Verified Path Following Using Neural Control Lyapunov Functions

Anonymous Author(s)
Affiliation
Address
email

Abstract: We present a framework that uses control Lyapunov functions (CLFs) for implementing provably stable path following for autonomous mobile platforms. Our approach is based on learning a guaranteed CLF for a dynamical model of path following using recent approaches that combine machine learning with automated theorem proving to train neural networks that are verified to obey specific criteria. The resulting neural network CLFs are restricted to stabilizing around simple straight-line or low-curvature paths. We then use key properties of CLFs to derive bounds on curvature for which the original straight-line CLF and resulting control inputs remain valid. We show that our approach yields a controller that obeys theoretical guarantees on behavior in simulation, and also that withstands the test of realism in practice. The controller furthermore can be calculated online at a very high frequency amenable to fast operation on agile platforms. To demonstrate this, we implement the controller on-board a $\frac{1}{8}$-scale autonomous vehicle testing platform and present results for various path following problems.

Keywords: Path Following, Trajectory Tracking, Control Lyapunov Functions, Plan Execution, Verified Autonomy.

1 Introduction

Path following is a fundamental primitive for autonomous robotic platforms, wherein the goal is to steer the platform along a desired path in the workspace. This is important for mobile platforms since many approaches to platform design relies on the separation between high level planners that search over paths without considering platform dynamics and lower-level plan execution which implements path following [1, 2]. However, path-following is a challenging problem since platform models of interest are non-linear. Popular approaches such as the "Stanley" controller (originally proposed by Hoffman et al. for the 2005 DARPA grand challenge [3]) can become unstable depending on the choice of controller parameters and the curvature of the path [4]. Similarly, feedback control based on linearization can also be unstable if the lateral deviation is too large [5].

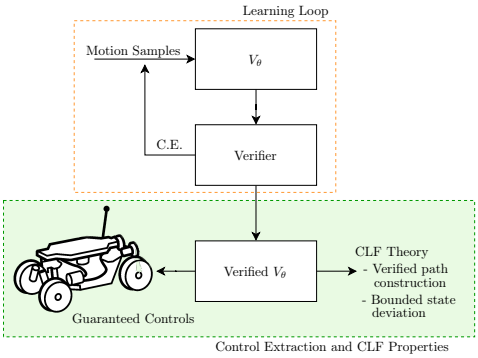


Figure 1: Learning CLFs: Motion samples consist of state-control pairs as training data for the neural network V with parameters θ. The Verifier produces counterexamples (C.E.) until the network satisfies the Verifier. The green box contains this paper's contributions: 1) the extraction of controls from V_θ that ensure bounded deviation of paths, 2) the framework to construct advanced maneuvers using a single CLF rather than a hybrid controller of many CLFs, and 3) the demonstration of this on a real platform.

Control Lyapunov functions (CLFs) are a general approach to designing stabilizing controllers, typically for steering a control system towards an

Submitted to the 6th Conference on Robot Learning (CoRL 2022). Do not distribute.

equilibrium [6, 7]. A CLF is a function which is positive-definite, such that for any state within a well-defined basin of attraction, there exists a control input that can cause its derivative to be negative. Thus, CLFs naturally define stabilizing controllers [8, 9]. However, the generation of CLFs has remained a challenging problem for researchers, and various CLF generation methods have been developed. Neural networks have recently become a popular method for designing CLFs given their ability to converge even on systems with sophisticated dynamics. The key idea is to use machine learning algorithms by defining a special loss function called "Lyapunov risk" in combination with automated theorem proving to verify the results of the learning algorithm. As a result, if successful, the approach yields a CLF as well as a stabilizing controller. Thus, neural CLF generation is a promising approach for designing controllers for nonlinear systems. Nevertheless, its use has been limited to stabilization to fixed equilibria, whereas our application seeks to stabilize to a time varying reference along the path. Secondly, empirical validation on an actual robotic test platform has not been publicly performed.

In this paper, we first formulate the path following problem for a path in the work-space with its curvature given by a function $\kappa(s)$, where s represents distance along the path. Our approach has direct application to existing neural Lyapunov function synthesis tools for $\kappa(s) = 0$, which represents a straight-line path, but is progressively harder as path curvature increases. To address this, we exploit key properties of the generated CLF to design a formalism wherein the controller and CLF for stabilizing around a straight-line path can be used for paths with non-zero curvature. We derive bounds on the maximum path curvature for which the generated straight-line CLF is valid and show that we can construct complex maneuvers using this approach.

Next, we demonstrate our approach on a 1/8-scale vehicle platform in a lab environment. Our demonstrations show that (a) the model-based design approach transfers over to actual robotic platform even though the model used for learning the CLF is a simpler model; and (b) the resulting navigation algorithm performs well for driving around curved paths even though the original CLF was learned over simple straight-line paths.

1.1 Related Work

Numerous methods have been studied for generating CLFs. Certain nonlinear control design methods such as backstepping automatically synthesize CLFs but are not widely applicable. Tan and Packard studied an approach using bilinear constraints that are hard to solve in practice [10]. Majumdar et al. propose a similar approach for path following but use a control feedback law derived through linearization as a heuristic initialization to solve bilinear constraints efficiently in practice [11]. Data-driven approaches for CLFs were first studied by Khansarizadeh et al. [12]. However, a lot of data is potentially needed to generate CLFs and the final result needs to be verified against the system dynamics. Previous work by Ravanbakhsh et al. [13] showed an active approach to deriving CLFs for nonlinear systems that was adapted to path following using the idea of control funnel functions [14, 15]. Furthermore, their approach was shown to be effective on a autonomous vehicle testbed in the lab. However, the techniques described so far focus mostly on deriving polynomial CLFs. It is well known that CLFs can be non-polynomial even for polynomial dynamical systems. Recently neural networks have been used to generate CLFs for systems with complex, non-polynomial dynamics. Introduced by Richards et al. [16] and Chang et al. [17], neural networks have been shown as a effective and reliable method for generation of CLFs for a diversity of systems.

Majumdar et al. [18] provide the framework of "funnel libraries" in which a set of funnels $\mathcal{F}$ and associated feedback controllers are generated. This library of funnels can be sequenced together to form complex vehicle maneuvers. Bouyer et al. [19] showed that these sequential funnels can reduce the reach-avoid problem to that of time automata, that is, a system that can switch between appropriate controllers based on state measurements. Funnel libraries are an interesting approach towards a more rigorous plan execution that supports a motion planner which can restrict itself to maneuvers available in the library. However, maintaining such a library can be cumbersome. In particular, a large number of controllers may be needed to cover the space of possible maneuvers.

The approach presented in this paper could be used to simplify funnel libraries to cover a larger set of possible trajectories with fewer controllers.

2 Preliminaries: Problem Definition and CLF Learning

Consider the problem of stabilizing a dynamical system $\dot{x} = f(x, u)$ where $x \in \mathcal{D} \subseteq \mathbb{R}^n$ and $u \in \mathcal{U} \subseteq \mathbb{R}^m$ to target equilibrium point $x = 0$ without loss of generality. This can be considered as the typical control problem, e.g., where a mobile platform has command inputs that can lead it to follow a specific trajectory which may be parametrized by a point in time. Let $V : \mathcal{D} \to \mathbb{R}$ be a control Lyapunov function (CLF) for f on $\mathcal{D}$. Formally, a CLF is defined in Definition 1.

Definition 1. *A CLF is a continuously differentiable function $V : \mathcal{D} \to \mathbb{R}$ that is positive semi-definite: $V(0) = 0$ and $V(x) > 0$ for all $x \neq 0$ that satisfies the following decrease condition*

$$(\forall\, x \in \mathcal{D} \setminus \{0\})\, (\exists\, u \in \mathcal{U}), \nabla V(x) \cdot f(x, u) < 0. \tag{1}$$

Rather than require the system to reach an equilibrium, we are often interested in region stability, wherein we define a neighborhood $\mathcal{I}$ of 0, and require that the Lyapunov conditions Eq. (1) are satisfied in the region $\mathcal{D} \setminus \mathcal{I}$, called the *Lyapunov (satisfiability) region*. Using the resulting CLF V, controls can be generated for any state x within a level set of V contained completely within $\mathcal{D}$ to drive the system to $\mathcal{I}$ in finite time and then keep the system in a neighborhood of $\mathcal{I}$ for all time [19].

2.1 CLF Generation

We briefly describe the neural network based learning of CLFs [17]. In general, finding V is a hard problem as it requires a search in function space that satisfies the constrained optimization problem in Eq. (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 [17] 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 in Definition 1 after a *learning* phase. The learning process stochastically updates θ to increase the *likelihood* of satisfying the conditions in Eq. (1). 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 conditions in Eq. (1). Generally the problem of designing a CLF is one of finding a function V that satisfies these conditions by minimizing the cost:

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

However, minimizing Eq. (2) 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. (2) making the problem tractable. Namely, we sample N points $x_1, \ldots, x_N$ in the Lyapunov region and compute the empirical risk:

$$L_N(\theta) = \frac{1}{N} \sum_{i=1}^{N} (\max(0, -V_\theta(x_i)) + \max(0, \nabla V_\theta(x_i) \cdot f(x_i, u_i)) + V_\theta^2(0)), \tag{3}$$

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

However, minimizing Eq. (3) might not be sufficient to obtain a CLF, as the Lyapunov conditions in Eq. (1) 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 (see Figure 1), 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 [13], SOS [20, 21], SMT and most recently MIP solvers [22] 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 [23] 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:

$$\textbf{unsat: } \varphi \text{ is unsatisfiable OR } \delta\textbf{-sat: } \varphi^{\delta} \text{ is satisfiable,}$$

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.

Region of Attraction While the target of CLF training is to generate a CLF that is valid inside the Lyapunov region, an important metric is the region of attraction (ROA) [24].

Definition 2. *Let V be a CLF with domain $\mathcal{D}$ for some system $f(x, u)$. Any level set of V completely contained within $\mathcal{D}$ defines a region of attraction. That is, for $\beta > 0$, if $R_\beta = \{V(x) \leq \beta\} \subseteq \mathcal{D}$, then R_β is an ROA for the system. The maximal region of attraction is the maximum level set of V contained within $\mathcal{D}$.*

The Lyapunov conditions do not directly ensure safety (i.e. infinite-time horizon containment) for any states in the verified Lyapunov region since large level sets of V may exit $\mathcal{D}$ while descending the gradient. Upon exiting $\mathcal{D}$ there is no guarantee that actions will be available to reduce V. To ensure safety, a system's initial state must be contained within the maximal region of attraction.

2.1.1 Vehicle Model

To train the system we use a path following model with a fixed velocity. The model consists of two states: d_e and θ_e, where d_e is the distance from the line as measured from the rear axle and θ_e is the theta offset from nearest point on the target line (see Figure 2). The dynamics of this system are:

$$\begin{aligned} \dot{d}_e &= v \sin(\theta_e) \\ \dot{\theta}_e &= \frac{v \ \tan(u)}{L} - \frac{v \ \kappa(s) \ \cos(\theta_e)}{1 - d_e \kappa(s)}, \end{aligned} \tag{4}$$

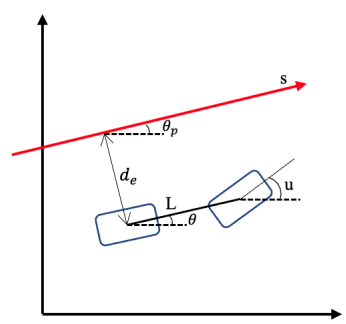


Figure 2: Diagram of the states of the wheeled vehicle described in Equation 4, s being the parameterized curve.

where $\theta_e = \theta - \theta_p$ and $\kappa(s)$ is the curvature of the reference path at the nearest point. Note $\kappa(s)$ is 0 for the straight path shown in Figure 2.

To calculate u at any given (d_e, θ_e) we use an LQR controller. Other controllers may be used to avoid recalculating the LQR gains for various paths.

2.2 Straight Path to Curved Path Following

Finding CLFs for the vehicle model described in Eq. (4) for paths of varying curvature leads to a discrete set of curves (e.g. a funnel library) which the vehicle can execute. Instead, we find a single CLF for the straight-line path, and compute the maximum curvature for which the CLF is valid.

If the minimal required steering input for the straight-line path-following CLF is smaller than the maximum allowed steering input $u_{\max}$, we can derive a curved path-following CLF. More precisely, given a CLF V for the straight-line path, i.e., for Eq. 4 with $\kappa(s) = 0$ with ROA contained in the region $d_e \in [-d_{\max}, d_{\max}]$, we define the minimal required input u_{req} for V as the smallest value of u satisfying that for all state x in the Lyapunov region. Given this, there exists a steering input u with $|u| \leq u_{\text{req}}$ that decreases the value of V, i.e., $\nabla V(x) \cdot f(x, u) < 0$. If $u_{\text{req}} < u_{\max}$, then one can exploit the gap $\tan(u_{\max}) - \tan(u_{\text{req}})$ to follow a path with curvature $\kappa(s)$ bounded as follows.

Proposition 1. *If* $|\kappa(s)| \leq \frac{\tan(u_{\max})-\tan(u_{\text{req}})}{L+d_{\max}}$*, then* V *is a CLF for Eq. 4.*

Proof. Note that for all $d_e \in [-d_{\max}, d_{\max}]$, $|\frac{\kappa(s)\cos(\theta_e)}{1-d_e\kappa(s)}| < \frac{|\kappa(s)|}{1-d_{\max}|\kappa(s)|}$. Thus, if $\frac{|\kappa(s)|}{1-d_{\max}|\kappa(s)|} < \frac{\tan(u_{\max})-\tan(u_{\text{req}})}{L}$, one can use the steering input $u(s) = \arctan(\tan(u) + \frac{L|\kappa(s)|}{1-d_{\max}|\kappa(s)|})$, where $u \in [-u_{\text{req}}, u_{\text{req}}]$ satisfies $\nabla V(x) \cdot f(x, u) < 0$. It holds that $|u(s)| \leq u_{\max}$. Moreover, with this input the second equation of the curved path following Eq. (4) becomes $\dot{\theta} = \frac{v\tan(u)}{L}$, which is the same as when $\kappa(s) = 0$. Since u is a decreasing input for V with $\kappa(s) = 0$, it is clear that V is a CLF for curved path following with input $u(s)$. □

Similar to a library of funnels [18], the CLF generated for straight-line path following ($\kappa(s) = 0$) can be used for any path where $|\kappa(s)| \leq \frac{\tan(u_{\max})-\tan(u_{\text{req}})}{L+d_{\max}}$ per Proposition 1. We can use target paths of various values of $\kappa(s)$ to construct complex vehicle maneuvers. While previous works require state-dependent controller switching [19, 18], our solution relies on a single CLF for the duration of any constructed maneuver.

2.3 Control Extraction

Algorithm 1 Vehicle Rollout CLF Minimization

```
import CLF, set vehicle model parameters, generate target path, discretize steering action space,
set dt (time step), set K_v (control cost)
while sim == true do
    for steer in steeringAngles do
        θ_e, d_e ← calcError(updateState(currentState, steer))
        vxRollouts.append(calcVx(θ_e, d_e))
    end for
    u ← steeringAngles[argmin(square(steeringAngles) * K_v + vxRollouts)]
end while
```

A simple method of extracting the best control to execute given a state in a valid CLF is to simply execute the control that minimizes the Lie derivative at the current state. One can plot a theoretically executable path to descend the CLF as quickly as possible using this method. However this approach usually results in saturation of the control surface. In our system this holds true as the Lie derivative is a monotonic function of u. The control that minimizes the Lie derivative at any point in the CLF will always be the maximum or minimum $u \in \mathcal{U}$. This is referred to as a bang–bang controller [25].

In practice a better method of control extraction is to simulate some actions and select the action that will result in the largest reduction of $V(x)$. Given Eq. (1), there always exists a control to reduce the CLF given the vehicle starts in a valid level set of the CLF. The addition of a small control cost can be included for smoother operation. This approach allows for stabilization to the origin, as well as more realistic models to be introduced to compute realistic maneuvers. A simple, functional implementation of this idea is outlined in Algorithm 1.

This algorithm can be expanded to operate like a traditional MPC [26] by increasing the look-ahead, and minimizing a cost function balancing future $V(x)$ value, control cost, and terminal $V(x)$ value.

3 Implementation and Experimental Evaluation

3.1 CLF Generation

Our neural Lyapunov control framework was successful in generating CLFs for various target paths and vehicle parameters. Figure 4(a) shows a generated CLF for a straight-line path. The CLF is verified inside the Lyapunov region defined by a lower and upper Lyapunov bound of 0.2 and 0.8 respectively. Note again that while the CLF is valid in those bounds, the maximal ROA is the only region in which we can guarantee safety within the specified bounds, although in practice we observe success both converging to the minimum and remaining in the Lyapunov region outside the ROA.

3.2 Controller Implementation and Evaluation

Platform To extract and execute controls in the real world we use a $\frac{1}{8}$-scale vehicle platform in a motion capture space. The wheel base of the vehicle is 34 cm and the velocity for all tests is fixed at 0.5 m/s. The vehicle includes an on-board computer for real-time computation and receives pose estimates from the motion capture system through a WiFi connection. After computing a control action the action is transmitted to an electronic control unit which controls the steering servos and motor for the vehicle. Remarkably, each iteration of Algorithm 1 can be executed in less than 150 μs.

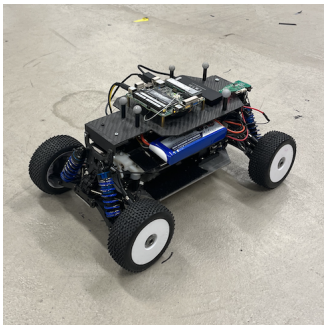


Figure 3: Vehicle Testing Platform

Control Extraction in Real Time The test objective is to use the generated CLF to stabilize the platform to a straight-line path from $x = -2.5$ m to $x = 1$ m at a constant velocity of 0.5 m/s. On the real-world platform the Lie derivative minimization control extraction method is unable to stabilize the platform to a straight line. The platform continuously overshoots the target path and is unable to stabilize, as the control surface is always saturated.

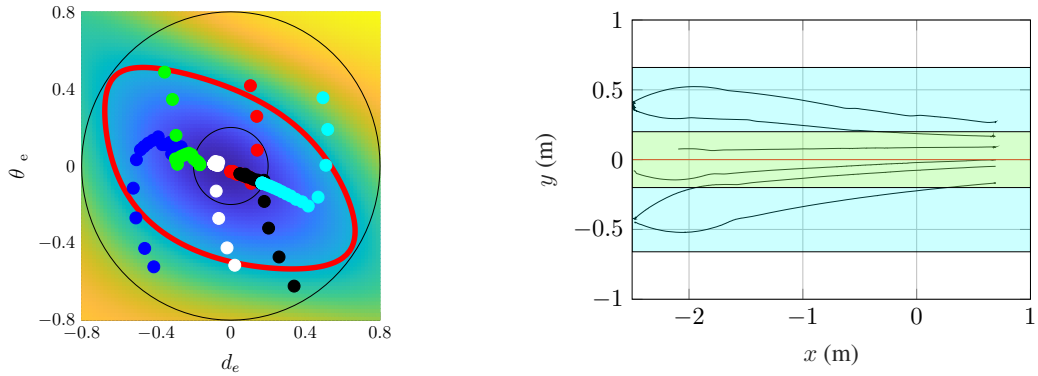
Using Algorithm 1 we simulate potential rollouts for the platform and execute the lowest cost action. We discretize the continuous action space $[-\frac{\pi}{4}, \frac{\pi}{4}]$ to 31 equally-spaced steering values. We use the kinematic bicycle model to project our current state a single time-step into the future, setting the time-step to be equal to our system average update rate $dt = 150$ μs. The platform is positioned at various y- and θ-offsets within the valid region of the Lyapunov function.

Figure 4(a) shows the convergence of all starting positions to the minimum of the CLF using algorithm 1. In practice, while not guaranteed to converge we see that even starting positions outside the maximal ROA converge to $V(x) = 0$.

Given any starting point in the maximal ROA we can plot the guaranteed maximum d_e and θ_e by computing the level-set of $V(x_{start})$ and taking the $\max \theta_e$ and $\max d_e$ of the computed set. Figure 5(a) shows a reduced invariant region based on $x_0 = (0, -0.4)$. For this particular starting position we can guarantee the platform will deviate no further than $[-48, 44]$ cm from the target path (a 27% d_e reduction when compared to the maximal ROA maximum d_e of 0.66).

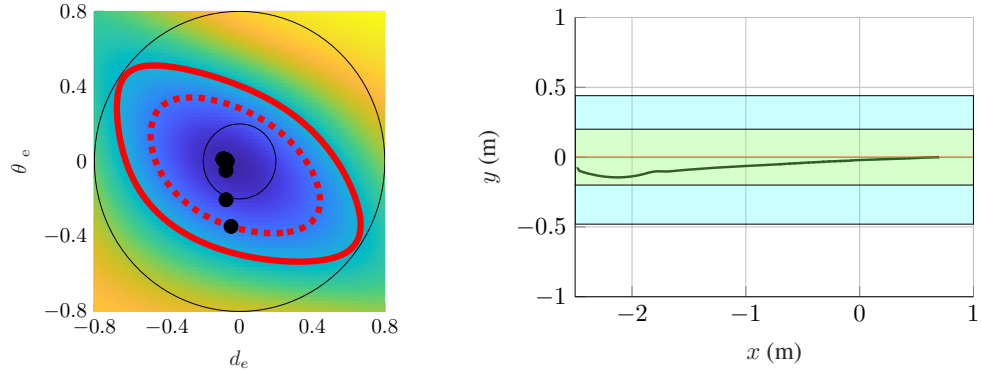
Maximum Curvature Computation In Subsection 2.2, we showed that a CLF for straight-line path following can be extended for curved path following if the range of input required for straight-line path following is smaller than the maximum available steering input $u_{\max} = \frac{\pi}{4}$. The maximum allowed curvature depends on gap $= \tan(u_{\max}) - \tan(u_{\text{req}})$, where u_{req} is the smallest range required for straight-line path following. Given the CLF V computed Subsection 3.1, u_{req} can be computed as follows:

$$u_{\text{req}} = \max_{d_e, \theta_e} \min \left\{ |u| : \frac{\partial V}{\partial d_e}(d_e, \theta_e) v \sin(\theta_e) + \frac{\partial V}{\partial \theta_e}(d_e, \theta_e) \frac{v \tan(u)}{L} < 0 \right\}, \tag{5}$$



(a) (b)

Figure 4: Results of platform rollout CLF minimization on real world platform. (a) Colored points are sampled platform trajectories given various starting positions and orientations, black lines are lower and upper Lyapunov bound, red line is maximal ROA. Surface colors show value of $V(x)$ orange being larger, blue being smaller. (b) Vehicle paths at starting states (d_e, θ_e) : $[(0.4, 0.4), (0.4, -0.4), (0.0, 0.4), (0.0, -0.4), (-0.4, 0.4), (-0.4, -0.4)]$. Green area is the lower Lyapunov bound, red line is the target path, black lines are platform paths, cyan area is the largest d_e in the maximal ROA.



(a) (b)

Figure 5: (a) Reduced invariant region based on starting point (0,-0.4) where solid red line is maximal ROA and dashed red line is the reduced invariant region at $V(0, -0.4)$. The black lines are the upper and lower Lyapunov bound. Surface colors show value of $V(x)$ orange being larger, blue being smaller. (b) Computed d_e boundary when platform starts at starting at $d_e = 0.0$, $t_e = -0.4$, and colors are as in Figure 4a

where the maximum is taken for d_e, θ_e in the Lyapunov region. Eq. (5) can be rewritten as a single maximization problem:

$$\frac{1}{L}\tan(u_{\text{req}}) = \max_{d_e,\theta_e} \left| \frac{\frac{\partial V}{\partial d_e}(d_e, \theta_e)\sin(\theta_e)}{\frac{\partial V}{\partial \theta_e}(d_e, \theta_e)} \right|. \tag{6}$$

Eq. (6) has only two variables (d_e, θ_e), so that it can be solved easily using for instance grid search, or nonlinear solvers like Ipopt [27]. We compute the value $\frac{1}{L}\tan(u_{\text{req}}) = 0.496$. With $L = 0.3$, we get that gap $= 0.8512$. By Proposition 1, with $d_{\max} = 0.8$, it follows that V can be extended to curved path following for any path with curvature $\kappa(s) \leq \frac{\text{gap}}{L+d_{\max}} = 0.774$, i.e., with radius of curvature $R \geq 1.3$ meters.

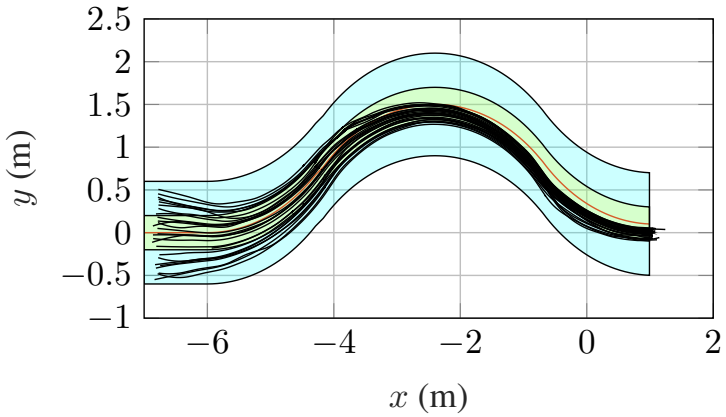


Figure 6: Experimental results where colors are as in Figure 4a. It is shown that if the platform's initial position is contained with the maximal ROA the platform will never exit the teal region.

Using the piecewise path construction approach discussed in Section 2.2 we can construct and execute a complex object avoidance maneuver. The maneuver is generated from 3 paths; a straight path and a left and right turn with a radius of curvature of 2.5 m. Figure 6 shows platform paths of various starting positions and orientations after executing algorithm 1 on the constructed path. The platform path is contained within the maximal ROA for all starting positions tested.

4 Limitations

The primary limitation of this work is the method to generate verified CLFs. We are able to generate various CLFs that fit our system, however many of the CLFs that we would like to design do not converge to a solution (such as CLFs with larger valid Lyapunov regions, or smaller Lyapunov lower bound). The limiting factor is likely the verification phase, specifically the verification of Eq. (1). Rather than verifying $\exists u \in \mathcal{U}$ for which $\nabla V(x) \cdot f(x, u) < 0$, we write u in terms of our LQR controller (a function of the system state) to provide a single u for each state. Of course this greatly constrains the problem and means that the LQR controller constants must be trained simultaneously to the CLF parameters [17]. In theory SMT solvers should be able to directly check Eq. (1) as it is a first order logic inequality, however in practice the control spaces are often too large. In future work one may be able to reduce the space by bounding $\mathcal{U}$ to some neighborhood of a sample control (provided by an LQR or other controller similar to the current framework).

Additionally these verification techniques are conditioned on how well-represented our system is by the model $f(x, u)$. Namely, models which have been developed to represent certain behaviors of our system limit the extent of maneuvers that are verified and may be executed by the platform. This crucial limitation exists at the interface of *model validation*, wherein the appropriateness of the model is evaluated. Possible next steps to address this limitation may be to consider online model synthesis and the extension of verification to these new models.

5 Conclusion

In this work we show the ability to generate CLFs using neural networks [17], extract executable controls to stabilize the system, and construct safe trajectories using a single CLF. Future work will seek improvements to CLF generation and verification. Additionally, CLFs can be used inside a receding horizon formulation in Algorithm 1 to provide potentially smoother control experience and faster stabilization to the target. Finally the feasibility of non-smooth maneuvering (e.g., for emergency/short-notice object avoidance) in this framework would provide another layer of robustness when operating in less ideal conditions.

References

[1] E. Gat. On three-layer architectures. In *Artificial Intelligence and Mobile Robots*. MIT Press, 1998.

[2] L. Claussmann, M. Revilloud, D. Gruyer, and S. Glaser. A Review of Motion Planning for Highway Autonomous Driving. *IEEE Trans. Intell. Transp. Syst.*, 21(5):1826–1848, May 2019. ISSN 1558-0016. doi:10.1109/TITS.2019.2913998.

[3] G. M. Hoffmann, C. J. Tomlin, M. Montemerlo, and S. Thrun. Autonomous automobile trajectory tracking for off-road driving: Controller design, experimental validation and racing. In *2007 American Control Conference*, pages 2296–2301, 2007. doi:10.1109/ACC.2007.4282788.

[4] J. M. Snider. Automatic steering methods for autonomous automobile path tracking. *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RITR-09-08*, 2009.

[5] H. Ravanbakhsh, S. Sankaranarayanan, and S. A. Seshia. Formal policy learning from demonstrations. In *International Conference on Robotics and Automation (ICRA)*, pages 6037–6043. IEEE Press, 2019.

[6] E. Sontag. A lyapunov-like characterization of asymptotic controllability. *SIAM Journal on Control and Optimization*, 21:462–471, 1983.

[7] Z. Artstein. Stabilization with relaxed controls. *Nonlinear Analysis: Theory, Methods & Applications*, 7(11):1163 – 1173, 1983. ISSN 0362-546X. doi:10.1016/0362-546X(83)90049-4.

[8] E. Sontag. A 'universal' construction of artstein's theorem on nonlinear stabilization. *Systems and Control Letters*, 13(2):117–123, Aug. 1989. ISSN 0167-6911. doi:10.1016/0167-6911(89)90028-5. Funding Information: * Research supported in part by US Air Force Grant 88-0235.

[9] Y. Lin and E. D. Sontag. A universal formula for stabilization with bounded controls. *Systems & Control Letters*, 16(6):393 – 397, 1991. ISSN 0167-6911. doi:10.1016/0167-6911(91)90111-Q.

[10] W. Tan and A. Packard. Searching for control lyapunov functions using sums of squares programming. *sibi*, 1(1).

[11] A. Majumdar, A. Ahmadi, and R. Tedrake. Control design along trajectories with sums of squares programming. In *IEEE International Conference on Robotics and Automation, ICRA 2013*, pages 4054–4061, 2013.

[12] S. Mohammad Khansari-Zadeh and Aude Billard. Learning control lyapunov function to ensure stability of dynamical system-based robot reaching motions. *Robotics and Autonomous Systems*, 62(6):752 – 765, 2014. ISSN 0921-8890. doi:10.1016/j.robot.2014.03.001.

[13] 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`.

[14] M. Mason. The mechanics of manipulation. In *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, volume 2, pages 544–548, 1985. doi:10.1109/ROBOT.1985.1087242.

[15] H. Ravanbakhsh, S. Aghli, C. Heckman, and S. Sankaranarayanan. Path-following through control funnel functions. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 401–408, 2018. doi:10.1109/IROS.2018.8593637.

[16] S. M. Richards, F. Berkenkamp, and A. Krause. The lyapunov neural network: Adaptive stability certification for safe learning of dynamical systems. In A. Billard, A. Dragan, J. Peters, and J. Morimoto, editors, *Proceedings of The 2nd Conference on Robot Learning*, volume 87 of *Proceedings of Machine Learning Research*, pages 466–476. PMLR, 29–31 Oct 2018. URL https://proceedings.mlr.press/v87/richards18a.html.

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

[18] A. Majumdar and R. Tedrake. Funnel libraries for real-time robust feedback motion planning. *CoRR*, abs/1601.04037, 2016. URL http://arxiv.org/abs/1601.04037.

[19] P. Bouyer, N. Markey, N. Perrin, and P. schlehuber caissier. Timed-automata abstraction of switched dynamical systems using control invariants. *Real-Time Systems*, 53, 05 2017. doi:10.1007/s11241-016-9262-3.

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

[21] 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.

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

[23] 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.

[24] J. E. Slotine and W. Li. *Applied Nonlinear Control*. Prentice-Hall, Englewood Cliffs, NJ, 1991.

[25] H. Hermes et al. Functional analysis and time optimal control. 1969.

[26] R. Rajamani. *Vehicle Dynamics and Control*. Mechanical Engineering Series. Springer US, 2011. ISBN 9781461414339. URL https://books.google.com/books?id=cZJFDox4KuUC.

[27] A. Wächter and L. T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106:25–57, 2006.