# Closing the Sim-to-Real Gap: Addressing Parametric Uncertainty Using Adaptive Control and Reinforcement Learning

Timothy Chen
*Dept. of Aeronautics and Astronautics*
*Stanford University*
Stanford, USA
chengine@stanford.edu

***Abstract*—A novel algorithm RL-aCLF is constructed to provide nominal stability certificates when porting controllers from learned dynamics in simulation to real dynamics with parametric uncertainty. The algorithm leverages stability guarantees from the aCLF framework and exploits reinforcement learning to learn basis functions that span the dynamics space. Initial results are provided to verify the stability claims in simple systems.**

## I. Introduction

Simulations often do not truly reflect real-world dynamical systems. They may simplify very complex dynamics (e.g. linearization) or make some assumptions and approximations. In any case, results generated through simulation do not truly reflect those when implemented in reality. This is called the sim-to-real gap.

Modern control methods to generating stable controllers have historically been utilized. And while a great deal of them have some sort of analysis or convergence proof tied to them, many are analyzed in continuous time and therefore, not respecting hardware constraints. Moreover, model mismatch or noise in the system may cause controllers to lose a level(s) of stability. While this paper will not explore controllers in discrete time, it will leverage adaptive methods to address uncertain dynamics of a system.

Recently, the field of reinforcement learning has ushered in learning-based controllers heavily situated in simulation in order to learn the dynamics of a potentially uncertain system. However, a vast majority lack any sort of convergence certificate other than verification, and controllers simulated on one system in simulation results in poor performance on the real-world system in the presence of model uncertainty/noise.

The goal here is to create a unified adaptive and reinforcement learning-based controller that is able to converge to some desired position or trajectory without any working model of the underlying dynamics. The benefits are that reinforcement learning can generate optimal controllers in the sense that it will try to minimize some cost, while adaptive methods still maintain some stability certificate. Our proposed method will be seek to marry the two fields.

Section II will explore the construction of the controller from a Control Lyapunov perspective. Section III will briefly introduce the DDPG reinforcement learning algorithm and integration. Section IV will present the initial findings.

## II. Construction

This section will explore the construction of our proposed controller and system, along with stability certificates.

### A. Control Lyapunov Functions

Normally, a general dynamical system is presented in a control affine form:

$$\dot{x} = f(x) + g(x)u \tag{1}$$

and a controller is constructed around this nonlinear system. However, because the dynamics are unknown to us, we need to maintain an estimate of the system.

We draw inspiration from neural networks that the reinforcement learning algorithm utilizes in order to construct the dynamical system differently. Like with neural networks, we try to approximate the underlying, unknown dynamics through basis functions. If the neural network can eventually form basis functions that can model the simulated dynamical system, then we can leverage specifically the $n-1^{th}$ neurons as said basis functions for our estimated system and controller. We specifically choose the $n-1^{th}$ layer because the output (whether it be an action or state) is a linear mapping from these basis functions. The claim is that, while the adaptive controller seeks to create a controller to stabilize the current estimated system using a set of basis functions, the reinforcement learning algorithm will incrementally improve these functions such that they can be used to approximate both the underlying dynamics and the control input through different sets of weights.

We pose the system as:

$$\dot{x} = W_{dyn}\phi(x) + g(x)u_{ff} + g(x)u \tag{2}$$

$$u = \kappa(x, W_{dyn}) \tag{3}$$

where $x \in R^n, W_{dyn} \in R^{n \times m}, u \in R^n, \phi(x) \in R^m$. The number of neurons in the $n-1^{th}$ layer is equal to $m$. $u_{ff}$ is the feedfoward control action proposed by the reinforcement

learning algorithm, while $u$ is the control action of the adaptive method. We then propose a controller of the form:

$$u = W\phi(x) \tag{4}$$

One possible way to derive a suitable set of W is to use feedback linearization on a suitable output. For simplicity (relative degree 1), we seek to drive only the output $\eta = x_i$ to zero asymptotically. $\dot{x}_i$ is a parameter we have control over in order to find a controller that enforces our choice in parameter.

$$\dot{\eta} = \dot{x} = W_{dyn}\phi(x) + g(x)u_{ff} + g(x)u = \nu \tag{5}$$

We seek to drive $x_i$ to 0 exponentially quickly, so our choice is $\nu = -\alpha\eta$ for any $\alpha > 0$.Therefore, our stabilizing controller is of the form:

$$u = g(x)^{-1}(-\alpha x_i - g(x)u_{ff} - W_{dyn}\phi(x)) \tag{6}$$

where $g(x)^{-1}$ is a pseudo-inverse.

Feedback linearization brings about more sophisticated ways to control the unknown system. With the actuation matrix $g(x)$ known, we can calculate the relative degree for simple systems and outputs, and therefore devise a feedback linearization controller without having to know any of the true dynamics. Assuming we want to control a component of the state, as long as the relative degree $\gamma^i$ for that output is less than or equal to the highest derivative of that component that appears in the state, then we can derive its relative degree using only g(x).

$$\exists j \; s.t. \nabla_x x_i^{(j)} g(x) = v \neq 0 \; and \; \nabla_x x_i^{(k)} g(x) = 0 \; \forall \; 1 \leq k < j \tag{7}$$

where $x_i^{(j)}$ is the $j^{th}$ derivative of $x_i$. The above system would have relative degree $j + 1$.

Then, we can transform our system to a linear output system.

$$\begin{aligned} \dot{\eta} &= F\eta + Gv \\ \eta &= [x_i, x_i^1, ..., x_i^j]^T \\ F &= \begin{bmatrix} 0 & 1 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & 0 & 1 & \ddots & & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & & \ddots & 0 & 0 & 1 \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & 0 & 0 \end{bmatrix} \end{aligned} \tag{8}$$

$$G = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \tag{9}$$

A $v$ of the form $v = K\eta$ that yields $A_{cl} = F + GK$ will form the linear stable output system $\dot{\eta} = A_{cl}\eta$, assuming $A_{cl}$ is stable. The above procedure can be extended to multiple desired outputs by stacking the outputs and its derivatives, and generating $F$ and $G$ accordingly.

We can create a quadratic Lyapunov function to prove exponential stability of our outputs. We need to solve the Continuous Time Lyapunov Equation such that $PA_{cl} + A_{cl}^T P + Q = 0$. Q is positive definite and $A_{cl}$ is stable. Then, our Lyapunov function is:

$$V(x) = \frac{1}{2}\eta^T P \eta \tag{10}$$

where P is symmetric positive definite.

$$\dot{V} = \eta^T P\dot{\eta} = L_F V + L_G V v \leq -\lambda V \leq -\frac{min(\lambda(Q))}{max(\lambda(P))} V < 0 \tag{11}$$

where $L_F V, L_G V$ are Lie derivatives of V. $\lambda$ is constrained to be greater than or equal to $\frac{min(\lambda(Q))}{max(\lambda(P))}$ by the Continuous Time Lyapunov Equation used to generate P from $A_{cl}$. However, we may want to optimize our control effort to our system. Thereby, we can also propose an optimization based controller:

$$\begin{aligned} &minimize \; v^T v \\ &subject \; to \; L_F V + L_G V v \leq -\lambda V \end{aligned} \tag{12}$$

This optimization has a closed form solution, the min-norm controller:

$$v(x) = \begin{cases} -\frac{a(x)b(x)^T}{b(x)^T b(x)} & a(x) < 0 \\ 0 & otherwise \end{cases} \tag{13}$$

$$\begin{aligned} a(x) &= L_F V + \lambda V \\ b(x) &= L_G V \end{aligned} \tag{14}$$

Convergence of this controller is guaranteed. By plugging (13) into (11), we find the inequality producing exponential stability is satisfied. If the dynamics are already stable, then the controller will do nothing; otherwise, the controller utilizes feedback linearization.There are other analytical controllers solving the scalar constrained optimization problem such as Sontag's universal controller.

To transform $v$ to $u$, we undo the output transformation.

$$\begin{aligned} x_i^{(j+1)} = \mathcal{F}(x) + \mathcal{G}(x)u &= v \\ u &= \mathcal{G}^{-1}(v - \mathcal{F}) \end{aligned} \tag{15}$$

### *B. Adapting CLFs to basis functions*

Because we want to frame our control input to the system as the output of a neural network, we can also treat the neural network weights as a control variable. To do so, we must treat the matrix of weights as a vector, and the basis functions as a matrix.

$$u(x) = W\phi(x) = \begin{bmatrix} \phi(x)^T & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \phi(x)^T \end{bmatrix} \begin{bmatrix} w_{11} \\ w_{12} \\ . \\ . \\ . \\ w_{ij} \\ . \\ . \\ . \\ w_{nm} \end{bmatrix} := [\phi(x)]w \tag{16}$$

We rewrite (15) in accordance with our dynamical system.

$$x_i^{(j+1)} = \nabla_x x^{(j)}(W_{dyn} * \phi(x) + g(x)u_{ff} + g(x)u) = v \tag{17}$$

(12) can be rewritten as follows:

$$\begin{aligned} &minimize\ ||[\phi]w||^2 \\ &subject\ to\ L_F V + L_G V(\mathcal{F} + \mathcal{G}[\phi]w) \leq -\lambda V \end{aligned} \tag{18}$$

Of course, other controllers using different objective functions may also be used.

Thus far, we have assumed that we knew our dynamical weight matrix $W_{dyn}$. However, we would like our adaptive controller to learn these weights by using the min-norm controller to drive the output to desired. We leverage adaptive control Lyapunov functions to accomplish this task. We frame (2) as a parametric uncertainty problem:

$$\dot{x} = [\phi(x)]w^*_{dyn} + g(x)u_{ff} + g(x)u \tag{19}$$

where $w^*_{dyn}$ is now a vector of the system dynamic weights and the star denotes the true weights of the system. We assume $[\phi(x)]$ is locally Lipschitz continuous and that $[\phi(0)] = 0$, so that the origin is an equilibrium point. Because we do not have access to the true dynamic weights, we work with the system using an estimate of the parameters $\hat{w}_{dyn}$:

$$\dot{x} = [\phi(x)]\hat{w}_{dyn} + g(x)u_{ff} + g(x)u \tag{20}$$

We propose the following Lyapunov candidate and extract the parameter update equation from the convergence proof. Consider:

$$V(x, \hat{w}_{dyn}) = V_a(x, \hat{w}_{dyn}) + \frac{1}{2}\tilde{w}^T_{dyn}\Sigma^{-1}\tilde{w}_{dyn} \tag{21}$$

$V_a$ is an aCLF, and we define $\tilde{w}_{dyn} = w^*_{dyn} - \hat{w}_{dyn}$. $\Sigma$ is a symmetric positive definite matrix. We can then use our Lyapunov function in (10) as a valid aCLF $V_a = V_x$, as it is independent of the parameters.

$$\begin{aligned} &\dot{V}(x, \hat{w}_{dyn}) = \\ &\frac{\partial V_x}{\partial x}(x)([\phi(x)]w^*_{dyn} + g(x)u_{ff} + g(x)u(x, \hat{w}_{dyn})) \\ &- \tilde{w}^T_{dyn}\Sigma^{-1}\dot{\hat{w}}_{dyn} \\ &= \frac{\partial V_x}{\partial x}(x)([\phi(x)]\hat{w}_{dyn} + g(x)u_{ff} + g(x)u(x, \hat{w}_{dyn})) \\ &+ \frac{\partial V_x}{\partial x}(x)[\phi(x)]\tilde{w}_{dyn} - \tilde{w}^T_{dyn}\Sigma^{-1}\dot{\hat{w}}_{dyn} \\ &= \frac{\partial V_x}{\partial x}(x)([\phi(x)]\hat{w}_{dyn} + g(x)u_{ff} + g(x)u(x, \hat{w}_{dyn})) \\ &+ (\frac{\partial V_x}{\partial x}(x)[\phi(x)] - \dot{\hat{w}}^T_{dyn}\Sigma^{-1})\tilde{w}_{dyn} \\ &\leq -\lambda V_x(x) \end{aligned} \tag{22}$$

As a result, we still have a globally asymptotically stable controller and retrieve the parameter update law:

$$\dot{\hat{w}}_{dyn} = \Sigma(\frac{\partial V_x}{\partial x}(x)[\phi(x)])^T \tag{23}$$

## III. Reinforcement Learning

We choose DDPG (Deep Deterministic Policy Gradient), an actor-critic method, as our choice of deep RL algorithm to iteratively change our basis functions $\phi(x)$ to better approximate the real dynamics. DDPG is an off-policy algorithm.

DDPG consists of four neural networks and two sets of actor and critics. One set proposes an action to an agent, while the other are target networks. The weights of the proposal networks are soft-updated to the target networks. In this way, the target networks change gradually and prevents the proposal networks from changing erratically through the difference between the proposal and target's state-action value.

Because the proposal actor is usually the network that proposes the action, $u_{ff}$ will be the feedforward action proposed by the proposal actor. Moreover, $\phi(x)$ will be the $n - 1^{th}$ layer of the proposal actor. The aCLF controller will then generate outputs to the agent, relying on the basis functions to be sufficient in describing the dynamics space.

**Algorithm 1:** RL-aCLF

```
initialize: Neural Network, ŵ_dyn;
while best score ≤ Threshold do
    while Episode not converged or expired do
        1. Get state;
        2. Get u_ff and φ(x) from proposal networks;
        if Step(action) then
            a. aCLF generates control input u;
            b. Propagate dynamics;
            c. reward = - cost;
            d. score += reward;
        end
    end
    Do Policy Gradient;
    if score ≥ best score then
        best score = score
    end
end
```

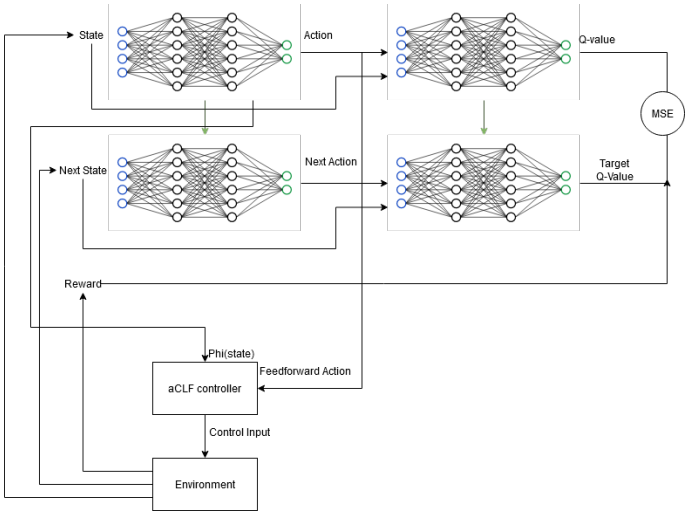


Fig. 1. Flowchart of Deep RL-aCLF. Green arrows represent soft updated weights.

Once the model has converged on a valid set of basis functions for the unknown dynamical system, the system is then implementable on real hardware. After simulation, no gradient steps are taken on the RL networks. Assuming that the dynamics of the real-world system is not too far from the simulated system that the basis functions cannot span the real dynamics, the aCLF controller will then update the estimated weights $\hat{W}_{dyn}$ such that system asymptotically converges.

## IV. Results

In this section, we present initial findings with the proposed algorithm and compare it against a baseline.

### A. Baseline

We use pendulum dynamics to verify our algorithm.

$$\dot{\chi} = \begin{pmatrix} \dot{\theta} \\ -\frac{g}{l} sin(\theta) \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} u \tag{24}$$

Our baseline will use the nominal dynamics as the basis functions. Therefore,

$$\phi(x) = \begin{pmatrix} \dot{\theta} \\ sin(\theta) \end{pmatrix} \tag{25}$$

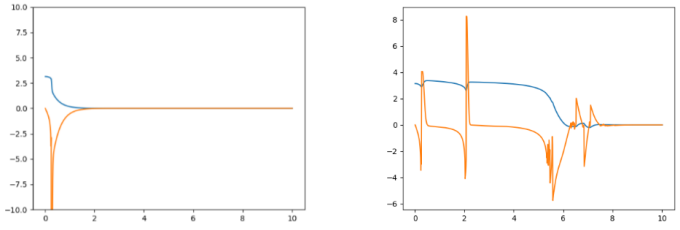

Fig. 2. Convergence of nominal dynamics-basis functions on down (left) and inverted (right pendulum. Blue is $\theta$ and orange is $\dot{\theta}$.

The simulation (Fig. 2) was performed with $\Sigma = 100I_4$, $g = 1$, $l = 1$, $x_0 = [\pi, 0]^T$, $A_{cl} = [0\ 1;\ -1\ \ -2]$, $Q = I_2$. Using the same parameters, we used the same basis functions for the inverted pendulum $g = -1$ (Fig. 3).

We then compare it to the best models generated by our RL-aCLF controller. Figure 3 illustrates the trained policy stabilizing the pendulum on their respective systems. Figure

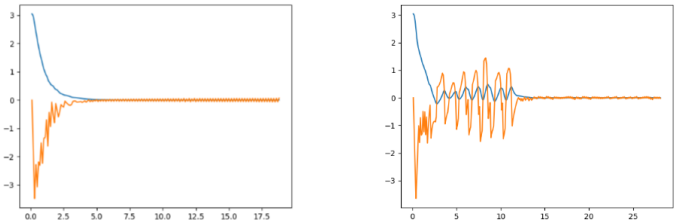

Fig. 3. RL-aCLF during training of down pendulum (left, $g = 9.81$) and inverted pendulum (right, $g = -9.81$). $\theta$ is plotted in orange, angular rate in blue.

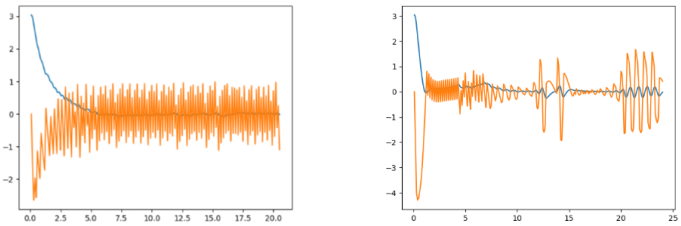

Fig. 4. RL-aCLF policy tested on the opposite pendulum system with $g = \pm 9.81$. $\theta$ on the left is stabilized around 0.

Figure 4 validates our algorithm, as it can handle out-of-distribution systems, parametrically different than the system used to train the policy. We compare performance to DDPG trained on the inverted pendulum, but tested on the down pendulum. In Figure 5, it is apparent that DDPG is unable to stabilize the new system.

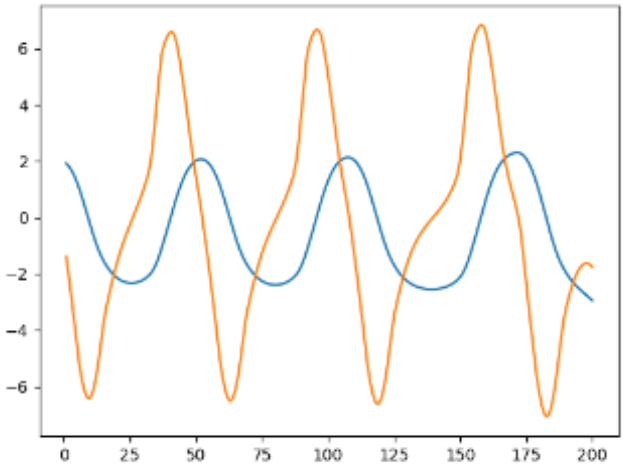

Fig. 5. DDPG policy trained on inverted pendulum but tested on down pendulum.

### B. Drawbacks

The algorithm in its current form presents a few implementation challenges. Due to the nature of the CLF structure based in optimization, the control input is not guaranteed smooth. Therefore, simulations may struggle to propagate non-smooth dynamics. All simulations performed utilized a zero-order hold on the control input to the agent, operating at 100 Hz.

Intuitively, scaling down the basis functions in the dynamics (20) should not impact learning by the RL algorithm nor are any of the assumptions of the algorithm violated since scaled basis are still basis. However, the generated control input by the aCLF tends to grow very large such that the simulation cannot continue; this is true even in the baseline, but the scaling factor required for convergence is much lower (0.1 vs $10^4$).

## V. Conclusion

RL-aCLF holds much potential in being a nominally stable controller. However, much can be improved in the learning of basis functions. Instead of a LQR reward structure, RL may learn better by minimizing the error between the estimated trajectory (20) and the real evolution of the system. In the algorithm's current form, the actuation matrix must be known. However, as a brief experiment with the baseline, changing $g(x)$ did not change the convergence. Another direction would be to explore whether RL-aCLF can stabilize a larger class of uncertain systems, since the basis functions span the whole dynamics rather than just subsets like in traditional aCLF formulations. Safety and control limits are also readily implementable in the CLF framework, such that the RL can learn safely and within hardware constraints. Finally, we intend to also backpropagate through the convex program and explore other RL algorithms to improve learning stability.

In this paper, a novel algorithm RL-aCLF is introduced to learn unknown dynamics in simulation in the way model-free RL does, but still maintain some semblance of convergence when ported to real hardware assuming only parametric uncertainty between the real and simulated agent. The nominal stability certificates of the algorithm are explored, as well as its integration with DDPG, though any model-free deep RL algorithm should suffice. Initial results were given for the down and inverted pendulum demonstrating initial success in the algorithm.

## Acknowledgments

Many thanks to Preston Culbertson and Professor Mac Schwager for their valuable input in developing this algorithm.

## References

[1] L. Wang, A.D. Ames, M. Egerstedt, "Safety Barrier Certificates for Collision-Free Multirobot Systems," IEEE Transactions on Robotics, Vol. 33, No. 3, June 2017.

[2] A.J. Taylor, A.D. Ames, "Adaptive Safety with Control Barrier Functions," arXiv preprint arXiv:1910.00555v1, 2019.

[3] R. Cheng, G. Orosz, R.M. Murray, J.W. Burdick, "End-to-End Safe Reinforcement Learning through Barrier Functions for Safety-Critical Continuous Control Tasks," Thirty-Third AAAI Conference on Artificial Intelligence, 2019.

[4] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T.Erez, Y. Tassa, D. Silver, D. Wierstra, "Continuous Control with Deep Reinfrocement Learning," ICLR, 2016.