857–864, 2006.