2023-10-24

RL-MPC: Reinforcement Learning Aided Model Predictive Controller for Autonomous Vehicle Lateral Control

Abstract

This paper presents a nonlinear model predictive controller (NMPC) coupled with a pre-trained reinforcement learning (RL) model that can be applied to lateral control tasks for autonomous vehicles. The past few years have seen opulent breakthroughs in applying reinforcement learning to quadruped, biped, and robot arm motion control; while these research extend the frontiers of artificial intelligence and robotics, control policy governed by reinforcement learning along can hardly guarantee the safety and robustness imperative to the technologies in our daily life because the amount of experience needed to train a RL model oftentimes makes training in simulation the only candidate, which leads to the long-standing sim-to-real gap problem–This forbids the autonomous vehicles to harness RL's ability to optimize a driving policy by searching in a high-dimensional state space. The problem of robustness and constraints satisfaction can be alleviated by using NMPC technique which has proved itself in various industrial control tasks; however, traditional NMPC usually uses one fixed set of parameter matrices in its cost function while the changing path-tracking conditions faced by an autonomous vehicle may require the optimizer to place varying emphasis on different terms of the objective. Therefore, we propose to use a RL model to dynamically select the weights of the NMPC objective function while performing real-time lateral control of the autonomous vehicle (we call this RL-NMPC). The RL weight-search model is trained in a simulator using only one reference path, and is validated first in a simulation environment and then on a real Lincoln MKZ vehicle; the RL-NMPC achieved considerably better performance in lateral tracking during simulation and on-board tests.

Introduction

Autonomous driving (AD) has been one of the most attractive areas in both the automotive industry and academia. In this paper, we devote our attention to the lateral control stack inside the AD system, where the controller receives a pre-computed reference trajectory from the upper stream perception and planning modules, and then proceeds to solve for an optimal steering angle that minimizes the lateral error between the vehicle and the desired path while ensuring a smooth overall trajectory. The control module takes on the responsibility of making the autonomous vehicle stay on a safe path while cleaving to human-like driving behavior–this requires the lateral controller to be both accurate and robust when given a wide range of reference paths in different traffic situations, including cruising, turning, or lane changing–the immense cost of safety breach in an AD system makes it imperative for automotive practitioners and researchers to keep on searching for more stable and capable lateral control schemes.

Reinforcement learning (RL) has succeeded magnificently in generating capable and robust control policy for complex robotic systems governed by nonlinear dynamics–such as the Unitree A1 robot dog that can traverse diverse terrains relying solely on the low-level commands given by the deep RL model based on multi-modal transformers [1, 2]; the robot hand trained using model-based RL with the level of dexterity that enables it to play two Baoding balls single-handedly [3]; and the spectacular success of the auto-racing quadrotor Swift developed in [4] that beats human drone racing world champions while being controlled by deep RL model alone. However, these dazzling breakthroughs do not always transfer to real-world applications because of the famous sim-to-real problem in RL, where an RL agent trained in simulated environment fails in real-world scenarios due to reasons like: 1) the agent has not been trained for that experience because of the state-space complexity; 2) the modeling error in the training simulator (a problem especially prevalent in vehicle simulator where a tire or friction model with perfect fidelity is impossible) is reflected in the irrational behaviors of the agent in real world [5]. Recently, there has been attempt to create end-to-end large-scale deep learning models that map the sensory data on the autonomous vehicle directly to control commands [UniAD, Tesla], but a notable shortcoming for these effort is that the large neural network models similar to GPT and BERT require too much data to train [6, 7] with computational power not accessible by most automotive manufacturers, and even the data needed to train such a model requires considerable labor to prune and label in order to be training-fit. This motivates us to consider alternatives–can RL build upon other traditional control techniques, and thereby obtain lower bound performance while enhancing their capabilities?

Model predictive control (MPC) is a good candidate. It is a conventional control technique that precedes the blooming of RL in recent years and, similar to its artificial intelligence counterpart, has been successful in many applications such as controlling autonomous vehicle (both ground and aerial) [8, 9, 10, 11, 12, 13], and all the way from controlling CNC end effector, where the agent's goal is tracking a predetermined trajectory, to optimize power electronics system such as power converter, electrical drives, and static synchronous compensators (STAT-COMs) [5, 14, 15, 16, 17]. MPC is a special case of optimal control that solves an optimization problem constrained by the plant's dynamics, where the solution would usually contain a sequence of control input, with a predefined length (also referred to as horizon); more implementation details will be presented in the methodology section.

In this paper, we propose to improve the lateral tracking performance of the nonlinear MPC by combining it with a pre-trained RL model whose core function is to dynamically assign the weight matrices in the MPC's cost function. The goal is to adapt the lateral controller to

path of different shapes, and one way to achieve this is by controlling how much weight is put on each of the cost function terms–the dynamic selection of weight matrices can be characterized as a Markov Decision Process (MDP); therefore, the search for an optimal selection policy can be done by employing RL technique, where the policy is represented by a multi-layer perceptron (MLP), and objective of the RL training is to maximize the expected reward obtained by this policy. We demonstrated in both simulation and on-board experiments that the use of RL weight selection improves lateral tracking performance of the controller considerably. The RL algorithms applied, the design of the nonlinear MPC, and the structure of our deep reinforcement learning pipeline are discussed in more detail in the methodology section; and we report the testing result in both simulation and the real Lincoln MKZ vehicle in the result section.

Related Works

Deep Reinforcement Learning (DRL) Although DRL has attained great success in various robotics control tasks [1, 2, 3, 4], it has yet to realize its potential in controlling autonomous vehicles (AVs). As opposed to autonomous driving that requires an extremely robust controller that can map any sensory data to safe control input (steering, throttle, etc.), the world-champion level auto-racing drone developed in [4] only needs to operate under a constant environment–the racing tracks–thus demanding re-training of the DRL model for different racing settings. The pioneering effort to rely only on DRL techniques to control an autonomous vehicle comes from Alex et al. [18], where they employed the classic DRL algorithm DDPG [19] to train the autonomous vehicle to complete lane following task under an hour using only on-board computation power, with the input to the RL controller being a single monocular image, and output being steering angle plus speed setpoint. Jianyu takes the performance one step further in [20] by reducing the high-dimensional perception input to low-dimension bird-view representation before the RL algorithm is applied so that the training can cover a larger portion of the observation space, allowing the vehicle to cope with more complex traffic scenarios. The scenario of merging and negotiating in dense traffic is taken on by Dhruv et al. [21], where they applied model-free reinforcement learning network to train a policy that can open up gap in dense traffic while maneuvering with comfort; [21] achieves higher merging success rate than naive MPC, and is robust against the vehicle distributions on the road. Multi-agent RL (MARL) is also gaining interests among researchers because it gives insight about how autonomous vehicles interact with each other when one agent does not know the intention of those around it–this translates to AVs operating in dense traffics where other agents are controlled by human drivers; recent work [22] have tried transferring driving policy trained by MARL techniques in simulation to real-world execution. In summary, much of the labor is spent on reducing the dimensionality of the observation space so as to better generalize the trained policy but the DRL control schemes still tend to be restricted in simulated environment [19, 20, 21, 23, 24, 25] because of the tight safety constraints and extremely diverse real-world driving environment.

Learning-based MPC Model predictive control (MPC) can be viewed as akin to reinforcement learning in that they both seek to either maximize or minimize an objective function–for RL, the goal is to maximize the expected return over an entire trajectory characterized by Markov property and performs policy update iteratively guided by the reward received from executing an action in current state. For MPC, it is solving an oftentimes nonlinear optimization problem constrained by the system's dynamics model (thus the name "model"). The advantage of MPC is that it guarantees to satisfy hard constraints on the result, but its performance is largely dependent on its cost function design and the accuracy of system dynamics modeling. Researchers at ETH are able to enhance the lap time of their AMZ Driverless race car by performing Gaussian Process (GP) regression on-line to account for the residual model uncertainty in the MPC [26]. Ostafew et al. [27] proposed a robust MPC while incorporating a GP learned in real-time, allowing the mobile robot to work at its capability limits while staying within safety constraints. Another insight to interactions between learning and MPC is to utilize the cost function as a value function approximator within RL because the researchers found that economic NMPC can find optimal solutions after modification even if it has no access to accurate dynamics modeling [28]. The nominal work done by Bhardwaj et al. [29] developed an algorithm named MPQ(λ) that blends RL approximated cost function with the original MPC cost function, enabling the controller, when using faulty system dynamics, to achieve comparable performance to MPC using privileged dynamics model. Alternatively, G. Williams et al. choose to use model-based reinforcement learning to approximate the model dynamics, thereby generalizing model predictive path integral (MPPI, a subset of MPC methods) to systems with non-control-affine dynamics. Chen *et al.*'s work on using RL to determine the optimal triggering frequency provides another insight into lowering the computation cost of MPC [30].

Previous works either seek to create an end-to-end controller that maps observations available to the control signals for the autonomous vehicles, or try to amend the dynamics model using learning techniques; however, end-to-end controller based on RL is notoriously difficult to construct, and improving the model errors also requires the practitioners to be expert in both control and learning theory in order to correctly interpret the model error and apply learning methods appropriately. To get the best of both worlds, we propose to improve the nonlinear MPC performance by tuning the cost function parameters using model-free RL algorithms which have proven itself in complex optimization tasks. We use the cost function weight matrices as a handle to steer the performance of the MPC; therefore, the problem becomes searching for a set of optimal weight matrices everytime the MPC is invoked so that it obtains best performance, and this process can be modeled as a Markov Decision Process (MDP), where the task of the RL pipeline is to find a mapping from a state composed of vehicle states and MPC parameters to a set of weight matrices that maximize the reward–the tracking error and driving smoothness–at each time step. Using RL in hyperparameter tuning has precedents in the field of computer vision [31], where the hyperparameters used for object tracking algorithm is treated as a Deep Q-learning agent; and a similar work is conducted in [32]; both works report that modeling the tuning as a MDP and apply RL to fit for an optimal policy achieves better performance than the state-of-the-art methods; our work seeks to extend this technique to the improvement of nonlinear MPC.

Method

The Deep RL algorithms The MPC parameter search problem can be formulated as a finite-horizon Markov Decision Process (MDP) defined by M = (S, A, T, r), where S represents the state space of the agent, A its action space, T its transition dynamics, and r the reward corresponding to each state the agent is in. The goal of RL is to find the optimal policy θ^* such that [33]:

$$\theta^* = \arg\max_{\theta} \sum_{t=1}^{T} E_{(\mathbf{s}_t, \mathbf{a}_t) \sim \mathrm{p}_\theta(\mathbf{s}_t, \mathbf{a}_t)} [r(\mathbf{s}_t, \mathbf{a}_t)] \tag{1}$$

which means the optimal policy should maximize the sum of expected return *r* at state *s* taking action *a* (the p_θ represents the learned transition model of the MDP that's embedded in the learned policy). The deep RL algorithms use Deep Neural Network (DNN) as the decision making policy p_θ, or π_θ, and as approximation for the expected return, or value function, (characterized by the sum in Eqn. 1). Three following algorithms are tried to solve the search problem: Deep Deterministic Policy Gradient (DDPG) [34], Twin Delayed Deep Deterministic Policy Gradient (TD3) [35], and Proximal Policy Optimization (PPO) [36]. PPO belongs to *Policy Gradient* method which performs gradient descent directly on the expected return objective in Eqn. 1, and PPO seeks to perform the gradient descent in policy space, using a step size by either constraining the KL-divergence $D_{KL}(\pi_{old} \parallel \pi_{new})$ between the old and updated policy or simply clip the magnitude of the gradient, thereby avoiding too-large or too-small update each step[DRL book]. Similarly, DDPG

and TD3 belong to the combination of *Policy Gradient* and *Actor-critic* methods; actor-critic framework generally consists of two DNNs, one named actor network in charge of choosing the agent's next action, and the other named critic network that assigns performance score (value function) for the action. The two DNNs learn simultaneously so that the critic network can give an increasingly more accurate score for the actor network's output, and the actor network improves by maximizing the critic's score-more concretely, critic network seeks to minimize the TD error [33]:

$$J_{V_\psi^{\pi_\theta}}(\psi) = \frac{1}{2}\Big(\sum_{t=i}^{i+L-1} \gamma^{t-i} R_t + \gamma^L V_\psi^{\pi_\theta}(S_{i+1}) - V_\psi^{\pi_\theta}(S_i)\Big)^2 \quad (2)$$

while the actor network seeks to maximize [33]

$$J_\theta = \sum_t \log \pi_\theta(A_t|S_t)(R_t + \gamma V_\psi^{\pi_\theta}(S_{t+1}) - V_\psi^{\pi_\theta}(S_t)) \quad (3)$$

DDPG uses DNN as policy which makes it compatible with continuous state and action spaces; TD3 builds on DDPG by adding in clipped double q-learning, target policy smoothing, delayed policy update to make the training more stable with quicker convergence and less sensitive to hyper-parameter settings. The three algorithms are chosen have been validated in many demanding control tasks [1, 2, 4] and are recognized because of their training stability, convergence speed, and exploration-exploitation balance.

Vehicle Model The lateral controller in this paper adopts the bicycle model [37], which is shown to perform adequately in previous on-board validations [38]. The following quantities are needed: $x = [p_x \;\; p_y \;\; \phi]^T$ where $p_x p_y$ are the x y positions of the vehicle's Center of Gravity (CG) in the world frame; the time derivative of x: $\dot{x} = [\dot{p_x} \;\; \dot{p_y} \;\; \dot{\phi}]^T$; L_{xr} and L_{xf}, the distances from CG to vehicle rear and frontal axles; u_f and u_r, the front and rear wheel steering angle, respectively ($u_r = 0$ for conventional front-steering vehicles); and finally, β, the slip angle, calculated by

$$\beta = \arctan\Big(\frac{L_{xr}\tan(u_f)}{L_{xr} + L_{xf}}\Big) \quad (4)$$

The bicycle model is given below:

$$\begin{aligned} \dot{p_x} &= V\ \cos(\phi\ +\ \beta) \\ \dot{p_y} &= V\ \sin(\phi\ +\ \beta) \\ \dot{\phi} &= \frac{V\cos\beta}{L_{xf} + L_{xr}}(\tan(u_f) - \tan(u_r)) \end{aligned} \quad (5)$$

MPC Cost Function The model predictive control (MPC) scheme seeks a set of optimal solution $\{(x_1, u_1), (x_2, u_2), ..., (x_H, u_H)\}$ by optimizing over a cost function constrained by the system's dynamics model (Eqn. 5).

$$\begin{aligned} \min_{\mathbf{x,\,u}} J(x,u) = &\sum_{t=1}^{H} (x(t) - x_d(t))^T \mathcal{Q}(x(t) - x_d(t)) \\ &+ u(t)^T \mathcal{K} u(t) + (\Delta u(t))^T \mathcal{P}(\Delta u(t)) \\ &\text{subject to:} \\ x_{t+1} = x_t &+ f(x_t, u_t)\delta t \\ \mathbf{g(x)} &\leq 0 \end{aligned} \quad (6)$$

where x represents the system state and u the control input; in this paper, system state is the state of the vehicle under control, and the control input is the steering angle, and H denotes the control horizon, e.g. the overall length of the prediction. $\mathcal{Q}$, $\mathcal{K}$, $\mathcal{P} \in \mathbb{R}^{d\times d}$ are square

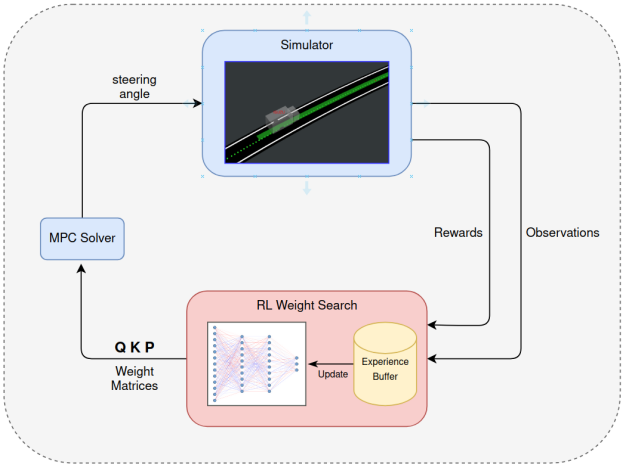


Figure 1: The training cycle for the RL weight search module. The output of the RL are a set of matrices (or scalars) that is used in the cost function of the MPC; the solution of the MPC (steering angle u_t) is then executed in the simulator; and the simulator returns the observations of the environment and calculated reward r to the RL module which is updated using the information feedback from the simulator.

diagonal matrices that assign weight to each of the term in the cost function; the higher the weight for that term, the impact of that term will be more prominent in the final solution; for example, let $\mathcal{P}$'s diagonal entries have a non-zero value, and $\mathcal{Q}$, $\mathcal{K}$ be zero matrices; then the resulting solution will only fulfill the requirement of the last term containing matrix $\mathcal{P}$ where the adjacent control inputs will have minimum difference but the solution does not minimize the first and second term of the cost function. The $f(x_t, u_t)$ is the vehicle dynamics model defined by (5), and **g(x)** includes all the inequalities constraining the vehicle states and control inputs. The goal of the RL pipeline is to choose the three weight matrices dynamically in real time so that the MPC can adapt to path segments of various shapes; for example, if the road ahead is straight-line, the first waypoint-tracking term should not take the highest weight, while the second and third term representing the steering smoothness should be the most important in making the vehicle cruising stable and comfortable instead of control inputs that result in zig-zag path of the vehicle due to the path-tracking term.

Training Framework The training for the RL weight matrices search (later referred to as RL pipeline for brevity) pipeline (figure 1) is done in a custom-built simulator in Robot Operating System (ROS), where the maps are built by recorded GPS coordinates of real-world roads. ROS is used mainly for visualization purposes and all the physics simulation (e.g. vehicle dynamics) can be wrapped in one script, boosting the training efficiency by avoiding the message publishing and subscribing mechanism inherent to ROS. Since the RL pipeline is only responsible for optimizing the lateral control of the autonomous vehicle, we choose to assign a constant longitudinal velocity for the vehicle in every lap it traverses (design choices will be discussed in further detail). During each time step of the training, the MPC sends the steering signal u_t to the vehicle, whose states s_{t+1} are then updated using the vehicle dynamics model, after which the reward r for the steering signal u_t is calculated; the collected observations and the reward are then be sent to the RL module, based on which the RL weight search module is updated.

RL Input and Reward Design The *input space*, or the *state space* of the RL agent, consists of the following components:

$$\mathbf{s}_t = (e_t,\ \phi_t,\ v_x^t,\ v_y^t,\ \dot{\phi}_t, param(\mathbf{C}))$$

where e_t represents the lateral tracking error of the vehicle at time t; ϕ_t is the yaw angle; v_x^t and v_y^t are the x and y component vehicle

velocity in world frame; $\dot{\phi}_t$ is the yaw rate at that instant; and the last term $param(\mathbf{C})$ is a vector with the coefficients of a polynomial curve that fits the coming segment of waypoints, which serves as a prompt for the RL algorithm that encodes the shape of the vehicle path, resembling human vision that provides look-ahead information for the human drivers to modify their driving decisions. One important consideration in determining the state space of the RL agent is its dimensionality; a low-dimensional state space means the real-world and training simulation would have a smaller gap than that of a high-dimensional state space, thus leading to more stable and robust RL weight selection agent since it has richer knowledge regarding the real-world environment it operates in. Following this insight, vehicle lateral tracking error e_t is used as input instead of the x-y position of the vehicle, which, in theory, is boundless in its complexity; conversely, e_t is much more tangible and bounded in its value. For the same reason, polynomial curve is chosen in place of spline curve when generating the *path shape* prompt because polynomial curve requires less number of coefficients to define, allowing the RL agent to explore in a lower-dimensional input space, thus increasing the training efficiency significantly; polynomial fit is also shown to capable of providing adequate fitting quality especially when the geometry is simple-like the shape of a real-world road segment. The polynomial fitting is done using Numpy [39] *polyfit* function; experiments show that, when the horizon length H = 10 (meaning the road segment contains H number of map waypoints), a second order polynomial is sufficient in capturing the shape of the path segments.

Reward function for the RL pipeline is composed of the following:

$$r = r_{\text{tr}} + r_{\text{steer}} \tag{7}$$

where r_{tr} is the reward for small lateral tracking error, and r_{steer} is to reward smooth steering angle sequence; each term is defined in Eqn. 8.

$$r_{\text{tr}} = \begin{cases} \frac{0.02}{\|e_t\|+5\cdot 10^{-4}} & \text{if } e_t < \epsilon \\ -2.5\,\|e_t\| & \text{if } e_t \geq \epsilon \end{cases} \tag{8a}$$

$$r_{\text{steer}} = \begin{cases} 10^5\,\|2\cdot 10^{-4} - \Delta u\| & \text{if } \Delta u \leq 2\cdot 10^{-4} \\ -200 & \text{if } \Delta u > 2\cdot 10^{-4} \end{cases} \tag{8b}$$

where e_t is the vehicle lateral tracking error at time step t, and ϵ is the threshold below which the RL agent will be awarded. In r_{tr}, the positive term is designed to be reciprocal function of the form $f(x) = \frac{1}{x+c}$ so the reward increases at a faster rate when it gets closer to zero, and the constant c exists to avoid undefined value error. It turns out that in order to guide the MPC to output smooth control sequence $\mathbf{u}_{1:H}$, the r_{steer} term needs to dominate, and this turns out to be a reasonable design choice such that both the tracking error and smoothness converge to target performance as training progresses.

Multiprocessing Training To boost the training efficiency and the utilization of Graphic Processing Unit (GPU), multiple independent simulations are run in parallel. In single-thread training set-up, there is only one active deep RL network, which is updated after a batch of experience tuples (s, a, r, s') are collected and the RL algorithm must wait for the experience collection is completed; but multi-threaded training can perform updates asynchronously, by using one global network and multiple local networks. During training, each agent is controlled by one local network to collect experience tuples in its own independent simulation; and the global network can be updated as soon as one of the local agent completes experience collection without the need to wait for other agents to complete [40]. Moreover, the parallel simulation set-up has also been used for experience collection only where the experience buffer can be filled more quickly. Figure 2 shows the multi-threaded training structure utilized in this paper.

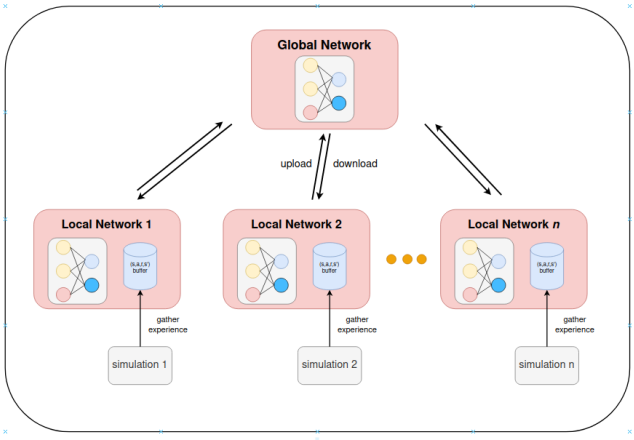


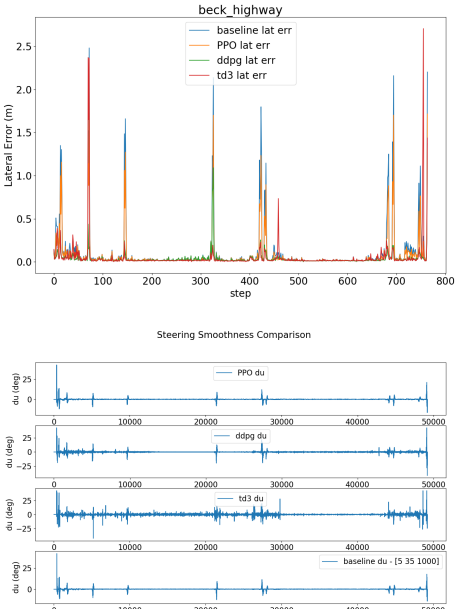
Figure 2: The multi-threaded training structure. n number of local agents work in parallel to collect experiences from their own simulation environment in order to perform global network update asynchronously or fill the global buffer more efficiently; the local agent downloads the updated global network each time the global network is updated.

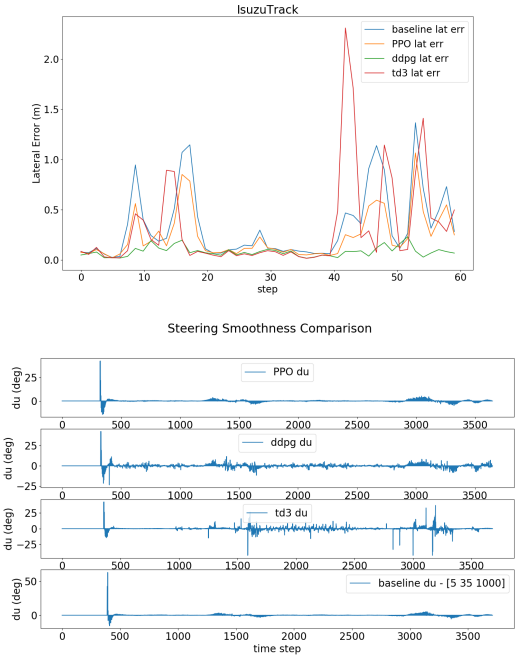
Results

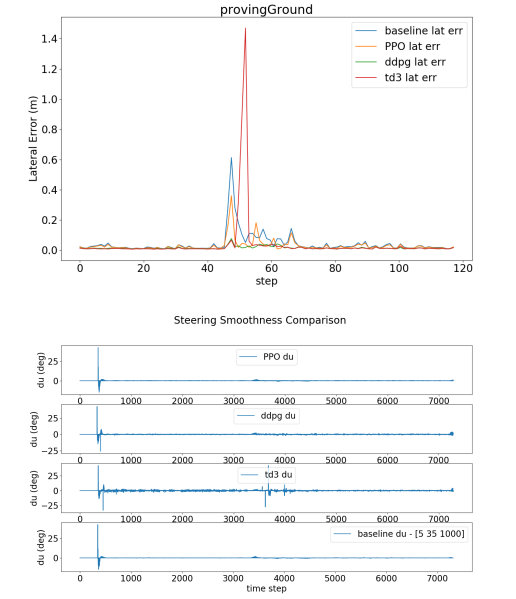
We performed two tests to validate the performance of the weight selection RL model. First, the model is tested in simulation environment, where the simulated vehicle is tasked with tracking a series of recorded waypoints (coordinates of real road recorded by vehicles with human drivers) and the performance is measured by the following two criteria: its lateral tracking error and smoothness of the MPC-generated steering angles; secondly, we deployed the RL weight selection model on a real Lincoln MKZ vehicle and performed tests on the testing track in Isuzu Technical Center of America, and performance is evaluated using the same two benchmarks as in simulation tests.

Simulation Test Here we present the simulation tests result. The tests are run on four different maps, all from recorded waypoints of test tracks and public roads. For simulation tests we tested the performance of all three RL algorithms: DDPG, TD3, and PPO. We also included a group of static weight parameters for comparing purposes; this static set of weights come from our previous trials of tuning the MPC, and it is chosen in order to: 1) investigate how dynamic assignments of weight parameters impact the vehicle's lateral tracking performance; 2) gain insights for RL's ability to tune a controller by adjusting high-level hyper-parameters comparing against human preliminary tuning.

As shown in Figure 3, the RL weight selection scheme attains comparable or better performance in both lateral tracking error and steering smoothness. It is to be observed that when the model has good lateral tracking performance, the steering smoothness might be sacrificed because the vehicle may adjust its bearing too frequently, leading to vacillating steering angles; this is especially true for the DDPG algorithm which has adequate lateral tracking on the Isuzu test track (green line) but has rather unstable steering angle trajectory. The overall best performing RL weight selection model is the PPO algorithm, which achieves good balance between lateral tracking and steering smoothness; the average lateral tracking error and the steering smoothness of each RL algorithm alongside the map on which it is tested are summarized in Table 1 and 2.







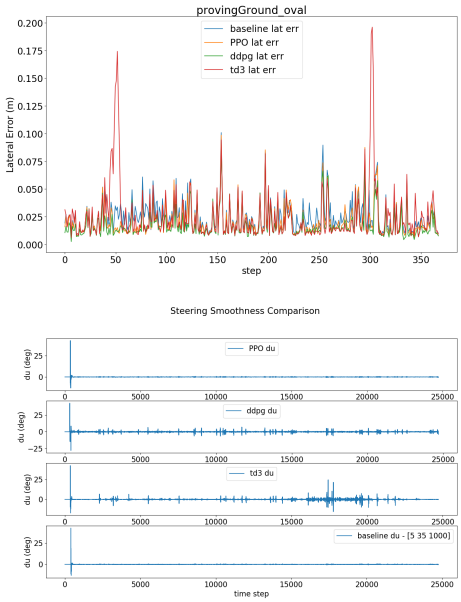


Figure 3: Lateral tracking error and steering smoothness (the difference between consecutive steering angles) for simulated tests using four different real-world waypoints maps.

Table 1: Lateral Tracking Errors (meter)

	Beck Highway	Isuzu Track	P.G.	P.G. Oval
Baseline	0.125	0.404	0.0931	0.0427
DDPG	**0.0533**	**0.157**	**0.0523**	**0.0313**
TD3	0.0615	0.385	0.0786	0.0422
PPO	0.0987	0.301	0.0758	0.0377

Table 1 lists out the average lateral tracking error (each lateral error data point is calculated at the end of a 22-waypoint segment) for each RL algorithm tested on four maps in simulation.

Table 2: Steering Smoothness (degrees)

	Beck Highway	Isuzu Track	P.G.	P.G. Oval
Baseline	**0.133**	**0.384**	**0.0737**	**0.0404**
DDPG	0.227	0.660	0.178	0.171
TD3	0.390	0.897	0.368	0.165
PPO	0.140	0.405	0.0826	0.0475

Table 2 lists out the average steer angle difference between consecutive control inputs for each weight selection method on four test maps.

We also provide here the value of each of the weight matrices of the MPC objective during simulation test on Isuzu test track (the most representative test with reasonable track length and variety of path shapes) in Figure 4, 5, and 6

It can be observed that the value of steering smoothness weight (Figure 6) is at all time steps nearly 100 times larger than the other two. This is largely determined by the reward function design during training, where the steering smoothness is rewarded a large positive feedback to ensure that the RL-MPC puts the passenger comfort and steering stability as its priority-this conforms to human driver behavior since driver does not need to adjust its steering frequently

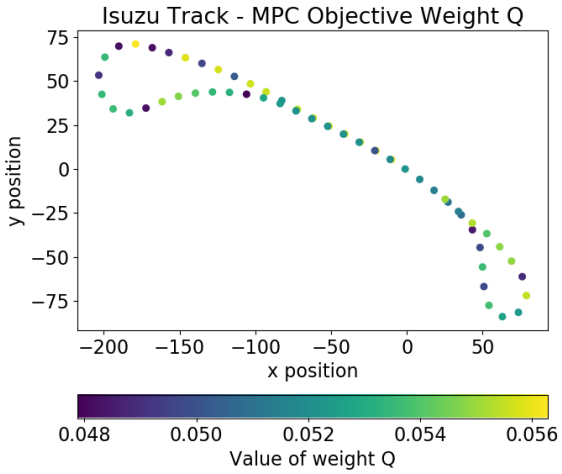


Figure 4: The weight matrix $\mathcal{Q}$ for the lateral error term in MPC objective.

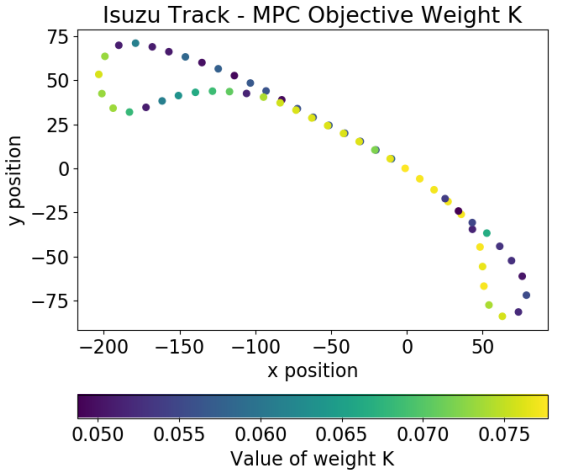


Figure 5: The weight matrix $\mathcal{K}$ for the lateral error term in MPC objective.

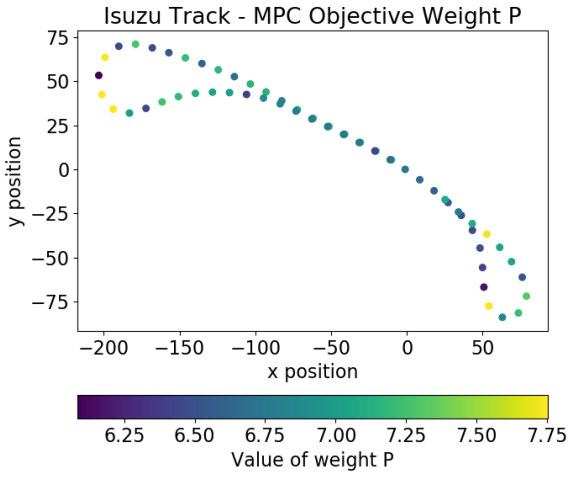


Figure 6: The weight matrix $\mathcal{P}$ for the lateral error term in MPC objective.

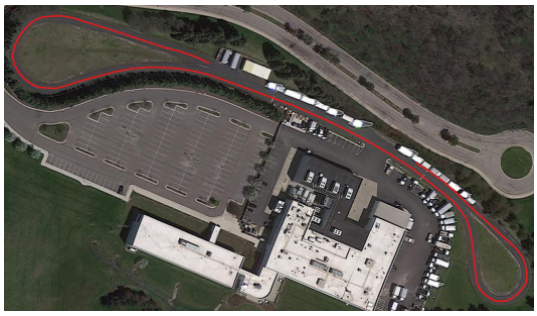


Figure 7: The testing track inside the Isuzu Technical Center of America facility.

Figure 8: The full-sized sedan used for the RL-MPC on-board testing; equipped with drive-by-wire system, Polynav 2000P GNSS-inertial system, a Calmcar front view camera, and a Dspace Autera computing unit.

when traveling on low-curvature road. This is reflected in the selection of weight $\mathcal{K}$ and $\mathcal{P}$, which is given relatively high value during low-curvature road (Figure 5 6) to ensure the vehicle is traveling smoothly in straight segments; RL algorithm seems to choose to sacrifice the lateral tracking accuracy to keep steering smoothness when the vehicle performs large angle turning by choosing relatively lower weight $\mathcal{Q}$ (Figure 4) when the road curvature starts to increase.

On-board Testing The on-board testing of the weight selection scheme is done on a real Lincoln MKZ vehicle. The vehicle is modified so that its low level controller can communicate with the host laptop via Robot Operating System (ROS). The RL model is run on the host laptop along with the MPC, and then the computed steering angle will be sent to the low level controller on the vehicle in the form of ROS message; similarly, the states of the vehicle are sent to the host laptop via ROS messages. The vehicle used is the same as the one in our previous work [38], where the distances from the front and rear axles to the vehicle's center of gravity are 1.2m and 1.65m, respectively; additionally, the vehicle has a full drive-by-wire system that can communicate with the host computer via ROS, a Polynav 2000P GNSS-inertial system, a Calmcar front view camera, and a Dspace Autera computing unit (Figure 8). The on-board test is carried out on the test track in Isuzu Technical Center of America facility (Figure 7).

The lateral tracking errors and the steering angles generated during the on-board test using the PPO algorithm RL-MPC are shown. It can be observed that the average lateral error in real-car test is larger than the result in simulation test, and the steering commands generated are also more aggressive than that in the simulation tests. One possible explanation for this is that the modeling error in the MPC constraint is magnified in on-board testing, and incorrect dynamics leads to off-target predictions of the vehicle's positions in the prediction horizon, which is then propagated to the RL predicted weights; consequently, the heading offset of the vehicle may be larger than the MPC predicted, so it is forced to generate a higher than anticipated steering angle to try to correct the vehicle's orientation, thus the aggressive steering command trajectory.

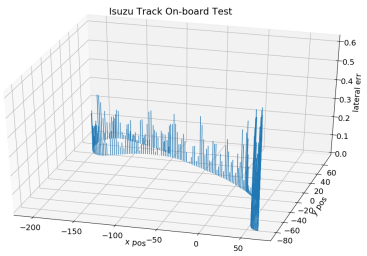


Figure 9: The lateral tracking error obtained during on-board testing. The vertical axis represents the lateral error, and the x y axis represents the location of vehicle when the lateral error is calculated.

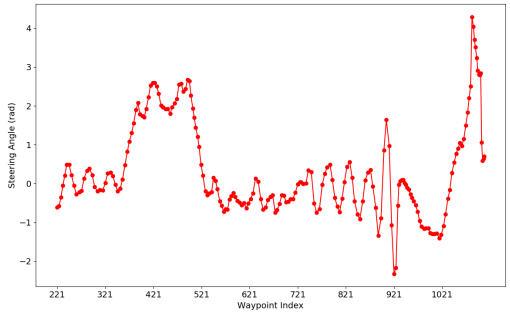


Figure 10: The steer angles of the vehicle during on-board testing. The x axis is the index of the waypoint the vehicle is passing through, and y axis is the corresponding steering angle at that waypoint.

Conclusion

In this paper we explored the possibility of applying reinforcement learning (RL) algorithms to improve the lateral controller on autonomous vehicle. We propose to use RL to tune the high-level decision parameters of the MPC-the weight matrices in MPC's objective so as to make the MPC more robust to trajectories of various shapes. The RL-MPC show promising lateral tracking ability in the simulated environment where the major difference from the real world would be the vehicle dynamics model, and RL-MPC, in simulated environment, obtain better performance than MPC that uses hand-tuned static weight matrices; the RL-MPC also obtains reasonable performance in real-world on-board testing, where the ROS communication framework and MPC solver cause latency in the control pipeline that hinders its performances. Next step would be to work on the domain randomization of the RL training so that it is more capable of handling unseen environments in the real-world and more robust against system noises, communication lags, and vehicle dynamics modeling inaccuracies.

References

1. R. Yang, G. Yang, and X. Wang, "Neural volumetric memory for visual locomotion control," 2023.
2. R. Yang, M. Zhang, N. Hansen, H. Xu, and X. Wang, "Learning vision-guided quadrupedal locomotion end-to-end with cross-modal transformers," 2022.
3. A. Nagabandi, K. Konoglie, S. Levine, and V. Kumar, "Deep dynamics models for learning dexterous manipulation," 2019.
4. E. Kaufmann, L. Bauersfeld, A. Loquercio, M. Müller, V. Koltun, and D. Scaramuzza, "Champion-level drone racing using deep reinforcement learning," *Nature*, vol. 620, pp. 982–987, Aug 2023.
5. P. Karamanakos, E. Liegmann, T. Geyer, and R. Kennel, "Model predictive control of power electronic systems: Methods, results, and challenges," *IEEE Open Journal of Industry Applications*, vol. 1, pp. 95–114, 2020.
6. A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018.
7. J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019.
8. A. Romero, S. Sun, P. Foehn, and D. Scaramuzza, "Model predictive contouring control for time-optimal quadrotor flight," 2022.
9. C. Sun, X. Zhang, Q. Zhou, and Y. Tian, "A model predictive controller with switched tracking error for autonomous vehicle path tracking," *IEEE Access*, vol. 7, pp. 53103–53114, 2019.
10. Y. Liu, L. Wang, and M. Brandt, "Model predictive control of laser metal deposition," *The International Journal of Advanced Manufacturing Technology*, vol. 105, pp. 1055–1067, Nov 2019.
11. D. Stenger, M. Ay, and D. Abel, "Robust parametrization of a model predictive controller for a cnc machining center using bayesian optimization," 2020.
12. Z. Zhou, C. Rother, and J. Chen, "Event-triggered model predictive control for autonomous vehicle path tracking: Validation using carla simulator," *IEEE Transactions on Intelligent Vehicles*, vol. 8, no. 6, pp. 3547–3555, 2023.
13. C. Rother, Z. Zhou, and J. Chen, "Development of a four-wheel steering scale vehicle for research and education on autonomous vehicle motion control," *IEEE Robotics and Automation Letters*, vol. 8, no. 8, pp. 5015–5022, 2023.
14. J. Rodríguez and P. Cortés, *Predictive Control Power Converters and Electrical Drives*. UK: Wiley, 2012.
15. T. Geyer, *Model Predictive Control High Power Converters and Industrial Drives*. NJ, USA, Wiley, 2016.
16. P. Cortes, M. P. Kazmierkowski, R. M. Kennel, D. E. Quevedo, and J. Rodriguez, "Predictive control in power electronics and drives," *IEEE Transactions on Industrial Electronics*, vol. 55, no. 12, pp. 4312–4324, 2008.
17. S. Kouro, P. Cortes, R. Vargas, U. Ammann, and J. Rodriguez, "Model predictive control—a simple and powerful method to control power converters," *IEEE Transactions on Industrial Electronics*, vol. 56, no. 6, pp. 1826–1838, 2009.
18. A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J.-M. Allen, V.-D. Lam, A. Bewley, and A. Shah, "Learning to drive in a day," 2018.
19. T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2019.
20. J. Chen, B. Yuan, and M. Tomizuka, "Model-free deep reinforcement learning for urban autonomous driving," 2019.
21. D. M. Saxena, S. Bae, A. Nakhaei, K. Fujimura, and M. Likhachev, "Driving in dense traffic with model-free reinforcement learning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, may 2020.
22. E. Candela, L. Parada, L. Marques, T.-A. Georgescu, Y. Demiris, and P. Angeloudis, "Transferring multi-agent reinforcement learning policies for autonomous driving using sim-to-real," 2022.
23. P. Wolf, C. Hubschneider, M. Weber, A. Bauer, J. Härtl, F. Dürr, and J. M. Zöllner, "Learning how to drive in a real world simulation with deep q-networks," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, pp. 244–250, 2017.
24. A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani, "Deep reinforcement learning framework for autonomous driving," *Electronic Imaging*, vol. 29, pp. 70–76, jan 2017.
25. J. Chen, Z. Wang, and M. Tomizuka, "Deep hierarchical reinforcement learning for autonomous driving with distinct behaviors," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1239–1244, 2018.
26. J. Kabzan, L. Hewing, A. Liniger, and M. N. Zeilinger, "Learning-based model predictive control for autonomous racing," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3363–3370, 2019.
27. C. J. Ostafew, A. P. Schoellig, and T. D. Barfoot, "Robust constrained learning-based nmpc enabling reliable mobile robot path tracking," *The International Journal of Robotics Research*, vol. 35, pp. 1547 – 1563, 2016.
28. S. Gros and M. Zanon, "Data-driven economic NMPC using reinforcement learning," *IEEE Transactions on Automatic Control*, vol. 65, pp. 636–648, feb 2020.
29. M. Bhardwaj, S. Choudhury, and B. Boots, "Blending mpc and value function approximation for efficient reinforcement learning," 2021.
30. J. Chen, X. Meng, and Z. Li, "Reinforcement learning-based event-triggered model predictive control for autonomous vehicle path following," in *2022 American Control Conference (ACC)*, pp. 3342–3347, 2022.
31. G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou, "Information theoretic mpc for model-based reinforcement learning," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1714–1721, 2017.
32. X. Dong, J. Shen, W. Wang, Y. Liu, L. Shao, and F. Porikli, "Hyperparameter optimization for tracking with continuous deep q-learning," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 518–527, 2018.
33. Z. D. Hao Dong, *Deep Reinforcement Learning: Fundamentals, Research, and Applications*. Springer Nature, 2020.
34. T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2019.
35. S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," 2018.

36. J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

37. R. Rajamani, *Vehicle dynamics and control*. Springer Science and Business Media, 2011.

38. Z. Zhou, J. Chen, M. Tao, P. Zhang, and M. Xu, "Experimental validation of event-triggered model predictive control for autonomous vehicle path tracking," in *2023 IEEE International Conference on Electro Information Technology (eIT)*, pp. 35–40, 2023.

39. C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, sep 2020.

40. V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," 2016.