Supplementary Materials

In addition to the sections below, a video demonstrating our controller's performance on the kinematic car trajectory tracking and 2D quadrotor obstacle avoidance benchmarks can be found at https://youtu.be/4MWVLtURxG0, and source code can be found at https://github.com/dawsonc/neural_clbf/.

Proof of Theorem 1

The proof of Theorem 1 follows from the following lemmas, which prove stability and safety of CLBF-based control, respectively.

Lemma 1. *If $V(x)$ is a CLBF, then any control policy $\pi(x) \in K(x)\ \forall\ x \in \mathcal{X}$ will exponentially stabilize the system $\dot{x} = f(x) + g(x)\pi(x)$ to x_{goal}.*

Proof. Since $\pi(x) \in K(x)$, it follows that $\frac{dV}{dt} \leq -\lambda V(x)$ for the closed loop system. Thus, V is a Lyapunov function and proves exponential stability about x_{goal}. □

Lemma 2. *If $V(x)$ is a CLBF, then for any control policy $\pi(x) \in K(x)\ \forall\ x \in \mathcal{X}$ and any initial condition $x(0) \in \mathcal{X}_{\text{safe}}$, $x(t) \notin \mathcal{X}_{\text{unsafe}}\ \forall\ t > 0$ (i.e. any trajectory starting in the safe set will never enter the unsafe region).*

Proof. For convenience, define $\mathcal{V} = V \circ x(t)$. Since $x(0) \in \mathcal{X}_{\text{safe}}$, condition (1c) implies that $\mathcal{V}(0) \leq c$. Conditions (1b) and (1e) ensure that $\mathcal{V}$ is strictly decreasing in time (except when $x(t) = x_{\text{goal}}$, at which point $\mathcal{V}$ is constant at zero). As a result, $\mathcal{V}(t) < \mathcal{V}(0) \leq c\ \forall\ t > 0$. If $x(t)$ were to enter the unsafe region, there would exist $t_u > 0$ such that $\mathcal{V}(t_u) > c$. This is a contradiction, so we conclude that $x(t)$ will never enter the unsafe region for $t > 0$. □

Proof of Theorem 2

Proof. By assumption, f_θ and g_θ are affine in θ. Additionally, the Lie derivatives $L_f V$ and $L_g V$ are affine in f and g, and the rCLBF constraint (4) is affine in $L_f V$ and $L_g V$. As a result, the overall mapping from Θ to the left-hand side of (4) is affine and thus maps the convex hull of $\theta_0, \ldots, \theta_{n_s}$ to the convex hull of $L_{f_{\theta_0}} V + L_{g_{\theta_0}} V u + \lambda V, \ldots, L_{f_{\theta_{n_s}}} V + L_{g_{\theta_{n_s}}} V u + \lambda V$. It follows that if (4) is satisfied for each scenario θ_i then it will be satisfied for any possible $\theta \in \Theta$. We can conclude that the rCLBF satisfies the conditions of a standard CLBF for any particular realization of the system with parameters $\theta \in \Theta$, so the safety and stability results of Theorem 1 apply. □

Implementation of Learning Approach

In this section, we describe several details of our implementation of the system used to train V and π_{NN}. At a high level, our system is implemented in PyTorch [40] using PyTorch Lightning [41]. All neural networks were implemented with $\tanh$ activation functions, and we used batched stochastic gradient descent with weight decay for optimization (with learning rate 10^{-3} and decay rate 10^{-6}). The next paragraphs describe our training strategies.

Sampling of training data: we found that training performance could be improved by specifying a fixed percentage of training points that must be sampled from the goal, safe, and unsafe regions. For example, instead of sampling N_{train} points uniformly from the state space, we might sample $0.1N_{train}$ uniformly from the goal region, $0.1N_{train}$ uniformly from the unsafe region, $0.1N_{train}$ uniformly from the safe region, and the remaining $0.7N_{train}$ uniformly from the entire state space.

Network initialization: although it was not necessary for all experiments, we found that some experiments (particularly the car trajectory tracking benchmarks) performed better if the CLBF network was initialized to match the quadratic Lyapunov function found by linearizing the system about the goal point. After training V for several epochs to match this quadratic initial guess, we then alternated between training V and u_{NN}, optimizing one for several epochs before optimizing the other.

We found that on some examples this stabilized the learning process. We did not notice an improvement from episodic learning, although this may be more useful when training on higher-dimensional systems.

Hyperparameter tuning: during the development process, we optimized hyperparameters (c, λ, ϵ, the size of the V and u_{NN} networks, and the penalty applied to relaxations of the QP constraints) based on a combination of the empirical loss on a test data set and through controller performance in simulation. In most experiments, we found that $c = 1$ and $\lambda \in [0.1, 10]$ were sensible defaults, along with neural networks with 2 hidden layers of 64 units each. We found that tuning parameters $a_1 = a_2 = 100$ and $a_3 = 1$ yield controllers that perform well in simulation.

Reach-avoid problem specification: when defining reach-avoid problems for this approach, care should be taken when specifying $\mathcal{X}_{\mathrm{safe}}$ and $\mathcal{X}_{\mathrm{unsafe}}$. We found that it is necessary to have some region in between the safe and unsafe sets where the neural rCLBF has the freedom to adjust the boundary at $V(x) = c$ as needed to find a valid rCLBF. In addition, we found that including a safety constraint that prevents the system from leaving the region where training data was gathered improves the controller's performance.

rCLBF-QP Relaxation: to ensure that the controller is always feasible, we permit the QP to relax the constraints on the CLBF derivative, and the extent of this relaxation is penalized with a large coefficient in the QP objective. The penalty coefficients used in different experiments are included below. This relaxation also provides a useful training signal for the V network. To make use of this signal, we solve (rCLBF-QP) for each point at training-time and scale the last term of the loss function point-wise by the relaxation, effectively increasing the penalty for regions where the feasible set of (rCLBF-QP) is empty and decreasing the penalty in regions where there exists a feasible solution (even if π_{NN} has not yet converged to find that feasible solution).

Verification of Learned CLBFs

Our focus in this paper is primarily on the use of robust CLBFs to automatically synthesize feedback controllers for nonlinear safe control tasks. We find that our learning method yields functions that satisfy the rCLBF conditions in the vast majority of the state space, and yields feedback controllers that are successful in simulation, but we do not claim to have exhaustively verified our learned rCLBFs. Indeed, scalable verification for learned certificate functions remains an open problem. Relevant verification techniques include neural network reachability analysis (see [29] for a recent survey), SMT solvers [8], Lipschitz-informed sampling methods [30], and probabilistic claims from learning theory [10].

Additionally, these verification techniques might be used in future work to inform the training of an rCLBF neural network. For instance, spectral normalization [42] of the rCLBF network would allow us to tune the Lipschitz constant of $V(x)$, enabling more effective use of Lipschitz-informed sampling verification tools. Similarly, reachability tools and SMT solvers can provide counter-examples to augment the training data and make further failures less likely [8]. Further, almost Lyapunov functions [31, 43] show that even if the Lyapunov conditions do not hold everywhere the system is still provably stable; this result may generalize to CLBFs as well. These are all exciting directions that we hope to explore in our future work on this topic.

Implementation of Robust MPC

We implemented our robust MPC scheme in Matlab following the example in the YALMIP documentation [32], which is in turn based on the algorithm published in [32]. This MPC algorithm relies on a linearization of the system dynamics, and we used a constant linearization about the goal state. For trajectory tracking examples, we linearize the system about the reference trajectory.

The robust MPC problem was formulated in YALMIP and Gurobi [44] was used as the underlying QP solver. When measuring evaluation times for robust MPC, we first use YALMIP to precompile the robust QP then measure the time needed to solve the compiled QP using Matlab's built-in `timeit` function. We understand that additional optimizations (e.g. explicit MPC) might reduce the evaluation time of robust MPC further, but those optimizations can be applied equally well to speeding up the QP solution in our proposed controller. Effectively, for the purposes of mea-

suring performance, we optimize both approaches to the point where a single quadratic program is being sent to the Gurobi QP solver, and so we believe we have provided a fair comparison in our results.

Implementation of Hamilton Jacobi Control Synthesis

To compute the Hamilton-Jacobi value function, we used the helperOC package at `https://github.com/HJReachability/helperOC`, which wraps the toolboxLS software [38]. We over-approximate the parametric uncertainty with an additive uncertainty. In the Segway example, where the unsafe set is defined in terms of (x, y), we over-approximate this unsafe set using a polytope defined on (p, θ). We computed the HJ value function, then applied the optimal HJ controller forwards described in [45]. We used a time step of 0.05 seconds and a maximum horizon of 5 seconds while computing the backwards reachable set. The HJ value function was approximated on a grid, and the grid resolution was set to balance accuracy and running time.

Details on Simulation Experiments

This section reports the dynamics and hyperparameters used in our experiments. Note that in some of our examples, mass is an uncertain parameter but enters into the dynamics as $1/m$ (similarly for rotational inertia). In these cases we treat $1/m$ as the uncertain parameter and proceed with our method as described in Section 5. For clarity, we give the uncertainty ranges in terms of m rather than in terms of the reciprocal.

Training was conducted on a workstation with a 32-core AMD 3970X CPU and four Nvidia GeForce 2080 Ti GPUs (one GPU was used for each training job, allowing us to parallelize our experiments). Runtime evaluation was conducted on a consumer laptop with an Intel i7-8565U CPU running at 1.8 GHz, and no GPU.

Kinematic Car

We use the kinematic single track model of a car given in the CommonRoad benchmarks [34]. We modify this model to express position and orientation relative to a reference path parameterized by v_{ref}, a_{ref}, ψ_{ref}, and ω_{ref} (the linear velocity and acceleration, angle, and angular velocity of the reference path). To model a reference path with uncertain curvature, we treat ω_{ref} as the uncertain parameter and assume that it vares on $[-1.5, 1.5]$.

The state of the path-centric kinematic car model is $[x_e, y_e, \delta, v_e, \psi_e]$, representing Cartesian error, steering angle, velocity error, and heading error, and the control inputs are v_δ and a_{long} (the steering angle velocity and longitudinal acceleration). The dynamics are given by $\dot{x} = f(x) + g(x)u$, with

$$f(x) = \begin{bmatrix} v\cos(\psi_e) - v_{ref} + \omega_{ref} * y_e \\ v\sin(\psi_e) - \omega_{ref} x_e \\ 0 \\ -a_{ref} \\ \frac{v}{l_r + l_f}\tan(\delta) - \omega_{ref} \end{bmatrix} \tag{6}$$

$$g(x) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \tag{7}$$

where we define $v = v_e + v_{ref}$ and l_f and l_r are vehicle parameters measuring the distance from the center of mass to the front and rear axles (these parameters are taken from the CommonRoad `vehicle-2` benchmark).

We define a goal point x_{goal} as the origin with nominal parameters $v_{ref} = 10.0$, $a_{ref} = 0.0$, $\omega_{ref} = 0.0$ (note that the reference heading and reference position do not enter directly into the dynamics). These tracking tasks are not reach-avoid tasks, as there is no hard constraint other than maintaining bounded tracking error. We used the LQR solution with nominal parameters for $\pi_{nominal}$. Training data were sampled from $x_e, y_e, v_e \in [-3, 3]$, $\psi_e \in [-\pi/2, \pi/2]$, and $\delta \in [-1.066, 1.066]$, but we

selectively re-sampled until at least 40% of the data were within $\|x\| \leq 1$, at least 20% were within $\|x\| \leq 0.25$, and at least 20% were $\|x\| \geq 1.5$, which ensured that adequate training data were sampled from near the goal point. 125,000 samples were used for training, with 10% reserved for validation. V and π_{NN} are parameterized as two-layer fully-connected neural networks with hidden layer size of 64 and $\tanh$ activation. We set $c = 1$, $\lambda = 1$, and allowed relaxations of the constraints in (rCLBF-QP) with penalty coefficient 10.

A contour plot of the learned V is shown in Fig. 5 as a function of x_e and y_e, with all other state variables zero. From this plot, we see that some violation of (3) occurs near the origin (the violation was computed on a grid with maximum spacing of 0.008 between points). This makes sense, as this system is likely impossible to robustly stabilize around the origin (i.e. we suspect that there is no fixed u that renders the origin a fixed point for any ω_{ref}). Outside the origin, we see that the CLBF conditions are satisfied, which agrees with what we observe in simulation: our controller leaves the origin but then stabilizes with a relatively constant tracking error.

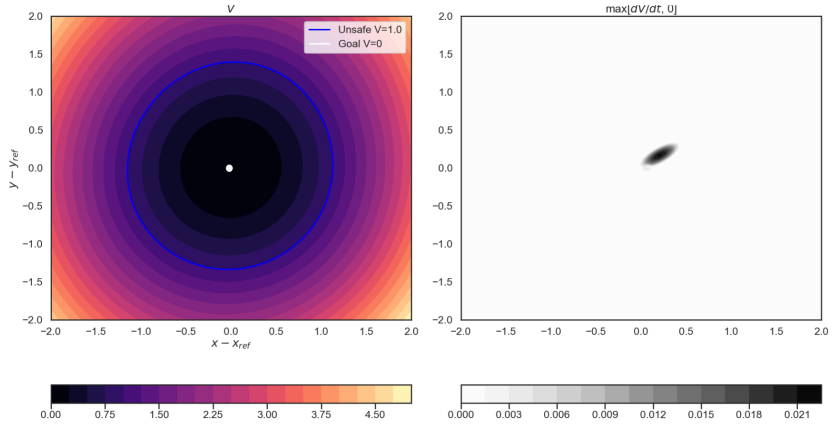

Figure 5: A contour plot of the learned rCLBF V (right) and violation of condition (3) (left) for the kinematic car tracking task. The violation of the rCLBF decrease condition (3), which was found to be at most 0.0225 over this range, was computed as $\max(dV/dt, 0)$, summed over both parameter scenarios.

Car with Sideslip

We use the single-track model given in the CommonRoad benchmarks [34]. Similarly to the kinematic model, we express the dynamics in path-centric coordinates and treat ω_{ref} (which determines the curvature of the reference path) as the uncertain parameter and assume that it vares on $[-1.5, 1.5]$.

Compared to the kinematic model, the single-track (sideslip) model has two additional state variables: β (the sideslip angle) and $\dot{\psi}_e$ (the rate of change of the heading error). The control inputs are the same as for the kinematic model. The dynamics are given by $\dot{x} = f(x) + g(x)u$, with

$$f(x) = \begin{bmatrix} v\cos(\psi_e) - v_{ref} + \omega_{ref} * y_e \\ v\sin(\psi_e) - \omega_{ref} x_e \\ 0 \\ 0 \\ \dot{\psi}_e \\ -\frac{\mu m}{vI_z(l_r+l_f)}(l_f^2 C_{Sf} g l_r + l r^2 C_{Sr} g l_f)(\dot{psi} + \omega_{ref}) + \frac{\mu m}{I_z(l_r+l_f)}(l_r C_{Sr} g l_f - l_f C_{Sf} g l_r)\beta + \frac{\mu m}{I_z(l_r+l_f)}(l_f C_{Sf} g l_r)\delta \\ (\frac{mu}{v^2(l_r+l_f)}(C_{Sr} g l_f l_r - C_{Sf} g l_r l_f) - 1)(\dot{psi}_e - \omega_{ref}) - \frac{\mu}{v(l_r+l_f)}(C_{Sr} g l_f C_{Sf} g l_r)\beta + \frac{\mu}{v(l_r+l_f)}(C_{Sf} g l_r) * \delta \end{bmatrix} \tag{8}$$

$$g(x) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \tag{9}$$

where we define $v = v_e + v_{ref}$ and $l_f, l_r, C_{Sf}, C_{Sr}, \mu$ are parameters whose values taken from the CommonRoad `vehicle-2` benchmark (g is gravitational acceleration). Since these dynamics become singular at low speeds, for $|v| < 0.1$ we revert to the kinematic model (as described in [34]).

We define a goal point x_{goal} as the origin with nominal parameters $v_{ref} = 10.0$, $a_{ref} = 0.0$, $\omega_{ref} = 0.0$ (note that the reference heading and reference position do not enter directly into the dynamics). These tracking tasks are not reach-avoid tasks, as there is no hard constraint other than maintaining bounded tracking error. We used the LQR solution with nominal parameters for $\pi_{nominal}$. Training data were sampled from $x_e, y_e, v_e \in [-3, 3]$, $\psi_e \in [-\pi/2, \pi/2]$, $\dot{\psi}_e \in [-\pi/2, \pi/2]$, $\delta \in [-1.066, 1.066]$, and $\beta \in [-\pi/3, \pi/3]$, but we selectively re-sampled until at least 40% of the data were within $\|x\| \leq 0.35$, at least 20% were within $\|x\| \leq 0.25$, and at least 20% were $\|x\| \geq 0.85$, which ensured that adequate training data were sampled from near the goal point. 125,000 samples were used for training, with 10% reserved for validation. V and π_{NN} are parameterized as two-layer fully-connected neural networks with hidden layer size of 64 and $\tanh$ activation. We set $c = 1$, $\lambda = 0.1$, and allowed relaxations of the constraints in (rCLBF-QP) with penalty coefficient 10^8.

When we examine the contour plot of the learned V (shown in Fig. 6 as a function of x_e and y_e, with all other state variables zero), we see that it has a similar shape as that learned for the kinematic model, but we did not detect any violation of (3) on a grid with maximum spacing of 0.004 between points. However, given our controller's simulation performance on this task, which included some error relative to the goal, it is likely that some violation of the CLBF conditions would be seen on other cross sections (i.e. where state variables other than x_e and y_e are varied).

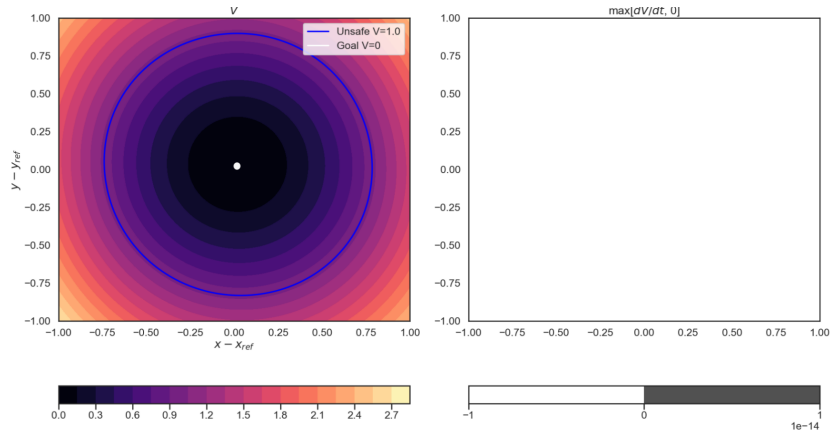

Figure 6: A contour plot of the learned rCLBF V (right) and violation of condition (3) (left) for the sideslip car tracking task. The rCLBF decrease condition (3), computed as $\max(dV/dt, 0)$, summed over both parameter scenarios, was not found to be violated on this range.

3D Quadrotor

The state of the 9-dimensional quadrotor is given by $x = [p_x, p_y, p_z, v_x, v_y, v_z, \phi, \theta, \psi]$, with control vector $u = [f, \dot{\phi}, \dot{\theta}, \dot{\psi}]$. This model is adapted from [9]. The system is parameterized by mass m. The dynamics are given by $\dot{x} = f(x) + g(x)u$, with

$$f(x) = [v_x, v_y, v_z, 0, 0, -g, 0, 0, 0]^T \tag{10}$$

$$g(x) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -(1/m)\sin\theta & 0 & 0 & 0 \\ (1/m)\cos\theta\sin\phi & 0 & 0 & 0 \\ (1/m)\cos\theta\cos\phi & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{11}$$

where g is the gravitational acceleration. Note that although these dynamics are not affine in m, they are affine in $1/m$, which can be treated as the uncertain parameter without loss in generality.

In this task, x_{goal} is the origin, $\mathcal{X}_{\text{safe}} = \{x : p_z \geq 0 \wedge \|x\| \leq 3\}$, and $\mathcal{X}_{\text{unsafe}} = \{x : p_z \leq -0.3 \vee \|x\| \geq 3.5\}$. To model uncertainty in the mass of the quadrotor's payload, we simulate both the rCLBF-QP and MPC controllers with masses sampled uniformly from $m \in [1.0, 1.5]$. To isolate the impact of parameter variation on controller performance, we use a constant initial condition $x(0) = [1, 1, 1, 1, 1, -1, 1, 1, 1]$. We used scenarios $m_0 = 1.0$ and $m_1 = 1.5$ in the rCLBF-QP controller.

For this example, V is parameterized as a two-layer fully-connected neural network with hidden layer size of 48 and $\tanh$ activation. π_{NN} is represented as a three-layer fully connected network with the same hidden layer size. Training data were sampled from $p_x, p_y, p_z \in [-4, 4]$, $\phi, \theta, \psi \in [-\pi/2, \pi/2]$, and $v_x, v_y, v_z \in [-8, 8]$. We used π_{nominal} based on an LQR approximation (ignoring state constraints). We set $c = 10$, $\lambda = 1$, and did not allow relaxations of the constraints in (rCLBF-QP).

In addition to simulating the performance of our controller, we can also examine the learned rCLBF function itself. A contour plot of V as a function of p_x and p_z (all other states set to zero) is shown in Fig. 7, comparing the level set at $V(x) = 0$ to the safe/unsafe boundaries. In computing this plot, we also computed the maximum violation of the rCLBF condition (3), which we found to be 1.9×10^{-4}. This plot was computed by sampling uniformly from p_x and p_z, and the maximum distance between adjacent grid points was 0.008. Based on this extremely small maximum violation (which occurs in a relatively small region of the space), we can conclude that our learning approach yields an rCLBF that is valid throughout most of the domain $\mathcal{X}$. The violation regions shown in this plot differ from those highlighted in Fig. 3 because the grid size in Fig. 7 is much smaller, highlighting the sparsity of violations in the state space. An interesting area for future work might involve counter-example guided training of the rCLBF, as in [8], to resolve these sparse violations. On the other hand, the theory of almost Lyapunov functions [31] suggests that these sparse, low-magnitude violations may not necessarily invalidate the learned CLBF.

Neural Lander

The state of the neural lander is given by $x = [p_x, p_y, p_z, v_x, v_y, v_z]$, with control vector $u = [f_x, f_y, f_z]$. p_z is defined to be positive upwards in this case. This model was developed in [35]. The system is parameterized by mass m. The dynamics are given by $\dot{x} = f(x) + g(x)u$, with

$$f(x) = [v_x, v_y, v_z, F_{a1}/m, F_{a2}/m, F_{a3}/m - g]^T \tag{12}$$

$$g(x) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1/m & 0 & 0 \\ 0 & 1/m & 0 \\ 0 & 0 & 1/m \end{bmatrix} \tag{13}$$

where g is the gravitational acceleration and F_a is the learned disturbance due to ground effect, represented as a 4-layer neural network. The presence of a learned component in the dynamics

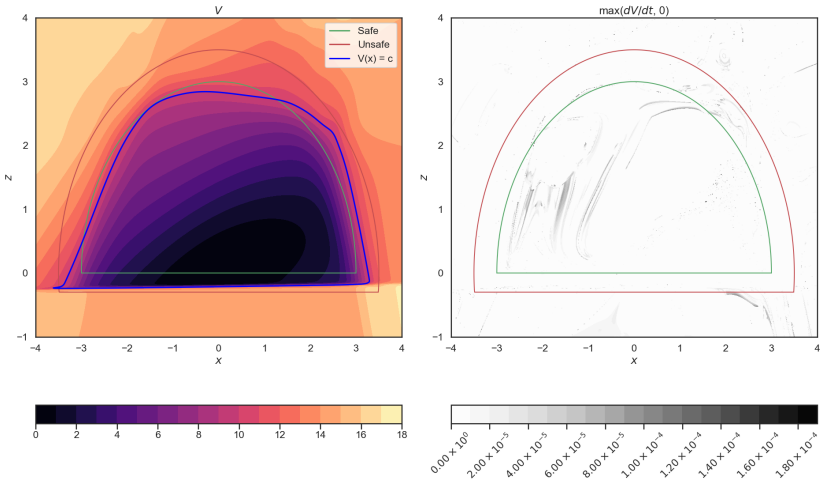

Figure 7: A contour plot of the learned rCLBF V (right) and violation of condition (3) (left) for the 3D quadrotor hovering task. The violation of the rCLBF decrease condition (3), which was found to be at most 1.9×10^{-4} over this range, was computed as $\max(dV/dt, 0)$, summed over both parameter scenarios.

means that many traditional control synthesis techniques (including sum-of-squares techniques) do not apply to this system. As with the 9-dimensional quadrotor, these dynamics are not affine in m, but they are affine in $1/m$, which can be treated as the uncertain parameter without loss in generality.

We use a similar safe hover task to that used for the 3D quadrotor, with x_{goal} at the origin, $\mathcal{X}_{\text{safe}} = \{x : p_z \geq -0.05 \wedge \|x\| \leq 3\}$, and $\mathcal{X}_{\text{unsafe}} = \{x : p_z \leq -0.3 \vee \|x\| \geq 3.5\}$. The mass of the vehicle is sampled uniformly from $m \in [1.47, 2.00]$ and initial conditions $x(0) = [0.5, 0.5, 0.5, 0.5, 0.5, -1.0]$.

For this example, V is parameterized as a two-layer fully-connected neural network with hidden layer size of 48 and $\tanh$ activation. π_{NN} is represented as a three-layer fully connected network with the same hidden layer size. Training data were sampled from $p_x, p_y \in [-5, 5]$, $z \in [-0.5, 2]$, and $v_x, v_y, v_z \in [-1, 1]$. We used π_{nominal} based on an LQR approximation (ignoring state constraints and learned dynamics). For completeness, the learned rCLBF for the neural lander with a high-resolution plot of the violation region is included in Fig. 8. We set $c = 10$, $\lambda = 0.1$, and penalized relaxations of the constraints in (rCLBF-QP) with penalty coefficient 7.

2D Quadrotor

The state of the 2D quadrotor model is given by $x = [p_x, p_z, \theta, v_x, v_y, \dot{\theta}]$, with control vector $u = [u_1, u_2]$. p_z, u_1, and u_2 are defined to be positive upwards. This model is adapted from [9]. The system is parameterized by mass m, rotational inertia I, and the distance of the rotors from the center of mass r, and we take m and I to be the uncertain parameters. The dynamics are given by $\dot{x} = f(x) + g(x)u$, with

$$f(x) = \begin{bmatrix} v_x, v_z, \dot{\theta}, 0, -g, 0 \end{bmatrix}^T \tag{14}$$

$$g = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ (1/m)\sin\theta & (1/m)\sin\theta \\ (1/m)\cos\theta & (1/m)\cos\theta \\ r/I & -r/I \end{bmatrix} \tag{15}$$

where g is the gravitational acceleration. These dynamics are not affine in m and I, but they are affine in $1/m$ and $1/I$, allowing the use of our rCLBF approach.

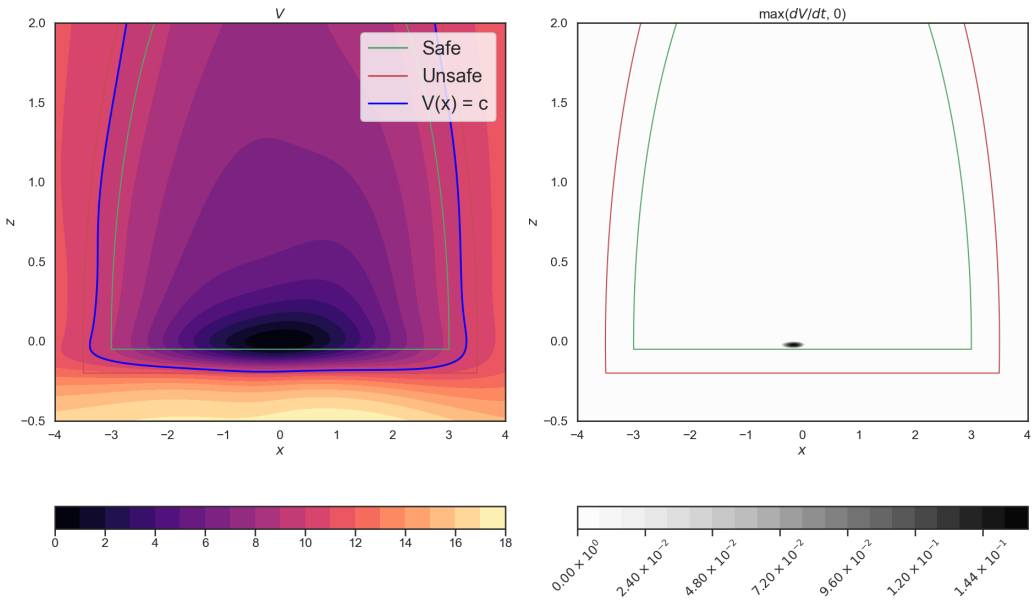

Figure 8: A contour plot of the learned rCLBF V (right) and violation of condition (3) (left) for the neural lander hovering task. The violation of the rCLBF decrease condition was computed as $\max(dV/dt, 0)$, summed over both parameter scenarios. We found the violation to be very small and restricted to a small region of the state space. This plot was computed on a grid in p_x and p_z (all other states set to zero). The maximum distance between grid points is 0.008.

$\mathcal{X}_{\text{unsafe}}$ is set to be the region inside the obstacles, and $\mathcal{X}_{\text{safe}}$ is offset from the obstacle boundaries by 0.1 m. To prevent the controller from driving the system out of region covered by the training data, we include a norm constraint in the safe and unsafe sets, $\|x\| \leq 4.5$ in $\mathcal{X}_{\text{safe}}$ and $\|x\| \geq 5$ in $\mathcal{X}_{\text{unsafe}}$. To model uncertainty in the mass and inertia of the quadrotor, we vary mass and inertia in $(m, I) \in [1.0, 1.05] \times [0.01, 0.0105]$, with nominal values $m_0 = 1.0$ and $I_0 = 0.01$ (the extreme points of this set are used as scenarios in the rCLBF-QP method).

For this example, V is parameterized as a two-layer fully-connected neural network with hidden layer size of 48 and $\tanh$ activation. π_{NN} is represented as a three-layer fully connected network with the same hidden layer size. Training data were sampled from $p_x, p_z \in [-4, 4]$, $\theta \in [-\pi, \pi]$, $v_x, v_z \in [-10, 10]$, and $\dot{\theta} = [-2\pi, 2\pi]$. We used π_{nominal} based on an LQR approximation (ignoring obstacles). We set $c = 1$, $\lambda = 6$, and penalized relaxations of the constraints in (rCLBF-QP) with penalty coefficient 1100.

To gain insight into the performance of our learning-based approach to rCLBF synthesis, we can examine the contour plot of the learned rCLBF $V(x)$, shown in Fig. 9. These contours were computed on a grid in p_x and p_z, with other states set to zero and the maximum distance between adjacent grid points equal to 0.003. We see that the level set of the learned rCLBF at $V(x) = c$ aligns well with the boundaries of the obstacles, and that the rCLBF derivative condition is satisfied in most of the state space (a small violation $\leq 7.3 \times 10^{-2}$ is observed in a small region).

Satellite

In this example, we consider the satellite rendezvous and docking task adopted from [37]. As is shown in Fig. 1, the blue chaser satellite attempts to close the distance to the black target satellite. Within the green dashed circle, the chaser must stay in the green sector, which represents the line-of-sight (LOS) region where the satellite's sensors are most effective. While both the target and chaser satellites orbit around the Earth, we choose a relative coordinate system centered at the target. The motion of the chaser with respect to the target can be modeled by the Clohessy-Wiltshire-Hill (CWH) equations [46]. The state $x = [p_x, p_y, v_x, v_y]$ is consisted of the relative position and velocity. The control inputs $u = [f_x, f_y]$ are the forces applied to the chaser satellite. To keep the

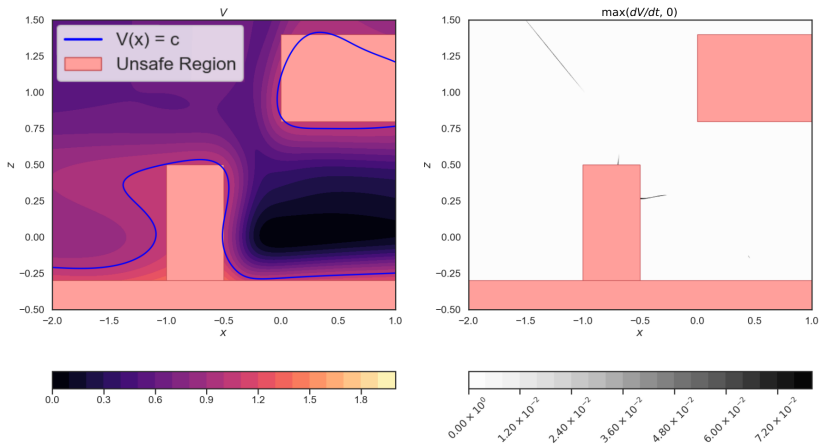

Figure 9: A contour plot of the learned rCLBF V (left) and violation of condition (3) (right) for the 2D quadrotor with obstacles. The violation of the rCLBF decrease condition (3), which was found to be at most 7.3×10^{-2} over this range, was computed as $\max(dV/dt, 0)$, summed over all parameter scenarios.

chaser satellite within the LOS region, we define the safe set as $\mathcal{X}_{\text{safe}} = \{x : 4 \leq \sqrt{p_x^2 + p_y^2} \leq 8 \vee (p_y \leq -|p_x| \wedge \sqrt{p_x^2 + p_y^2} \leq 4)\}$, which represents the green sector plus the ring between the grey circle and the green circle. The unsafe set is defined as $\mathcal{X}_{\text{unsafe}} = \{x : \sqrt{p_x^2 + p_y^2} \geq 9 \vee (p_y \geq -|p_x| \wedge \sqrt{p_x^2 + p_y^2} \leq 3)\}$.

There are two crucial parameters in the model dynamics: the mass of the chaser satellite and the mean-motion $\sqrt{\frac{\mu}{a^3}}$ of the target satellite. μ is the Earth's gravity constant and a is the length of the semi-major axis of the target's orbit.

The dynamics are given by $\dot{x} = f(x) + g(x)u$, with

$$f(x) = \begin{bmatrix} v_x, v_y, 2nv_y + 3n^2 p_x, -2nv_x \end{bmatrix}^T \tag{16}$$

$$g(x) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1/m & 0 \\ 0 & 1/m \end{bmatrix} \tag{17}$$

We used a three-layer fully connected neural network with hidden size 256 and $\tanh$ activation to represent the rCLBF V. During training, the state samples are uniformly drawn from the state space with range $[-12, 12]$ for each dimension. We set the nominal controller $\pi_{\text{nominal}} = 0$. In the implementation of MPC, we minimize the distance between the goal position and the last step of the planning horizon, subject to the control input constraint $f_x, f_y \in [-20, 20]$. We set constraints to enforce the chaser satellite to enter the green sector rather than anywhere else in the green circle. The planning horizon was 10 steps with timestep 0.02s.

We can examine the learned rCLBF V by plotting its contour in Fig. 10 (left) as a function of p_x and p_y (all other states set to zero). The contour plot shows that the learned V is able to distinguish the safe and unsafe sets. In Fig. 10 (right), we plot the violation of rCLBF decrease condition (3). For most of the samples within the range, the condition is satisfied, and we observe only a slight violation less than 5.5×10^{-2} for $(p_x, p_y) \in [6, 10] \times [-12, -10]$.

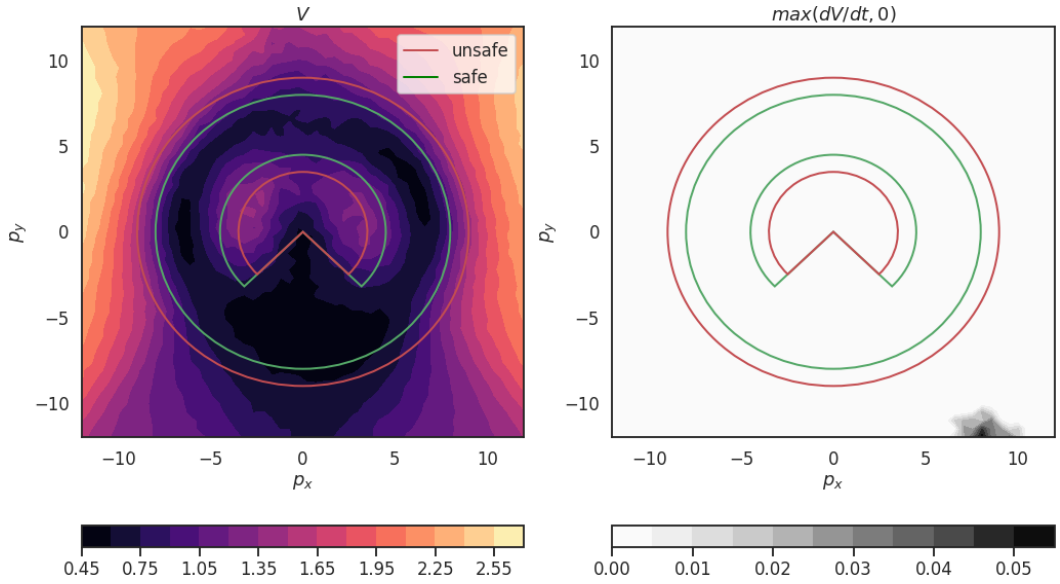

Figure 10: A contour plot of the learned rCLBF V (left) and violation of condition (3) (right) for the satellite rendezvous task.

Segway

We consider the Segway obstacle avoidance task illustrated in Fig. 1. The Segway attempts to avoid the obstacle while moving forward, which requires it to tilt forward to avoid collision. The state $x = [p, \theta, v, \omega]$ includes the horizontal position, angle, velocity and angular velocity of the Segway. The control u is the force applied at the base of the system. We assume the vertical position of the wheel's center is always 0 and the length of Segway is 1. The obstacle is a circle with radius 0.1 centered at $(0, 1)$. Denote the position of the Segway's top as $(p_x, p_y) = (p + \sin(\theta), \cos(\theta))$. Then the unsafe set $\mathcal{X}_{\text{unsafe}} = \{x | \sqrt{p_x^2 + (p_y - 1)^2} \leq 0.1\}$. We define the safe set as $\mathcal{X}_{\text{safe}} = \{x | \sqrt{p_x^2 + (p_y - 1)^2} \geq 0.15\}$.

The Segway model is from Chapter 3.2 of [36]. The state $x = [p, \theta, v, \omega]$ with control input u as the force aligned with p applied at the base of the system. Let M be the mass of the base, m and J be the mass and inertia of the system to be balanced. Denote the distance from the base to the center of mass of the system as l. Let g be the gravity constant. Define $M_t = M + m$ as the total mass and $J_t = J + ml^2$ be the total inertia. The system dynamics are given by $\dot{x} = f(x) + g(x)u$, with

$$f(x) = \begin{bmatrix} v \\ \omega \\ \frac{g s_\theta c_\theta + \lambda_1 v c_\theta + \lambda_2 v - l\omega^2 s_\theta}{c_\theta - \frac{M_t J_t}{m^2 l^2} + \lambda_9} \\ \frac{\lambda_3 v c_\theta + \lambda_4 v - \frac{M_t g}{ml} s_\theta - \omega^2 s_\theta c_\theta}{c_\theta^2 - \frac{M_t J_t}{m^2 l^2} + \lambda_9} \end{bmatrix} \tag{18}$$

$$g(x) = \begin{bmatrix} 0 \\ 0 \\ \frac{\frac{\lambda_6}{M_t}(\lambda_5 + c_\theta)}{c_\theta^2 - \frac{M_t J_t}{m^2 l^2} + \lambda_9} \\ \frac{\frac{\lambda_8 l}{J_t}(c_\theta + \lambda_7)}{c_\theta^2 - \frac{M_t J_t}{m^2 l^2} + \lambda_9} \end{bmatrix} \tag{19}$$

In the implementation of the rCLBF V, we used a three-layer fully connected neural network with hidden size 64 and $\tanh$ activation. The lower and upper bound of the state space are $[-3, -\pi/2, -1, -3]^T$ and $[3, \pi/2, 1, 3]^T$. Training samples are uniformly sampled from this range. We used an LQR controller for π_{nominal}. In the MPC baseline, we minimize the angle w.r.t. the vertical axis at each step, subject to the top of the Segway being outside the unsafe set. At each step, when the solver fails find a feasible solution satisfying the constraint, we use $u = 0$ for that step. The planning horizon is 10 steps, and the timestep is set to 0.005.

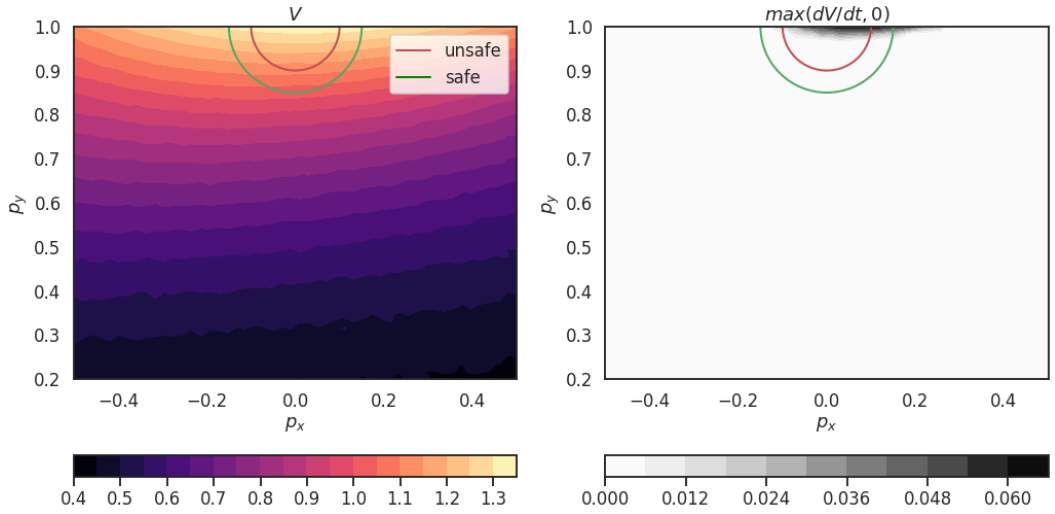

Figure 11: A contour plot of the learned rCLBF V (left) and violation of condition (3) (right) for the Segway obstacle avoidance task.

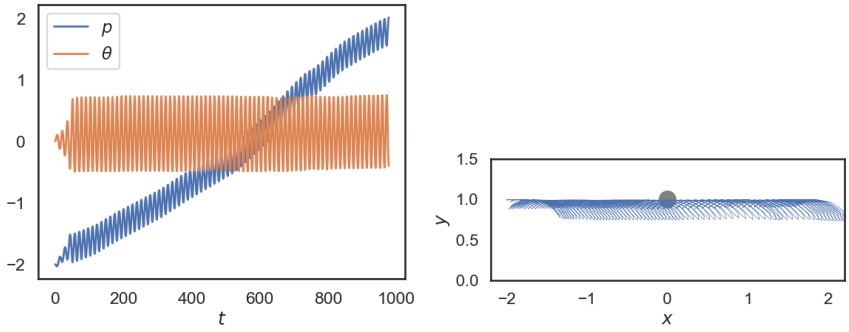

Figure 12: A plot of the CPO RL agent's performance on the Segway obstacle-avoidance task. Left: p and θ over time; Right: the path of the head of the Segway in the xy plane, showing collision between the Segway and the obstacle.

The contour plot of the learned V and the violation of the rCLBF condition are shown in Fig. 11. We set v and ω to zero, then sample p and θ to evaluate V. To make this plot more interpretable, we convert p and θ to the $x - y$ coordinates of the top of the Segway. Fig. 11 shows that the learned V has a wider safe and unsafe set than $\mathcal{X}_{\text{safe}}$ and $\mathcal{X}_{\text{unsafe}}$, which explains why the simulated rCLBF-QP trajectories give such a wide berth to the obstacle. The Segway learns to gradually tilt down when it is 0.5m away from the obstacle, instead of abruptly changing its angle when it is too close to the obstacle. In addition, in Fig. 11 (right) we observe only a minor violation of the rCLBF decrease condition; we have $dV/dt \leq 0$ for most of the plotted range.

Failure of Constrained Policy Optimization

We attempted to train a controller for this task using the constrained policy optimization reinforcement learning algorithm (CPO, [39]). We found that the RL agent was able to stabilize the Segway in the absence of obstacles, but it failed to stabilize the system when an obstacle was included during training. An example plot of the trained controller's performance is shown in Fig. 12.

Pandemic Considerations

Our paper is concerned with synthesizing controllers for robotic systems. Due to facility access limitations from the COVID-19, we were not able to gather experimental results on hardware, so our paper focuses on experiments conducted in simulation. We took a number of steps to ensure that performance in our simulations correlates with expected performance in hardware. In particular,

1. We report evaluation times for all controllers used in our experiments and compare these times to the control frequency, allowing us to determine whether the algorithms could feasibly be deployed in real-time.
2. We randomly vary the values of model parameters while computing safety and error rates, simulating the uncertainty present in models of physical systems.
3. Our framework can be easily extended to include physical constraints, particularly actuator limits, within the QP-based controller.
4. One of our simulated examples (the neural lander) includes a learned model of aerodynamic ground effect, which uses experimental data to make the simulation more realistic by including otherwise unmodeled effects.

That said, there are a number of gaps between our simulation and reality. The most glaring gaps are that

1. Although our framework supports physical constraints, we do not present a thorough evaluation of the effect of varying actuator limits on controller performance.
2. We assume full information about the robot state, which in practice means that we assume a high-quality state estimate is available at a suitably high frequency.
3. We do not study the effects of delay on the stability or safety of our controller (beyond our measurement of control frequency).

In the coming months, we hope to carry out hardware demonstrations that justify these assumptions and close these gaps.