Appendices to the Verified Path Following Using Neural Control Lyapunov Functions paper

Appendix A: CLF Generation Details

We briefly describe the neural network based learning of CLFs presented in Chang et al. [1]. In general, finding V is a hard problem as it requires a search in function space that satisfies the constrained optimization problem in Definition 1. There are many techniques to accomplish this, frequently with the first step being to select a basis set and a parametrization for V with a set of learning parameters θ. The basis functions may be, e.g., monomials or polynomials; recently, neural networks [1] have also been designed for this task. We adopt the latter basis set for our framework, i.e. a neural network, to meet the Lyapunov conditions after a *learning* phase. The learning process stochastically updates θ to increase the *likelihood* of satisfying the Lyapunov conditions. These conditions can be translated into a cost function, called the "Lyapunov risk," which measures the extent to which the current candidate CLF violates the Lyapunov conditions. Generally the problem of designing a CLF is one of finding a function V that satisfies these conditions by minimizing the cost:

$$\inf_{\theta} \sup_{x \in \mathcal{D} \setminus \mathcal{I}} \left(\max(0, -V_\theta(x)) + \inf_{u \in \mathcal{U}} \max(0, \nabla V_\theta(x) \cdot f(x,u)) + V_\theta^2(0) \right). \tag{1}$$

However, minimizing Eq. (1) is intractable in general as it requires computation of the maximum over an infinite set $\mathcal{D} \setminus \mathcal{I}$. Moreover, for each state x, one has to solve a subproblem to find the optimal u minimizing $\nabla V(x) \cdot f(x,u)$. Therefore, we consider a relaxation of Eq. (1) making the problem tractable. Namely, we sample N points $x_1, \ldots, x_N$ in the Lyapunov region and compute the empirical risk:

$$L_N(\theta) = \frac{1}{N} \sum_{i=1}^{N} (\max(0, -V_\theta(x_i)) + \max(0, \nabla V_\theta(x_i) \cdot f(x_i, u_i)) + V_\theta^2(0)), \tag{2}$$

In the above, u_i is a control input associated to the sampled state x_i and is independent of θ.

However, minimizing Eq. (2) might not be sufficient to obtain a CLF, as the Lyapunov conditions may be violated at states that were not sampled. Therefore, it must *verified* through a process which we will describe below. This results in a learning loop, where a candidate CLF is generated by a neural network, and then is verified by a Verifier. If the candidate CLF is valid in the Lyapunov region, the learning loop ends and returns the candidate CLF which is a valid CLF. If the verification fails, a counterexample is returned for which the CLF conditions fail. We then sample some number of points around the counterexample and append these points to the set of sampled states. The controls are calculated for the updated set of sampled states and the updated state-control pairs are used to train the neural network until the next pass into the Verifier.

SMT Verification of Candidate CLF It is common for CLF generation algorithms to implement a verification step and there are a number of solvers able to verify a candidate CLF. SDP [2], SOS [3, 4], SMT and most recently MIP solvers [5] solvers have all been used with some success to verify CLFs. For our framework, an SMT solver is an effective Verifier due to the limited state space and simplicity of implementation. We use the DREAL4 solver [6] to provide a precise verification solution. This solver relaxes the standard SMT decision problem to a δ-decision problem. On a given SMT problem φ, one of the following answers is provided:

unsat: φ is unsatisfiable OR δ**-sat:** φ^δ is satisfiable,

where φ^δ is called the δ-perturbation (or δ-weakening) of φ. If an SMT problem is deemed **unsat** by the δ-complete algorithm then it is guaranteed to not have any solution. This characteristic makes it an appropriate algorithm for CLF counterexample generation; DREAL4 may provide false counterexamples but will always provide a guarantee of a verified CLF being correct.

References

[1] Y.-C. Chang, N. Roohi, and S. Gao. Neural lyapunov control. *Advances in neural information processing systems*, 32, 2019.

[2] H. Ravanbakhsh and S. Sankaranarayanan. Learning control lyapunov functions from counterexamples and demonstrations. *Autonomous Robots*, 43(2):275–307, aug 2018. doi:10.1007/s10514-018-9791-9. URL `https://doi.org/10.1007%2Fs10514-018-9791-9`.

[3] P. A. Parrilo. *Structured semidefinite programs and semialgebraic geometry methods in robustness and optimization*. PhD thesis, 2000.

[4] J. Lofberg. Pre- and post-processing sum-of-squares programs in practice. *IEEE Transactions on Automatic Control*, 54(5):1007–1011, 2009. doi:10.1109/TAC.2009.2017144.

[5] H. Dai, B. Landry, L. Yang, M. Pavone, and R. Tedrake. Lyapunov-stable neural-network control, 2021. URL `https://arxiv.org/abs/2109.14152`.

[6] S. Gao, S. Kong, and E. M. Clarke. dreal: An smt solver for nonlinear theories over the reals. In *International conference on automated deduction*, pages 208–214. Springer, 2013.