Learning-Based Synthesis of Robust Linear Time-Invariant Controllers

Marc-Antoine Beaudoin and Benoit Boulet, Senior Member, IEEE

Abstract—Recent advances in learning for control allow to synthesize vehicle controllers from learned system dynamics and maintain robust stability guarantees. However, no approach is well-suited for training robustly-stabilizing linear time-invariant (LTI) controllers using arbitrary learned models of the dynamics. This article introduces a method to do so. It uses a robust control framework to derive robust stability criteria. It also uses simulated policy rollouts to obtain gradients on the controller parameters, which serve to improve the closed-loop performance. By formulating the stability criteria as penalties with computable gradients, they can be used to guide the controller parameters toward robust stability during gradient descent. The approach is flexible as it does not restrict the type of learned model for the simulated rollouts. The robust control framework ensures that the controller is already robustly stabilizing when first implemented on the actual system and no data is yet collected. It also ensures that the system stays stable in the event of a shift in dynamics, given the system behavior remains within assumed uncertainty bounds. We demonstrate the approach by synthesizing a controller for simulated autonomous lane-change maneuvers. This work thus presents a flexible approach to learning robustly stabilizing LTI controllers that takes advantage of modern machine learning techniques.

Index Terms—Machine learning, robust stability, robust control, safety guarantee.

I. INTRODUCTION

MACHINE learning can be used to improve the performance of controllers for intelligent vehicles. Typically the performance increase is enabled by a reduction in the modeling uncertainty. In fact, the fundamental tradeoff between robustness and performance in control engineering [1] reveals how modeling uncertainty can oftentimes be the limiting factor in system performance. However, it can be hard to maintain theoretical guarantees of stability and safety when incorporating learning components in control methods.

For vehicular control, neural network controllers can be trained with deep reinforcement learning and reach interesting levels of robustness [2], [3]. Domain randomization [4], [5] is often used to increase robustness. The idea is to randomly alter the physics of the simulated environment used by the learning algorithm, which better prepares the controller to be deployed in the real world. But it remains that whenever possible, it is preferable to maintain theoretical guarantees, and domain randomization does not.

In recent years, several lines of research presented successful approaches to theoretical guarantees. A safety framework [6] based on Hamilton-Jacobi reachability analysis [7] can be used to overwrite any learning controller in the event that the system is suspected to exit a given safe set. As more data is gathered, the safety framework gradually allows the learning controller to come closer to the unsafe region, as it becomes more certain of the system dynamics through learning with Gaussian processes (GP) [8], [9]. Another approach [10] consists of using Lyapunov theory to characterize the system’s region of attraction (ROA), and tune a neural network (NN) controller with gradient descent while considering the constraints imposed by the ROA. Here again, GPs are used to learn the system dynamics. The size of the ROA and the controller performance both gradually increase as a result of controller tuning and data collection. In two closely related approaches [11], [12], a quadratic programming controller is used to adjust a nominal controller’s output as to enforce a safety constraint. In [11] the safety constraint is defined using a control Lyapunov function, and in [12], a control barrier function (CBF). In these approaches the safety is contingent on having an accurate model of the system dynamics, so a NN is trained from collected data and used to improve the model’s accuracy. A time-varying CBF is introduced in [13] to enforce stability for vehicular control. In [14] learned control barrier functions are used to keep vehicles within state constraints despite modeling uncertainty, and the safe uncertainty-learning principle is introduced to guide the process of uncertainty estimation. In [15] researchers devised an outer-loop controller to adjust the output of a nominal controller. The model uncertainty estimates are updated in real time using the variance of a GP regression in order to better adjust the controller’s aggressiveness with respect to the local system configuration. Researchers in [16] use machine learning to estimate a linear system model along with error bounds, from which they can obtain a robust linear controller. The method can also be used to identify systems having linear dynamics with respect to nonlinear features of states and inputs [17]. It can also be used to design robust linear controllers for systems with complex high-dimensional sensorial inputs like camera-based autonomous vehicles [18], thereby taking steps toward formal guarantees even for the most complex problems. Finally, model predictive control (MPC) methods—which are
often used for vehicular control [19], [20]—can also benefit from learning [21]. In [22] researchers learn the system dynamics with GPs, which provides an uncertainty description than can be used in a stochastic MPC formulation in order to obtain probabilistic guarantees of keeping the system within state constraints.

In this study, we introduce a control learning method that is appropriate for robustly-stabilizing linear time-invariant (LTI) controllers, which typically have the state-space representation

\[ x_c = A_K x_c + B_K e, \]

(1)

\[ u = C_K x_c + D_K e, \]

(2)

where \( e \) is the error signal fed to the controller, \( x_c \) are the controller states, and \( u \) are the controller output signals. A time-proven method to synthesize such a controller is \( H_\infty \) loop shaping within a robust control framework [23]—see [24], [25] for instances. To do so, one assumes a nominal linear system model along with uncertainty bounds, defines performance criteria in terms of filters on the input and output signals, and solves a norm-minimization problem. However, \( H_\infty \) loop shaping can be difficult to implement in practice. Notably, the method is restricted to linear system models, so it can be hard to optimize closed-loop performance when the system is best described with a nonlinear model. As a result, the method is ill-suited for taking advantage of system identification through modern machine learning techniques, as the learned models are often nonlinear. Finally, it can be hard to translate the desire to track specific temporal reference trajectories into designs filters. Thus the objective in this research is to develop a LTI controller synthesis method that can take advantage of nonlinear learned models while ensuring robust closed-loop stability through a robust control framework.

Such a method does not exist in the literature. The approaches in [6], [10]–[12], [15] all consist of adjusting a base controller output according to a safety criteria computed in real time, but they do not tune the base controller parameters. Moreover, the approaches of [6], [10] suffer from the curse of dimensionality, which makes them ill-suited for multi-input multi-output systems with several states—a typical application for LTI controllers. The approach of [16] could be used to synthesize such controllers, but it assumes a linear system model, which defeats the objective of this work. Researchers in [26] use a robust control framework to ensure robust stability. By learning a probabilistic dynamical model with GPs, they reduce the modeling uncertainty by translating the GP posterior variance into the robust control framework, and ultimately increase the closed-loop performance. However, this approach is still a norm-minimization problem with design filters. Another approach [27], [28] is to attach a NN to a LTI controller. But in this work we aim to only tune the LTI controller without requiring a NN.

This paper introduces the method outlined in Algorithm 1. To obtain formal guarantees of robust stability, we define a nominal linear system model \( G(s) \) and uncertainty bounds. Then robust stability criteria are derived from a robust control framework. Also, an initial controller \( K_\star \) is obtained from \( H_\infty \) loop shaping. Then, the controller is tuned with gradients obtained from simulated rollouts using any nominal model \( f_n(x, u) \), thereby obtaining the tuned controller \( K_t \). The dynamics \( f_n(x, u) \) can be the same as \( G(s) \), but can also be a nonlinear model. During the training, the robust stability criteria are enforced through the gradients. Finally, the tuned controller \( K_t \) can be adjusted to the actual system dynamics, thereby obtaining the final adjusted controller \( K_a \). This is done by implementing \( K_t \) on the actual system, collecting data, learning a system model \( f_t(x, u) \), and then retraining the controller from the same gradient descent method. The same robust stability criteria are enforced during this second controller tuning.

Tuning controllers from simulated rollouts was done in PILCO [29]—a successful control learning approach that inspired part of the proposed method, notably the performance gradients presented in Section III-A. PILCO is a powerful method to tune controllers for physical systems since it requires comparatively few data points to improve closed-loop performance, as we demonstrated in [30] when tuning a gearshift controller for a multi-speed transmission. However, PILCO does not explicitly enforce robust stability though constraints in the optimization problem. Instead, PILCO propagates state uncertainty during rollouts, which is used to penalize controller behaviors leading to regions of the state space with high uncertainty. Because of the robust stability criteria, our proposed method can rely on deterministic policy rollouts, which is interesting given that uncertainty propagation is not a trivial problem.

The proposed method synthesizes a controller that is already robustly stabilizing when it is first implemented on the actual system. This is not the case for some of the methods outlined above, such as [11], [12]. Moreover, the controller remains stable even in the event of a sudden change in system dynamics, given that this change remains within assumed uncertainty bounds. By design, the learning methods reviewed above all rely on the assumption that the system dynamics are fixed. This allows to learn the uncertainty bounds from data, and increase the closed-loop performance. But it leaves no formal guarantee for the system safety when a sudden change in dynamics occur, which could be a concern for certain application. On the contrary, if for a given application it is judged that the fixed uncertainty bounds are too conservative, our methods still makes it possible to learn the bounds such as in [26], and retrain the controller.

In summary, the learning-based controller synthesis method introduced in this article maintains theoretical guarantees of robust stability during learning, which other methods such as domain randomization and PILCO do not. Because the method does not suffer from the curse of dimensionality, it is well-suited for multi-input multi-output systems and for linear controllers with a large number of internal states. The key constituents of this method are the use of a robust control framework to derive robust stability criteria, and gradient-based parameter optimization from simulated deterministic policy rollouts. The robust stability criteria are employed during the gradient-based optimization to maintain robust stability. Because the method is based on simulated policy rollouts, it is well-suited for trajectory-tracking problems where the performance criteria are hard to express as design filters. Moreover, it allows to use any learned model of the system dynamics such as neural networks.
and Gaussian processes. Finally, the robust control framework guarantees robust stability for a set of dynamical systems defined with uncertainty bounds. As a result, the controller is robustly stabilizing even before data can be collected, and it remains robustly stabilizing even if the system dynamics vary within the chosen uncertainty bounds.

Section II presents the robust control framework used in this work. Section III introduces the proposed controller learning method. An example problem is presented in Section IV, namely synthesizing a controller for a lane-change maneuver with an autonomous vehicle. The results are discussed in Section V.

II. ROBUST CONTROL FRAMEWORK

This section reviews concepts of robust control—a more detailed treatment is available in [23]—and introduces the robust control framework used in Algorithm 1. In this article, the notation $\|G(s)\|_\infty$ refers to the $\infty$-norm of a linear system $G(s)$ and is defined as $\|G(s)\|_\infty := \sup_{\omega \in \mathbb{R}} \sigma(G(j\omega))$, where $\sigma(M)$ denotes the maximum singular value of a transfer matrix $M$. The space $\mathcal{RH}_\infty$ is the space of all proper and real rational stable transfer functions.

A typical robust control problem is formulated based on system models such as that of Fig. 1. The goal is to find a stabilizing controller $K(s)$ that maximizes the system performance, which means minimizing $\|T_{z_2 w_2}(s)\|_\infty$, where $T_{z_2 w_2}(s)$ is the transfer function between the inputs $w_2$ and the outputs $z_2$. Design filters are used to weigh the penalty on the various signals of $z_2$ over frequency ranges of interest. In Fig. 1, $W_u(s)$ penalizes the control signal and $W_e(s)$, the error signal. Because the actual system dynamics is not perfectly known, it is assumed to be within a set of perturbed plants $\mathcal{G}$. Thus, nominal stability is not sufficient; $K(s)$ must stabilize the system for all possible plants in $\mathcal{G}$. In Fig. 1, additive uncertainty is used to describe $\mathcal{G} := \{ G(s) + W_u(s)\Delta(s) \}$, where $\Delta(s) \in \mathcal{RH}_\infty$ with $\|\Delta(s)\|_\infty < 1$, $G(s)$ is the nominal plant dynamics, and $W_u(s)$ is a design filter bounding the uncertainty. Also commonly used are output multiplicative uncertainty, where $\mathcal{G} := \{ (I + W_m(s)\Delta(s))G(s) \}$, and input multiplicative uncertainty, where $\mathcal{G} := \{ G(s)(I + W_i(s)\Delta(s)) \}$. From the small gain theorem [23], if $\Delta(s) \in \mathcal{RH}_\infty$ and $T_{z_1 w_1}(s) \in \mathcal{RH}_\infty$, the system of Fig. 1 is robustly stabilizing if and only if

$$\|\Delta(j\omega)T_{z_1 w_1}(j\omega)\| < 1 \forall \omega.$$ And because $\|\Delta(s)\|_\infty < 1$ by design, robust stability is guaranteed if $K(s)$ stabilizes $G(s)$ and makes $\|T_{z_1 w_1}(s)\|_\infty \leq 1$.

III. CONTROLLER LEARNING METHOD

The proposed approach is outlined in Algorithm 1. It requires to define a nominal system model $G(s)$ along with uncertainty bounds such that all possible system dynamics are contained in a set $\mathcal{G}$. Using the robust control framework of Section II, this set is used to define robust stability criteria. The method also requires to define possibly nonlinear system models with the form

$$\dot{x} = f(x, u),$$

$$y = g(x),$$

with states $x$, inputs $u$, and outputs $y$. These system models are to be used during the simulated policy rollouts under the closed-loop configuration of Fig. 2. The simulations consist of tracking $N_t$ trajectories in rollouts of $T$ time steps. The while loops starting at lines 6 and 14 tune the controllers by solving the constrained optimization problem

$$\arg \min_{\phi} c_p(\phi) = \arg \min_{\phi} \sum_{n_t=1}^{N_t} \sum_{t=0}^{T} c(y_{t|t}),$$

s.t. $K(s)$ stabilizes $G(s)$,

$$\|T_{z_1 w_1}(s)\|_\infty \leq 1,$$

where $\phi$ contains the control parameters to be tuned, namely the entries of the matrices $A_K, B_K, C_K$, and $D_K$. The two constraints of Equations (6) and (7) are turned into penalties with easily computable gradients. As such, a more general cost function is

$$J(\phi) = c_p(\phi) + \xi_c c_a(\phi) + \xi_c c_a(\phi),$$

where $c_p$ is the performance cost, $c_a$ is the nominal stability penalty, and $c_r$ is the robust stability penalty. The controller parameters $\phi$ are iterated through either a simple stochastic gradient descent scheme—where $\phi_{[i+1]} \leftarrow \phi_{[i]} + \alpha \nabla f(\phi)$ and $\alpha$ is the learning rate—or any other suitable optimization algorithm. Constants $\xi_c, \xi_r \in \mathbb{R}^+$ are used to weight the penalties and help the gradient descent method. The next sections demonstrate how to obtain the gradients of the performance cost and penalties.
\begin{algorithm}
\textbf{A. Performance Gradients}

In discrete time, the system of Fig. 2 transitions with
\begin{align}
    y_{[t]} &= g(x_{[t]}), \\
    x_{[t]} &= x_{[t-1]} + T_s f(x_{[t-1]}, u_{[t-1]}), \\
    u_{[t-1]} &= C_K x_{[t-1]} + D_K (y_{d[t-1]} - y_{[t-1]}), \\
    x_{c[t-1]} &= (I + T_s A_K)x_{c[t-2]} + T_s B_K (y_{d[t-2]} - y_{[t-2]}),
\end{align}

where $T_s$ is the sampling period, $\bar{u}$ is a nominal (feedforward) control command, and $y_{d}$ is the desired output trajectory. The performance gradients are obtained by applying the chain rule on the results of the simulated policy rollout
\begin{align}
    \frac{dc_{G}(\phi)}{d\phi} &= \sum_{t=1}^{N_c} \frac{dc(y_{[t]}]}{d\phi}, \\
    \frac{dc(y_{[t]}]}{d\phi} &= \frac{dc(y_{[t]}]}{d\phi} \frac{dy_{[t]}]}{d\phi}.
\end{align}

The first term on the right-hand side of Equation (14) depends on the chosen cost function, the other term can be further expanded as
\begin{align}
    \frac{dy_{[t]}]}{d\phi} &= \frac{d g(x_{[t]}]}{d\phi} \frac{dx_{[t]}]}{d\phi}, \\
    \frac{dx_{[t]}]}{d\phi} &= \frac{dx_{[t-1]}]}{d\phi} + T_s \frac{df(x_{[t-1]}, u_{[t-1]}]}{d\phi},
\end{align}

\begin{align}
    \frac{df(x_{[t-1]}]}{d\phi} &= \frac{\partial f}{\partial x_{[t-1]}]} \frac{dx_{[t-1]}]}{d\phi} + \frac{\partial f}{\partial u_{[t-1]}]} \frac{du_{[t-1]}]}{d\phi}, \quad (17) \\
    \frac{du_{[t-1]}]}{d\phi} &= C_K \frac{dx_{[t-1]}]}{d\phi} + (I)_{ij} (x_{c[t-1]}), \quad (18) \\
    \frac{dx_{c[t-1]}]}{d\phi} &= (I + T_s A_K) \frac{dx_{c[t-2]}]}{d\phi} + T_s (I)_{ij} (x_{c[t-2]}), \quad (19)
\end{align}

\begin{align}
    \frac{dc_{G}(\phi)}{d\phi} &= \sum_{t=1}^{N_c} \frac{dc(y_{[t]}]}{d\phi}, \\
    \frac{dc(y_{[t]}]}{d\phi} &= \frac{dc(y_{[t]}]}{d\phi} \frac{dy_{[t]}]}{d\phi}.
\end{align}

\begin{align}
    \frac{dy_{[t]}]}{d\phi} &= \frac{d g(x_{[t]}]}{d\phi} \frac{dx_{[t]}]}{d\phi}, \\
    \frac{dx_{[t]}]}{d\phi} &= \frac{dx_{[t-1]}]}{d\phi} + T_s \frac{df(x_{[t-1]}, u_{[t-1]}]}{d\phi},
\end{align}

\begin{align}
    \frac{df(x_{[t-1]}]}{d\phi} &= \frac{\partial f}{\partial x_{[t-1]}]} \frac{dx_{[t-1]}]}{d\phi} + \frac{\partial f}{\partial u_{[t-1]}]} \frac{du_{[t-1]}]}{d\phi}, \quad (17) \\
    \frac{du_{[t-1]}]}{d\phi} &= C_K \frac{dx_{[t-1]}]}{d\phi} + (I)_{ij} (x_{c[t-1]}), \quad (18) \\
    \frac{dx_{c[t-1]}]}{d\phi} &= (I + T_s A_K) \frac{dx_{c[t-2]}]}{d\phi} + T_s (I)_{ij} (x_{c[t-2]}), \quad (19)
\end{align}

\begin{align}
    \frac{dc_{G}(\phi)}{d\phi} &= \sum_{t=1}^{N_c} \frac{dc(y_{[t]}]}{d\phi}, \\
    \frac{dc(y_{[t]}]}{d\phi} &= \frac{dc(y_{[t]}]}{d\phi} \frac{dy_{[t]}]}{d\phi}.
\end{align}

\begin{align}
    \frac{dy_{[t]}]}{d\phi} &= \frac{d g(x_{[t]}]}{d\phi} \frac{dx_{[t]}]}{d\phi}, \\
    \frac{dx_{[t]}]}{d\phi} &= \frac{dx_{[t-1]}]}{d\phi} + T_s \frac{df(x_{[t-1]}, u_{[t-1]}]}{d\phi},
\end{align}

\begin{align}
    \frac{df(x_{[t-1]}]}{d\phi} &= \frac{\partial f}{\partial x_{[t-1]}]} \frac{dx_{[t-1]}]}{d\phi} + \frac{\partial f}{\partial u_{[t-1]}]} \frac{du_{[t-1]}]}{d\phi}, \quad (17) \\
    \frac{du_{[t-1]}]}{d\phi} &= C_K \frac{dx_{[t-1]}]}{d\phi} + (I)_{ij} (x_{c[t-1]}), \quad (18) \\
    \frac{dx_{c[t-1]}]}{d\phi} &= (I + T_s A_K) \frac{dx_{c[t-2]}]}{d\phi} + T_s (I)_{ij} (x_{c[t-2]}), \quad (19)
\end{align}

\begin{align}
    \frac{dc_{G}(\phi)}{d\phi} &= \sum_{t=1}^{N_c} \frac{dc(y_{[t]}]}{d\phi}, \\
    \frac{dc(y_{[t]}]}{d\phi} &= \frac{dc(y_{[t]}]}{d\phi} \frac{dy_{[t]}]}{d\phi}.
\end{align}

\begin{align}
    \frac{dy_{[t]}]}{d\phi} &= \frac{d g(x_{[t]}]}{d\phi} \frac{dx_{[t]}]}{d\phi}, \\
    \frac{dx_{[t]}]}{d\phi} &= \frac{dx_{[t-1]}]}{d\phi} + T_s \frac{df(x_{[t-1]}, u_{[t-1]}]}{d\phi},
\end{align}
are all in the open left-half plane. In effect, these are the \( A \) matrices of the state space realizations of the transfer functions in Equations (21)–(24). In the learning algorithm, a penalty is applied whenever the system is unstable. The penalty for the nominal stability is therefore defined as

\[
c_n(\phi) = \sum_{i=1}^{4} \max_i \left\{ 0, \max_j \{ \Re(\lambda_j(M_i)) \} \right\},
\]

where \( \Re(\cdot) \) designates the real part of a complex number. The gradient of \( c_n \) is easily obtainable with automatic differentiation [31]. In this work, we used TensorFlow [32].

### C. Robust Stability Gradients

A state space representation for \( T_{z_1 u_1}(s) \) is obtained with the linear fractional transformation \( T_{z_1 u_1}(s) = F_L\left[P_{red}(s), K(s)\right] \), where \( P_{red}(s) \) represents \( [z_1, e]^\top = P_{red}(s)[u_1, u]^\top \), and is partitioned as follows

\[
P_{red}(s) = \begin{bmatrix} A & B_1 & B_2 \\ C_1 & D_{11} & D_{12} \\ C_2 & D_{21} & 0 \end{bmatrix}.
\]

Then,

\[
T_{z_1 u_1}(s) = \begin{bmatrix} \tilde{A} & \tilde{B} \\ \tilde{C} & \tilde{D} \end{bmatrix} = \begin{bmatrix} A + B_2 D_K C_2 & B_2 C_K \\ B_K C_2 & A_K \\ C_1 + D_{12} D_K C_2 & D_{12} C_K \\ D_{11} + D_{12} D_K D_{21} \end{bmatrix} \begin{bmatrix} B_1 + B_2 D_K D_{21} \\ B_K D_{21} \end{bmatrix},
\]

and the complex transfer matrices for the various frequencies \( \omega \) can be obtained with

\[
T_{z_1 u_1}(j\omega) = \tilde{C}(j\omega I - \tilde{A})^{-1} \tilde{B} + \tilde{D}.
\]

The robust stability criteria is \( ||T_{z_1 u_1}(s)||_{\infty} < 1 \). This can be verified by computing the maximum singular value \( \tilde{\sigma} \) of the complex transfer matrices \( T_{z_1 u_1}(j\omega) \) on a sufficiently dense frequency grid. Thus, \( c_{r}(\phi) \) becomes

\[
c_r(\phi) = \max_\omega \sup_\lambda \sigma(T_{z_1 u_1}(j\omega)) \}
\]

where \( \tilde{\sigma}(M) = \left| \tilde{\lambda}(M^* M) \right|^{1/2} \), with \( M^* \) representing the conjugate transpose of a matrix \( M \), and \( \tilde{\lambda} \) is the largest eigenvalue, which is positive real. Here again, the gradients are obtained through automatic differentiation.

### IV. EXAMPLE PROBLEM

To demonstrate the control learning method, we simulated lane-change maneuvers for autonomous vehicles [33], [34] using the bicycle model shown in Fig. 3. In the simulated scenarios, the vehicle is also accelerated during the lane-change maneuvers. The varying longitudinal velocity makes this a non-trivial problem and it motivates using a robust control approach [35], [36]. We assume a full-state feedback, so \( y = x \). The states are \( x = [\dot{x}, \dot{y}, \psi, Y, \delta]^\top \) and controls are \( u = [F_{tx}, F_{ty}, \delta_t]^\top \). The system dynamics are

\[
f(x, u) = \begin{bmatrix} (F_{tx} + F_{tx} \cos(\delta) - F_{ty} \sin(\delta) - \frac{1}{2} \rho \dot{x}^2 A t c_d) / m + \dot{y} \psi \\ (F_{ty} \sin(\delta) + F_{tx} + F_{ty} \cos(\delta)) / m - \dot{x} \psi \\ F_{tx} l_t \sin(\delta) - F_{ty} l_t + F_{ty} l_t \cos(\delta) / I_z \\ \dot{x} \sin(\psi) + \dot{y} \cos(\psi) \\ \lambda_\omega(\delta_t - \delta) \end{bmatrix}.
\]

The steering behavior is modeled as a first order linear system [37]. Assuming the tire slip angles \( \alpha_T \) and \( \alpha_R \) remain small, a linear tire model is used

\[
F_{ty} = C_T \alpha_T, \quad \alpha_T = \left( \psi l_t - \dot{y} \right) / \dot{x}, \quad (37)
\]

\[
F_{ty} = C_T \alpha_T, \quad \alpha_T = \left( \psi l_t - \dot{y} \right) / \dot{x}, \quad (38)
\]

where \( C_T \) and \( C_T \) are the front and rear tire stiffness coefficients.

#### A. Linearization & Uncertainty

The bicycle model is linearized to obtain \( G(s) \). This is done by taking the Jacobian of \( f(x, u) \) with respect to both states and input vectors, and evaluating the expression for the initial states \( x(0) = [\dot{x}_0, 0, 0, 0, 0]^\top \) and inputs \( u(0) = [0, 0, 0]^\top \), thereby obtaining the \( A_K \) and \( B_K \) matrices for the state space representation of \( G(s) \). Assuming full-state feedback, \( B_G = J \).

Input multiplicative uncertainty is used to cover the set of all possible system dynamics, where \( \mathcal{G} := \{G(s)(I + \)
$W_1(s) \Delta(s) W_2(s)$} and $\|\Delta(s)\|_\infty < 1$. To obtain the uncertainty covers $W_1(s)$ and $W_2(s)$, multiple systems are sampled based on the assumed ranges for the parameters and states presented in Table I. Then the MATLAB command `ucorecover` is used to obtain $W_1(s)$ and $W_2(s)$ such that all the sampled systems are covered by $G$.

### B. Trajectory & Nominal Command

The trajectories $x_4$ consist of a concurrent lane change and vehicle acceleration. Only the states of interest $\dot{x}$ and $Y$ are prescribed and the rest of the entries in $x_4$ are zeros. Four scenarios are used in this work; they are parameterized with the values presented in Table II. The vehicle is accelerated from $\dot{x}_0 = 25$ m/s and reaches $\dot{x}_f$ at time $t_f$. The acceleration begins at 0 m/s², then is gradually increased up to a constant value $a_{\text{max}}$, which is held for a moment until it is brought back to 0 m/s² at time $t_{f1}$. The lateral position begins at $Y_0 = 0$ m and ends at $Y_f$ at time $t_{f2}$. The trajectory of $Y$ follows a 3-4-5 polynomial where the first and second derivatives are null at both ends of the trajectory.

The nominal command $\bar{u}$ is computed in two independent steps. First, the front and rear tractive forces are computed from the reference velocity profile in $x_{40}$, where

$$ F(t) = m \ddot{x}(t) + \frac{1}{2} \rho \dot{x}(t)^2 A_f C_{f0}, \quad (39) $$

$$ F_{tx}(t) = \frac{1}{2} F(t), \quad F_{rx}(t) = \frac{3}{2} F(t). \quad (40) $$

Then, the steering input is computed. A reduced dynamical system with $A_R$ and $B_R$ is obtained by only considering the lateral dynamics in $G(s)$, i.e., only considering the states $x = [\dot{y}, \dot{\psi}, \dot{y}, \dot{Y}, 0]^\top$ and controls $u = [\delta_r]$. Therefore, $A_R$ consists of the last five columns and five rows of $A_G$, and $B_R$ consists of the last five elements in the last column of $B_G$. The resulting system is still controllable. The nominal steering input is then computed with

$$ \delta_r(t) = B_R^T e^{A_R^T (t_f - t)} W_c^{-1} (t_f) [0, 0, Y_f, 0, 0]^\top, \quad (41) $$

where $W_c(t)$ is the Controllability Gramian.

### C. Controller Structure and Initialization

The feedback controller is structured to decouple the control of the longitudinal and lateral dynamics. The front $F_{tx}$ and rear $F_{rx}$ tractive forces are only used for the feedback control of the vehicle velocity $\dot{x}$, while the steering input $\delta_r$ is only used for controlling the lateral dynamics. As will become apparent in the result section, a simple integrator is sufficient for the vehicle speed control. Moreover, the tracking error is so negligible that no further controller tuning is required for this integrator.

The control of the lateral dynamics is more difficult, however. An initial controller is obtained with $\mathcal{H}_\infty$ loop shaping. For that, a nominal linear model $G(s)$ is built from the same $A_R$ and $B_R$ matrices introduced in the previous section. Then, a controller that stabilizes $G(s)$ and minimizes the tracking error is obtained using the MATLAB command `hinfsyn`. This $\mathcal{H}_\infty$ controller is assembled with the simple integrator obtained previously to form the complete initial feedback controller $K_i$, which has 24 internal states, and has $D_K = 0$. Together the $A_K$, $B_K$, and $C_K$ matrices of $K_i$ contain 792 entries.

### D. Controller Training With Nominal Model

The controller $K_i$ is tuned with simulated rollouts, see line 6 in Algorithm 1. The system model $f_n(x, u)$ is that of Equation (36) with the nominal parameters shown in Table I. The simulations have a time step of $T_s = 0.02$ s. The cost function used is

$$ c(x[t]) = (x[t] - x_{d[t]})^\top Q (x[t] - x_{d[t]}), \quad (42) $$

where $Q$ is a diagonal matrix weighting the relative importance of the state errors.

Only the first three scenarios of Table II are used during training. Scenario 4 is reserved for testing that the controller also does well on a scenario never seen during training.

### E. Controller Training With Learned Model

The controller $K_s$ is also tuned from simulated policy rollouts, but with a learned model of the system dynamics this time, see line 14 in Algorithm 1. In this example problem, the actual (unknown) system has an altered tire stiffness coefficient of $C_f = 40$ kN/rad. We chose to learn the dynamics using a model that has the form

$$ f_1 = f_n + [g_1(z), g_2(z), g_3(z), 0, 0, 0]^\top \quad (43) $$

where $g_d(z)$ are functions that represent the unknown dynamics and $z = [\alpha_{\dot{r}}, \alpha_r]^\top$ is the feature vector used in the prediction models. To learn $f_1$, first a lane change is simulated for Scenario 1 with $K_i$ (line 12 in Algorithm 1), and a dataset $D = \{x_{d[t]}, u_{d[t]}\}$ composed of $n_p + 1$ data points is collected. Then Gaussian
processes are used to learn the difference between the nominal system dynamics $f_n$ and the actual dynamics. In other words, the targets to be learned are issued from the prediction error

$$e_p[i] = x[i+1] - f_n(x[i], u[i]).$$

(44)

More specifically, each of the three dimensions to be learned has its own target vector $t_d \in \mathbb{R}^{n \times d}$, which are composed of the $d$-th component of the error vectors, such that

$$t_d = \left([e_p[1]]_d, \ldots, [e_p[n_p]]_d\right)^T.$$

(45)

All three dimensions share the same set of feature vectors $\{z_i\}$. The Gaussian processes used in this work has a zero mean function $m(z) := 0$ and a square exponential kernel function

$$k(z_i, z_j) := \sigma_f^2 \exp\left(-\frac{1}{2} (z_i - z_j)^\top \Lambda^{-1} (z_i - z_j)\right),$$

(46)

where $\sigma_f^2$ is the signal variance, and $\Lambda = \text{diag}(\lambda_1^2, \lambda_2^2)$ is a diagonal matrix composed of the characteristics length-scales of the two dimensions in $z$. These are the hyper-parameters of the Gaussian process, and every function $g_d$ has its own set of hyper-parameters. The predictive functions $g_d(x)$ are simply the mean of the Gaussian processes evaluated at the test inputs $z$,

$$g_d(z) = \mu_d(z) = K_{z,x}(K_{z,z} + \sigma_f^2 \Lambda)^{-1} t_d,$$

(47)

where $\sigma_f^2$ is the noise variance, $K_{zz} = [k(z_i, z_j)]_{ij}$, and $K_{zx} = [k(z_i, z_j)]_j$. The hyper-parameters, symbolically regrouped in the vector $\theta$, can be tuned by maximizing the logarithm of the marginal likelihood of the observed data points in $D$, as suggested in [8], with

$$\log p(t_d|D, \theta) = -\frac{1}{2} t_d^\top K^{-1} t_d - \frac{1}{2} \log |K| - \frac{n}{2} \log 2\pi,$$

(48)

where $K = K_{zz} + \sigma_f^2 \Lambda$, and $|K|$ is the determinant of $K$. The likelihood is maximized with a gradient-based optimization algorithm, where $\alpha = K^{-1} t_d$,

$$\frac{\partial}{\partial \theta_j} \log p(t_d|D, \theta) = \frac{1}{2} \text{tr}\left((\alpha \alpha^\top - K^{-1}) \frac{\partial K}{\partial \theta_j}\right).$$

(49)

V. RESULTS & DISCUSSION

Fig. 4 shows that the tracking error is negligible for the vehicle speed $\dot{x}$, but not for the lateral position $Y$. Therefore, the result section only discusses the tracking of the lateral position.

Fig. 4(b) shows that the controller tuned from simulated policy rollouts ($K_t$) improves the tracking performance when compared to the initial controller ($K_i$) obtained from $\mathcal{H}_\infty$ loop-shaping. Fig. 5(a) shows the tracking error $e_Y$ on the lateral position for this run, and Table III presents the $L_2$-norm of this error signal. The tuning method also reduces the tracking error for a lane-change scenario that was not used during training. In effect, both Fig. 5(b) and Table III show that $K_t$ does better than $K_i$ for Scenario 4. This suggests that the tuning method does not overfit on the training scenarios, and can generalize to others. From experience, this was not the case when a single scenario was used for training. Finally, the proposed method keeps the tuned controller $K_t$ robustly stabilizing, which can be seen in Fig. 6.

The tuning of $K_t$ requires 4000 training epochs. The first 2000 epochs are done with simulated rollouts of 8 s duration (simulated time), and the last 2000 epochs, with 40 s rollouts. The longer rollouts are used to ensure that the low-frequency dynamics are also considered during the controller tuning. The total training time for the first 2000 epochs is 1083 s, and for the last 2000 epochs, 5245 s. Fig. 7 shows the evolution of the performance cost $c_p$ in (a), the maximum real part of all eigenvalues in the matrices $M_1$ to $M_4$ in (b), and the maximum singular value of $T_{z w}$ in (c). This indicates that the penalties $\xi_c = 10,000$ and $\xi_r = 1000$. For smaller values of $\xi_c$ and $\xi_r$, however, the penalties were not strong enough
to enforce the stability constraints. The gradient descend algorithm used during training is Adam [38], which helps stabilize the gradients compared to a simple stochastic gradient descent algorithm.

Fig. 8 and Table III show that the tracking error of $K_t$ increases for policy rollouts on the actual dynamics, i.e., when $C_l = C_{\ell} = 40 \text{ kN/rad}$. However, the controller remains stable, as the actual dynamics are still within the assumed uncertainty bounds. This allows to bring back the performance to a similar level by re-tuning the controller with a learned model. As a result, the adjusted controller $K_a$ reduces the tracking error, even for the unseen Scenario 4. Training with the learned model is longer, but still reasonable as the same 2000 epochs of 8 s rollouts takes 2071 s of training time, which is approximately twice the time for the nominal model.

A. Augmented Cost

It is possible to augment the total cost $J(\phi)$ of Equation (8) with $c_t(\phi) = \|T_{z_2 w_2}(s)\|_{\infty}$ to help shape the closed-loop behavior with frequency-based specifications. The gradients $\nabla_{\phi} c_t(\phi)$ can be computed with the method of Section III-C used to obtain $\nabla_{\phi} c_f(\phi)$, but with $T_{z_2 w_2}(j\omega)$ instead. This additional cost $c_t$ could be used in problems where scenario-based tuning alone is inadequate. One such example could be a controller that must account for a large frequency range. In effect, if $T_\phi$ is small enough to allow the simulation of high-frequency dynamics, it also means that the policy rollouts must have a long time span to also include the effects of low-frequency dynamics. Therefore, the cost $c_t$ could be used to penalize the low-frequency errors, while the scenario-based performance cost $c_p$ tunes the high-frequency behavior. In the lane-change problem however, this was found unnecessary. As previously discussed, it was sufficient to simulate 40 s rollouts.

As a final remark, the nominal model $G(s)$ is not learned in the proposed method, so the additional cost $c_t$ could be less beneficial if the actual system deviates significantly from $G(s)$.

VI. CONCLUSION

This article presents a method to improve the performance of a LTI controller from a learned nonlinear model, while preserving robust stability using a robust control framework. The controller parameters are iterated with gradients obtained from simulated rollouts with the learned model. The robust stability criteria are enforced through the gradients as well, penalizing only unsafe controllers. The method yields a tuned controller that is already robustly stabilizing when implemented on the actual system for the first time. It also remains stable when sudden changes in the system dynamics occur, given that they are within assumed uncertainty bounds. This was demonstrated by simulating a bicycle model performing lane-change maneuvers. The approach successfully learns a robustly stabilizing controller, and thereby takes advantage of modern machine learning for synthesizing LTI controllers.

REFERENCES

[1] B. Boulet and Y. Duan, “The fundamental tradeoff between performance and robustness - a new perspective on loop shaping-classic control revisited part II,” IEEE Control Syst., vol. 27, no. 3, pp. 30–44, Jun. 2007.
[2] A. Amini et al., “Learning robust control policies for end-to-end autonomous driving from data-driven simulation,” IEEE Robot. Automat. Lett., vol. 5, no. 2, pp. 1143–1150, Apr. 2020.
[3] Y. Lin, J. McPhee, and N. L. Azad, “Comparison of deep reinforcement learning and model predictive control for adaptive cruise control,” IEEE Trans. Intell. Veh., vol. 6, no. 2, pp. 221–231, Jun. 2021.
[4] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, “Domain randomization for transferring deep neural networks from simulation to the real world,” in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., Vancouver, Canada, 2017, pp. 23–30.
[5] M. Andrychowicz et al., “Learning dexterous in-hand manipulation,” Int. J. Robot. Res., vol. 39, no. 1, pp. 3–20, 2020, doi: 10.1177/02787434176919887447.
[6] J. F. Fisac, A. K. Akametalu, M. N. Zeilinger, S. Kaynama, J. Gillula, and C. J. Tomlin, “A general safety framework for learning-based control in uncertain robotic systems,” IEEE Trans. Autom. Control, vol. 64, no. 7, pp. 2737–2752, Jul. 2019.
[7] M. Chen and C. J. Tomlin, “Hamilton-Jacobi reachability: Some recent theoretical advances and applications in unmanned airspace management,” Ann. Rev. Control Robot., Auton. Syst., vol. 1, no. 1, pp. 333–358, 2018, doi: 10.1146/annurev-control-060117-104941.
[8] C. E. Rasmussen and C. K. I. Williams, Gaussian Processes for Machine Learning, Cambridge, MA, USA: MIT Press, 2006.
[9] M. Liu, G. Chowdhary, B. Castra da Silva, S.-Y. Liu, and J. P. How, “Gaussian processes for learning and control: A tutorial with examples,” IEEE Control Syst., vol. 38, no. 5, pp. 53–86, Oct. 2018.
[10] F. Berkenkamp, M. Turchetta, A. P. Schoellig, and A. Krause, “Safe model-based reinforcement learning with stability guarantees,” in Proc. 31st Int. Conf. Neural Inf. Process. Syst., Long Beach, CA, USA, 2017, pp. 906–919.
[11] A. J. Taylor, V. D. Borobanu, H. M. Le, Y. Yue, and A. D. Ames, “Episodic learning with control Lyapunov functions for uncertain robotic systems,” in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., Macau, China, 2019, pp. 6878–6884.
Marc-Antoine Beaudoin received the Bachelor of Engineering degree in mechanical engineering from the Université de Sherbrooke, Sherbrooke, QC, Canada, in 2015, and the Master of Applied Science degree in mechanical engineering from the University of Toronto, Toronto, ON, Canada, in 2018. He is currently working toward the Ph.D. degree in electrical and computer engineering with the Intelligent Automation Laboratory, Centre for Intelligent Machines, McGill University, Montreal, QC, Canada. Prior to his doctoral studies, he was a practicing Engineer with the Electric Vehicle Industry. His research interests include the machine learning control of dynamical systems, with a focus on electric and autonomous vehicles.

Benoit Boulet (Senior Member, IEEE) received the bachelor’s degree in applied sciences from Université Laval, Quebec, QC, Canada, in 1990, the Master of Engineering degree in electrical engineering from McGill University, Montreal, QC, Canada, in 1992, and the Ph.D. degree in electrical engineering from the University of Toronto, Toronto, ON, Canada, in 1996. In 1998, he joined McGill University, where he is currently a Professor with the Department of Electrical and Computer Engineering, and the Director of the Technological Innovation and Entrepreneurship Centre, McGill Engine. He is also an Associate Vice-Principal of McGill Innovation and Partnerships. From 2014 to 2020, he was an Associate Dean (Research and Innovation) with the Faculty of Engineering. He is the former Director and current member of the McGill Centre for Intelligent Machines, where he heads the Intelligent Automation Laboratory. His research interests include the design and data-driven control of electric vehicles and renewable energy systems, machine learning applied to biomedical systems, and robust industrial control.