

Safe Optimal Control Using Stochastic Barrier Functions and Deep Forward-Backward SDEs

Marcus A. Pereira

Institute for Robotics and Intelligent Machines
Georgia Institute of Technology,
Atlanta, GA, 30332
mpereira30@gatech.edu

Ziyi Wang

School of Aerospace Engineering
Georgia Institute of Technology,
Atlanta, GA, 30332
ziyiwang@gatech.edu

Ioannis Exarchos

Department of Computer Science
Stanford University, Stanford, CA 94305
exarchos@stanford.edu

Evangelos A. Theodorou

School of Aerospace Engineering
Georgia Institute of Technology,
Atlanta, GA, 30332
evangelos.theodorou@gatech.edu

Abstract: This paper introduces a new formulation for stochastic optimal control and stochastic dynamic optimization that ensures safety with respect to state and control constraints. The proposed methodology brings together concepts such as Forward-Backward Stochastic Differential Equations, Stochastic Barrier Functions, Differentiable Convex Optimization and Deep Learning. Using the aforementioned concepts, a Neural Network architecture is designed for safe trajectory optimization in which learning can be performed in an end-to-end fashion. Simulations are performed on three systems to show the efficacy of the proposed methodology.

Keywords: Stochastic Optimal Control, Forward-Backward Stochastic Differential Equations, Stochastic Control Barrier Functions

1 Introduction

Recent advancements in the areas of stochastic optimal control theory and machine learning create new opportunities towards the development of scalable algorithms for stochastic dynamic optimization. Despite the progress, there is a scarcity of methodologies that have the characteristics of being solidly grounded in first principles, have the flexibility of deep learning algorithms in terms of representational power, and can be deployed on systems operating in safety critical scenarios.

Safety plays a major role in designing any engineering system in various industries ranging from automobiles and aviation to energy and medicine. With the rapid emergence of various advanced autonomous systems, the control systems community has investigated various techniques such as barrier methods [1], reachable sets [2], and discrete approximation [3] to ensure safety certifications. However, with the recent introduction of Control Barrier Functions (CBFs) [4, 5, 6], there has been a growing research interest in the community in designing controllers with verifiable safety bounds.

CBFs provide a measure of safety to a system given its current state. As the system approaches the boundaries of its safe operating region, the CBF value tends to infinity, leading to the name "barrier". References [1], [6] have implemented CBFs for deterministic systems in robotics such as bi-pedal walking on stepping stones. However, literature on CBFs for systems with stochastic disturbances is very scarce. Very recently, [7] introduced stochastic CBFs for relative degree 1 barrier functions, meaning that the function defining the safe set depends only on the states that are directly actuated. While the concept of stochastic CBFs is fairly recent, [7] only demonstrated their applicability to very simplified stochastic systems in conjunction with control Lyapunov functions. The idea of merging the concepts of CBFs with Control Lyapunov functions leads to some interesting results [4, 5, 6] in the sense of stability of the system. Another competitive approach to safety-critical control is that of Safe-RL [8, 9, 10]. Recent work [11] combine policy learning with barrier function learning, 4th Conference on Robot Learning (CoRL 2020), Cambridge MA, USA.

eliminating the problem of designing barrier functions. However, they do not guarantee *safety during learning the policy* and only address safety of the *learnt policy*.

In this paper, we propose to use Stochastic CBFs in a Safe-RL fashion. Additionally, we here propose to explore the inclusion of the concept of stochastic CBFs within the Stochastic Optimal Control (SOC) framework, which essentially leads to the problem of solving the Hamilton-Jacobi-Bellman (HJB) equation on a constrained solution set. Solving the HJB amounts to overcoming the curse of dimensionality. Popular solution methods in literature include iLQG [12], Path-Integral Control [13], and the Forward-Backward Stochastic Differential Equations (FBSDEs) framework [14], which tackle the HJB through locally optimal solutions. Of these, the algorithms based on FBSDEs are the most general in the sense that they neither require assumptions to simplify the HJB Partial Differential Equation (PDE) nor do they require Taylor’s approximations of the dynamics and value function [12]. More recently, with the introduction of deep learning methods to solve high-dimensional parabolic PDEs [15], deep learning based solutions of the HJB PDE using so-called Deep FBSDE controllers have emerged [16, 17, 18]. These algorithms leverage importance sampling using Girsanov’s theorem of change of measure [19, Chapter 5] for FBSDEs within deep learning models for sufficient exploration of the solution space. Additionally, they have been shown to scale to high-dimensional systems (101-states) and systems that are non-affine in controls [20].

Paralleling the work on deep learning-based SOC is deep learning-based optimization layers, which aims to increase the representational power of Deep Learning (DL) models. In [21], a differentiable optimization layer was introduced to explicitly learn nonlinear mappings characterized by a Quadratic Program (QP). Additionally, they introduce an efficient approach to compute gradients for backpropagation through an optimization layer using the KKT conditions. The latter allows such layers to be incorporated within Deep FBSDEs, which requires differentiability of all its subcomponents.

To the best of our knowledge, literature on combining SOC with Stochastic CBFs and differentiable convex optimization, all embedded within a deep model, is non-existent. Additionally, such an approach adds a sense of interpretability to the entire framework compared to the relative black box nature of deep neural networks used for end-to-end control. Our contributions are as follows:

- (i.) A novel, end-to-end differentiable architecture with embedded safety using Deep FBSDEs.
- (ii.) Safe-RL algorithm combining differentiable optimization layers and Deep FBSDEs.
- (iii.) An extension of existing theory on Stochastic Zeroing Control Barrier Functions (ZCBFs).

The rest of this paper is organized as follows: Section 2 goes over the problem formulation and Section 3 introduces the FBSDE framework. In Section 2.1, the stochastic CBF is presented, while the differentiable QP layer is summarized in Section 3.2. The algorithm and network architecture are explained in Section 3.3. The simulation results are included in Section 4. Finally, we conclude the paper and present some future directions in Section 5.

2 Mathematical Preliminaries and Problem formulation

We consider stochastic dynamical systems that are nonlinear in state and affine in control:

$$dx(t) = (f(x(t)) + G(x(t))u(t))dt + \Sigma(x(t))dw(t), \quad (1)$$

where $w(t)$ is a n_w dimensional standard Brownian motion, and $x \in \mathbb{R}^{n_x}$, and $u \in \mathbb{R}^{n_u}$ denote the state and control vectors, respectively. The functions $f : \mathbb{R}^{n_x} \rightarrow \mathbb{R}^{n_x}$, $G : \mathbb{R}^{n_x} \rightarrow \mathbb{R}^{n_x \times n_u}$ and $\Sigma : \mathbb{R}^{n_x} \rightarrow \mathbb{R}^{n_x \times n_w}$ represent the uncontrolled system drift dynamics, the actuator dynamics, and the diffusion matrix. We assume that the state space is divided into a *safe set* \mathcal{C} which consists of all states that are deemed safe for exploration, and its complement $\mathbb{R}^{n_x} \setminus \mathcal{C}$ denoting unsafe states.

2.1 Stochastic Control Barrier Functions

For the dynamical system defined by (1), safety is ensured if $x(t) \in \mathcal{C}$ for all t where the set \mathcal{C} defines the safe region of operation. \mathcal{C} is mathematically described by a locally Lipschitz function $h : \mathbb{R}^{n_x} \rightarrow \mathbb{R}$ [4] as:

$$\mathcal{C} = \{x : h(x) \geq 0\} \quad (2)$$

$$\partial\mathcal{C} = \{x : h(x) = 0\}. \quad (3)$$

The literature of CBFs makes use of two different kinds of control barrier functions [5]: *reciprocal* CBFs, whose value approaches infinity when x approaches the boundary of the safe set $\partial\mathcal{C}$, and

zeroing CBFs, whose value approaches zero close to the boundary. In this work we will focus on the latter kind. Without loss of generality, the function $h(\mathbf{x})$ which defines the safe set, is a zeroing CBF in itself and guarantees safety if for some class \mathcal{K} function¹ $\alpha(\cdot)$ the following inequality holds $\forall t$:

$$\frac{\partial h}{\partial \mathbf{x}} (\mathbf{f}(\mathbf{x}) + \mathbf{G}(\mathbf{x})\mathbf{u}) + \frac{1}{2} \text{tr} \left(\frac{\partial^2 h}{\partial \mathbf{x}^2} \boldsymbol{\Sigma}(\mathbf{x}) \boldsymbol{\Sigma}^T(\mathbf{x}) \right) \geq -\alpha(h(\mathbf{x})). \quad (4)$$

Above, the left-hand-side represents the rate of change of h while the right-hand-side provides a bound to that rate which depends on how close \mathbf{x} is to the boundary; essentially, the closer the state \mathbf{x} is to the boundary, the slower the value of h can further decrease, implying that \mathbf{x} may approach the boundary at ever-decreasing speed. The following theorem formalizes the safety guarantees:

Theorem 1. *Let the safe set \mathcal{C} be defined as per equations (2), (3) and let the initial condition $\mathbf{x}(0) \in \mathcal{C}$. Assume there exists a control process $\mathbf{u}(t')$ such that for all $t' < t$, equation (4) is satisfied. Then, for all $t' < t$ the process $\mathbf{x}(t)$ has remained within the safe set with probability 1, i.e., $\mathbb{P}(\mathbf{x}(t) \in \mathcal{C} \ \forall t' < t) = 1$.*

The proof is a generalization of Theorem 3 in [22], and is given in the Appendix. Note that for any given \mathbf{x} , eq. (4) essentially imposes a constraint on the values of control \mathbf{u} one can apply on the system if one wishes to maintain safety ($\mathbf{x} \in \mathcal{C}$).

2.2 Stochastic Optimal Control

The SOC problem is formulated in order to minimize the expected cost given as:

$$J(\mathbf{u}) = \mathbb{E}_{\mathbb{Q}} \left[\int_t^T (q(\mathbf{x}) + \frac{1}{2} \mathbf{u}^T \mathbf{R} \mathbf{u}) ds + \phi(\mathbf{x}(T)) \right], \quad (5)$$

subject to the stochastic dynamics given by (1), and the constraint that trajectories should remain in the safe set \mathcal{C} at all times. Here, $\phi : \mathbb{R}^{n_x} \rightarrow \mathbb{R}^+$ is the terminal state cost, $q : \mathbb{R}^{n_x} \rightarrow \mathbb{R}^+$ denotes the running state cost, and $\mathbf{R} \in \mathbb{S}_+^{n_u}$ where $\mathbb{S}_+^{n_u}$ represents the symmetric positive definite matrix space. Finally, \mathbb{Q} is the space of trajectories induced by the controlled dynamics in (1). We assume that all necessary technical requirements [23] regarding filtered probability space, Lipschitz continuity, regularity and growth conditions to guarantee existence and uniqueness of strong solutions to (1) are met. Letting $\mathcal{U}([0, T])$ be the space of admissible controls such that $\mathbf{x}(t)$ remains within \mathcal{C} over a fixed finite time horizon $T \in [0, \infty)$, the value function related to (5) is defined as

$$V(\mathbf{x}, t) = \inf_{\mathbf{u} \in \mathcal{U}([0, T])} J(\mathbf{u})|_{\mathbf{x}_0=\mathbf{x}, t_0=t},$$

and using Bellman's principle of optimality, one can derive the HJB PDE, given by

$$V_t + \inf_{\mathbf{u} \in \mathcal{U}([0, T])} \left[\frac{1}{2} \text{tr}(V_{\mathbf{x}\mathbf{x}} \boldsymbol{\Sigma} \boldsymbol{\Sigma}^T) + V_{\mathbf{x}}^T (\mathbf{f}(\mathbf{x}) + \mathbf{G}(\mathbf{x})\mathbf{u}) + q(\mathbf{x}) + \frac{1}{2} \mathbf{u}^T \mathbf{R} \mathbf{u} \right] = 0, \quad V(\mathbf{x}, T) = \phi(\mathbf{x}), \quad (6)$$

where we drop dependencies for brevity and use subscripts to indicate partial derivatives with respect to time and the state vector. The term inside the infimum operation defines the Hamiltonian:

$$\mathcal{H}(t, \mathbf{x}, \mathbf{u}, V_{\mathbf{x}}, V_{\mathbf{x}\mathbf{x}} \boldsymbol{\Sigma} \boldsymbol{\Sigma}^T) = \frac{1}{2} \text{tr}(V_{\mathbf{x}\mathbf{x}} \boldsymbol{\Sigma} \boldsymbol{\Sigma}^T) + V_{\mathbf{x}}^T (\mathbf{f}(\mathbf{x}) + \mathbf{G}(\mathbf{x})\mathbf{u}) + q(\mathbf{x}) + \frac{1}{2} \mathbf{u}^T \mathbf{R} \mathbf{u}.$$

Note that in this case the Hamiltonian is quadratic with respect to \mathbf{u} ; if the constraint on \mathbf{u} in order to remain within the safe set \mathcal{C} (eq. (4)) were not present, the optimal control could be calculated by setting $\partial \mathcal{H} / \partial \mathbf{u} = 0$, resulting in $\mathbf{u}^*(t, \mathbf{x}) = -\mathbf{R}^{-1} \mathbf{G}^T V_{\mathbf{x}}$. However, the CBF inequality constraint prevents such a closed-form solution for \mathbf{u}^* .

3 Safe Deep FBSDE Control

3.1 Deep FBSDE formulation

Using the nonlinear Feynman-Kac lemma, we can establish an equivalence between the HJB PDE and the following system of FBSDEs [23]: Given that a solution \mathbf{u}^* to the constrained minimization

¹A class \mathcal{K} function is a continuous, strictly increasing function $\alpha(\cdot)$ such that $\alpha(0) = 0$.

of \mathcal{H} exists, the unique solution of (6) corresponds to the following system of FBSDEs,

$$\begin{aligned} d\mathbf{x}(t) &= (\mathbf{f}(\mathbf{x}(t)) + \mathbf{G}(\mathbf{x}(t))\mathbf{u}^*(t)) dt + \Sigma(\mathbf{x}(t)) dw_t, & \mathbf{x}(0) = \mathbf{x}_0, & (\text{FSDE}) \\ dV(t) &= -(q(\mathbf{x}) + \frac{1}{2}\mathbf{u}^{*\top}(t)\mathbf{R}\mathbf{u}^*(t)) dt + V_{\mathbf{x}}^T(t)\Sigma(\mathbf{x}(t)) dw_t, & V(T) = \phi(\mathbf{x}(T)), & (\text{BSDE}) \end{aligned} \quad (8)$$

$$\begin{aligned} \mathbf{u}^*(t) &= \arg \min_{\mathbf{u}} \mathcal{H}(t) = \arg \min_{\mathbf{u}} \left\{ V_{\mathbf{x}}^T(t)\mathbf{G}(\mathbf{x}(t))\mathbf{u} + \frac{1}{2}\mathbf{u}^T\mathbf{R}\mathbf{u} \right\}, & (\text{QP}) \\ \text{s.t. } & \frac{\partial h}{\partial \mathbf{x}}(\mathbf{f}(\mathbf{x}(t)) + \mathbf{G}(\mathbf{x}(t))\mathbf{u}) + \frac{1}{2} \text{tr} \left(\frac{\partial^2 h}{\partial \mathbf{x}^2} \Sigma(\mathbf{x}(t)) \Sigma^T(\mathbf{x}(t)) \right) \geq -\alpha(h(\mathbf{x}(t))). \end{aligned} \quad (9)$$

Here, $V(t)$ is a shorthand notation for $V(\mathbf{x}(t), t)$ and denotes an evaluation of the function $V(\mathbf{x}, t)$ along a path of $\mathbf{x}(t)$, thus $V(t)$ is a stochastic process (same for $V_{\mathbf{x}}(t)$ and $V_{\mathbf{x}\mathbf{x}}(t)$). The above equations are time-discretized and (9) is a QP that needs to be solved for each time instant.

In the above system, $\mathbf{x}(t)$ evolves forward in time (due to its initial condition $\mathbf{x}(0) = \mathbf{x}_0$), whereas $V(\mathbf{x}(t), t)$ evolves backwards in time, due to its terminal condition $\phi(\mathbf{x}(T))$. However, a simple backward integration of $V(\mathbf{x}(t), t)$ would result in it depending explicitly on future values of noise (final term in eq. (8)), which is not desirable for a non-anticipating process, i.e., a process that does not exploit knowledge on future noise values. To avoid back-propagation of the BSDE, we utilize DL. Specifically, incorporating DL [16] allows us to randomly initialize the value function at start time by treating it as a trainable parameter, forward-propagate V , and use its deviation from the terminal condition as training loss signal to adjust the initial $V(\mathbf{x}(0), 0)$ and train an approximation of the gradient function $V_{\mathbf{x}}(\mathbf{x}, t)$. In this work, we utilize the Deep FBSDE controller architecture in [16] as our starting point and add a differentiable stochastic CBF layer to ensure safety and maintain end-to-end differentiability.

3.2 Differential Quadratic Programming Layer

In [21], the authors proposed a differentiable optimization layer capable of learning nonlinear mappings characterized by QPs of the form

$$\begin{aligned} \bar{\mathbf{u}} &= \underset{\mathbf{u}}{\operatorname{argmin}} \frac{1}{2} \mathbf{u}^T Q(\mathbf{u}_i) \mathbf{u} + q(\mathbf{u}_i)^T \mathbf{u} \\ \text{s.t. } & C(\mathbf{u}_i) \mathbf{u} \leq d(\mathbf{u}_i) \end{aligned} \quad (10)$$

where, \mathbf{u}_i is the QP layer's input, $\bar{\mathbf{u}}$ is the fixed point solution of the QP problem as well as the output of the QP layer, and the QP parameters are $Q : \mathbb{R}^{n_u} \rightarrow \mathbb{R}^{n_u \times n_u}$, $q : \mathbb{R}^{n_u} \rightarrow \mathbb{R}^{n_u}$, $C : \mathbb{R}^{n_u} \rightarrow \mathbb{R}^{n_q \times n_u}$, $d : \mathbb{R}^{n_u} \rightarrow \mathbb{R}^{n_q}$, and n_q is the number of inequality constraints.

Specifically, during the forward pass through the QP layer, the optimization problem is solved by running the Primal-Dual Interior Point Method (PDIPM) [24, Chapter 14] until convergence. PDIPM requires a feasible initialization and searches within solutions that satisfy the constraints.

Since the Deep FBSDE network is trained by backpropagating gradients through time, one needs to be able to pass gradients through QPs solved at each time step. During backpropagation, instead of relying on the auto-differentiation of forward computations, a linear system formulated from the KKT conditions that $\bar{\mathbf{u}}$ must satisfy is used to find the gradient of $\bar{\mathbf{u}}$ with respect to the QP parameters. In our approach, the differentiable QP layer is used to solve the QP problem in eq. (9). To match the QP in (10), we set $Q = R$, $q = V_{\mathbf{x}}^T \mathbf{G}$, $C = -\frac{\partial h}{\partial \mathbf{x}} \mathbf{G}$, $d = \alpha(h(\mathbf{x})) + \frac{\partial h}{\partial \mathbf{x}} \mathbf{f} + \frac{1}{2} \text{tr} \left(\frac{\partial^2 h}{\partial \mathbf{x}^2} \Sigma \Sigma^T \right)$. Unlike [21], the QP parameters are not trainable in our case, as they are pre-defined by our problem.

3.3 Deep FBSDEs for Safe SOC - Algorithm and Network Architecture

The time discretization scheme is same as that employed in [16, Section IV]. The network architecture of the safe FBSDE controller is presented in Fig. 1. The network consists of a FBSDE prediction layer (dark green), a safe layer (light green) and two forward propagation processes. The main differences between our proposed network and the one introduced in [16, Fig. 2] are: 1) the value function gradient at initial time comes from network prediction rather than random initialization, making the predictions more consistent temporally; 2) the propagated and true value function gradients are included in the loss function instead of only using the propagated and true value function. The safe

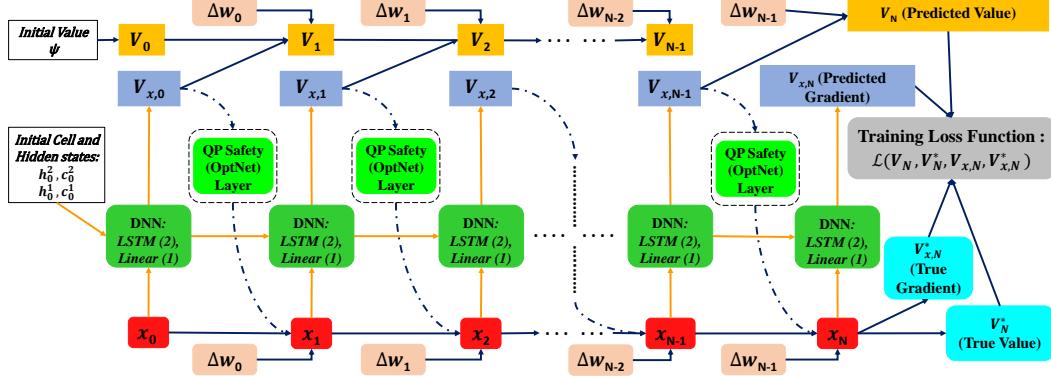


Figure 1: **Safe FBSDE Controller:** Note that the same trainable parameters of the DNN (2 LSTM + 1 Linear) layers are shared across all time steps, and the same noise is used to propagate the value function, V and the system state, \mathbf{x} . Additionally, the initial value, cell and hidden states are trainable.

layer, consisting of the differentiable QP OptNet layer [21], solves the constrained Hamiltonian minimization problem (9). The differentiability of components in the safe layer ensures that the entire network is end-to-end trainable (gradient can flow through all connections in Fig. 1). A summary of the algorithm is given in Alg. 1 and 2 in the Supplementary Material. Given an initial state, the value function gradient at that state is approximated by the FBSDE layer prediction and is used to construct the constrained Hamiltonian that is solved by the safe layer, as detailed in Alg. 2. The safe layer formulates the constrained Hamiltonian minimization as a QP and solves for the safe control using the KKT conditions and PDIPM interior point solver. For further details of PDIPM see [21, 25]. Both state \mathbf{x} and value function V are then propagated forward with the computed safe control. At terminal time T , the true value function and its gradient are calculated from the terminal state and compared against their propagated counterparts. The loss function is constructed using the difference between propagated and true value function, the true value function, the difference between propagated and true value function gradient, and the true value function gradient (equation in Alg. 1). Training is done using the Adam [26] optimizer.

4 Simulations

In this section we provide details of the successful application of our proposed Safe FBSDE controller on three different systems: pendulum, cart-pole and 2D car in simulation. These are conducted in a safe RL setting meaning we learn an optimal controller while never leaving the safe set *during the entire training process*. We have included a comprehensive description of each system in the supplementary material (SDEs in state-space form, CBFs and their Lie-derivatives as required in (10) for the Hamiltonian minimization as well as values for hyperparameters for training the deep networks). In all simulations, the results are compared to an *unsafe* FBSDE controller [16] without any CBF corrective action. It is important to note that safety must be included during training, rather than applied as a corrective layer to an unsafely trained network. This is because Deep FBSDE controllers learn the value function in those parts of the state space that are visited by the sampled trajectories during training. If one were to train unsafe FBSDEs and then apply a CBF correction during execution, this would induce trajectories to exit the “familiar” state space area, resulting in inaccurate value function extrapolation and thus non-optimal, unpredictable behavior. The barriers functions used in our simulations take a generic form of

$$h(\mathbf{x}) = (\text{position constraint}) - \mu(\text{velocity})^2,$$

$$\text{where } (\text{position constraint}) \begin{cases} > 0, & \forall \mathbf{x} \in \mathcal{C} \setminus \partial \mathcal{C} \\ = 0, & \forall \mathbf{x} \in \partial \mathcal{C} \end{cases} .$$

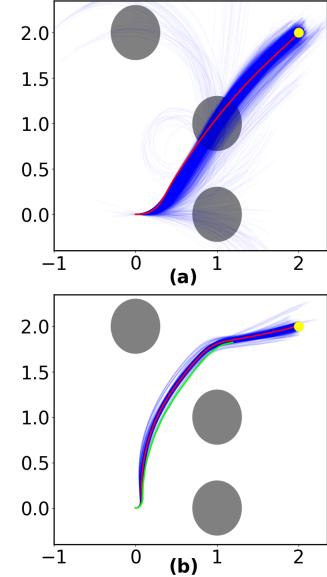


Figure 2: **Obstacle avoidance:** (a) and (b) represent *unsafe* and *safe* respectively. Policy is learnt avoiding obstacles (grey) during training (blue) to reach the target (yellow). Mean-test is shown in (red) and worst-case in (lime).

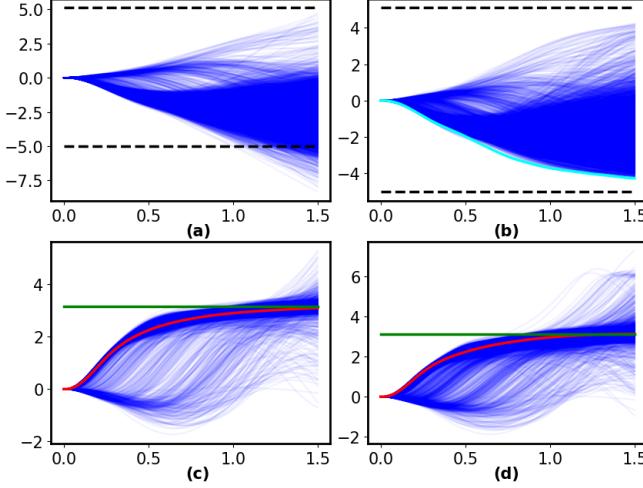


Figure 3: **Cart-pole Swing-up:** The horizontal axis represents *time (s)* in all subplots while vertical axis represents *cart-position (m)* in (a) and (b) and *pole-angle (rad)* in (c) and (d). Plots (a) and (c) show *unsafe* while (b) and (d) show *safe* trajectories. The safe controller learns to perform a swing-up while always remaining within the bounds (dashed *black* lines) during training (shown by *blue* trajectories). The mean test performance is shown in *red*, target pole-angle is shown in *green* and the worst-case (i.e. closest to either bound during training) is shown in *cyan*.

The parameter μ controls *how fast the system can move inside the safe set*. The above formulation causes the barrier function to have a relative degree of 1. For details regarding the *position constraint*, which is very task specific, we refer the interested reader to the Supplementary Material. We used time discretization of $\Delta t = 0.02s$ for all systems in all our simulations.

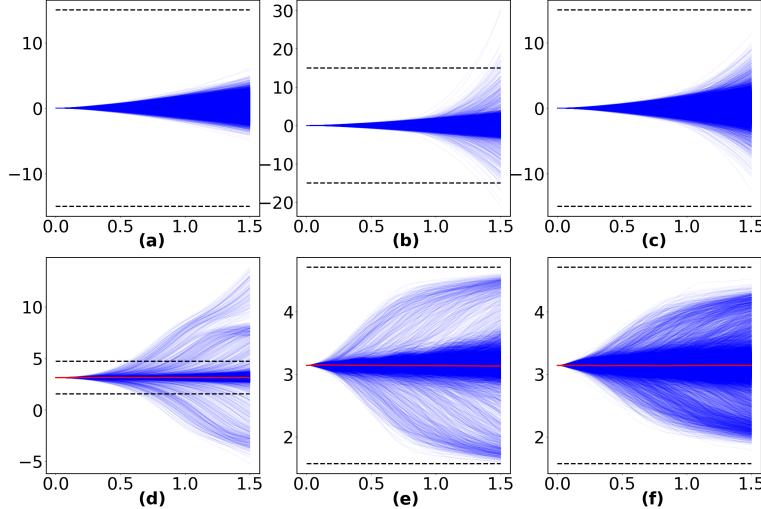


Figure 4: **Cart-pole Balancing:** The horizontal axis is *time (s)* in all subplots. Upper row plots are *cart-position (m)* and bottom row plots are *pole angular-position (rad)*. Plots (a) and (d) are *unsafe* trajectories, plots (b) and (e) are *one constraint safe* trajectories and plots (c) and (f) are *double constraint safe* trajectories. The safety bounds are indicated by dashed *black* lines and mean performance during testing in shown in *red*. Since all 25,728 training trajectories are plotted, there is no need to explicitly show the worst-case trajectory.

4.1 Pendulum Balancing

The states for this system are given by $\mathbf{x} = [\theta, \dot{\theta}]^T$ representing pendulum angle and its velocity. This task requires starting at the unstable equilibrium point of $\mathbf{x} = [\pi, 0]^T$ and balancing around it for a time horizon of 1.5 seconds. Due to stochasticity, the pendulum is perturbed from this initial

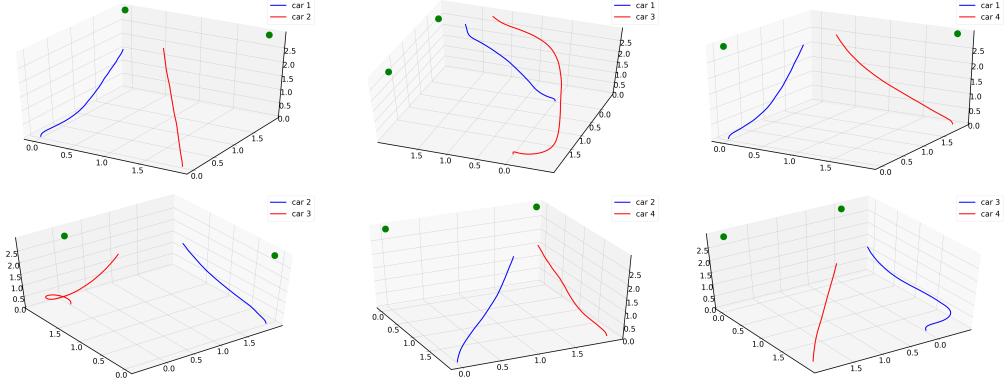


Figure 6: 2D Car Collision Avoidance: Figures above show worst-case (i.e. closest encounter) between each pair of cars during training. By design of the barrier function, each pair can be at 2 car radii from each other with zero velocity. This is seen above as the cars come to a halt much before reaching their targets. Vertical axes represent *time (s)* and horizontal axes represent (x, y) position.

position causing it to fall to the stable equilibrium point $\mathbf{x} = [0, 0]^T$. To enforce safety, we impose box constraints on the angular position of the pendulum of $\theta \in [2\pi/3, 4\pi/3]$. The results from our simulation are shown in Fig. 5 wherein all the 25,728 trajectories encountered during training are plotted in *blue* and compared with an unsafe solution. Our proposed controller Fig. 5(b) is able to match the unsafe controller Fig. 5(a) on average during testing (*red*), while respecting the imposed angular bounds during the entire training process.

4.2 Cart-pole

The states of this system are given by $\mathbf{x} = [x_c, \theta, \dot{x}_c, \dot{\theta}]^T$ representing the cart-position, pole angular-position, cart-velocity and pole angular-velocity respectively. We considered the following 3 different tasks (the time horizon for all tasks is 1.5 seconds):

4.2.1 Swing-up with bounds on cart-position: This task requires starting from an initial state of $\mathbf{x} = [0, 0, 0, 0]^T$ and finding a sequence of control actions to reach a final state of $\mathbf{x} = [0, \pi, 0, 0]^T$. To enforce safety, we impose constraints on the cart-position of $[-5.0 \text{ m}, 5.0 \text{ m}]$. The training (*blue*) and average testing (*red*) trajectories from our simulations are shown in Fig. 3. Since we cannot plot all 256,128 training trajectories we randomly sample 10,000 for plotting. However, we also plot the worst-case trajectory encountered (i.e. trajectory that gets closest to one of the boundaries) to emphasize that no trajectory violates the safety constraint *during the entire training process*.

4.2.2. Pole-balancing: Here we explored two sub-tasks (see Fig. 4):

With bounds on pole-angle: The pole has to be stabilized at the unstable equilibrium point of $\theta = \pi$ rads. The initial state is $\mathbf{x} = [0, \pi, 0, 0]^T$. Due to stochasticity, the pole is displaced off of $\theta = \pi$ and falls toward the stable equilibrium point of $\theta = 0$. In this case (referred to as *one constraint* case in Fig. 4(b) and Fig. 4(e)), the safe controller achieves the desired average performance during testing (*red*) while never leaving the safe set of $\theta \in [\pi/2, 3\pi/2]$ during training. Since the bounds on cart-position were not enforced, trajectories go beyond the limits as expected in Fig. 4(b).

With bounds on both cart-position and pole-angle: The task here (referred to as *double constraint* case in Fig. 4(c) and Fig. 4(f)) is identical to the one above, except now in addition to bounds on θ , the cart-position is constrained to $x_c \in [-15 \text{ m}, 15 \text{ m}]$. Compared to the single constraint case, the safe controller now respects both bounds during training.

Notice the slight similarity (heavy concentration around origin) in Fig. 4(a) and Fig. 4(c), while Fig. 4(b) vastly differs. This is because, in the *unsafe* Fig. 4(a), the optimal controller is allowed to let the pole fall beyond the safe bounds which does not encourage it to move the cart further away from the origin “*to save the pole from falling*” like in Fig. 4(b). In Fig. 4(c), because of additional restrictions, the controller is forced to rapidly move the cart back-and-forth within the allowable

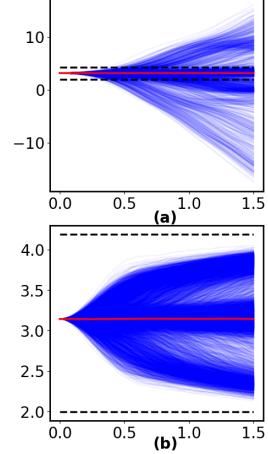


Figure 5: Pendulum Balancing: (a) and (b) represent *safe* and *unsafe* trajectories respectively.

bounds to stay safe. Comparing Fig. 4(e) and Fig. 4(f) we see the range of θ covered is reduced in the *double constraint* case. Through this example we would like to highlight the adaptability of the safe FBSDE controller to find the best solution while respecting the imposed constraints.

4.3 Two-dimensional Car navigation

For each 2D car system, the state vector is given by $\mathbf{x} = [p_x, p_y, \theta, v]^T$ representing the x-position, y-position, heading angle (w.r.t x-axis) and the forward velocity of the car respectively. We consider the following two tasks:

4.3.1. Single Car Multiple Obstacle Avoidance: In this task, a single car has to navigate from an initial state of $\mathbf{x} = [0, 0, 0, 0]^T$ to a final state of $\mathbf{x} = [2, 2, 0, 0]^T$ while avoiding three obstacles shaped as circles whose centers are located at $(1, 1)$, $(1, 0)$ and $(0, 2)$ each with a radius 0.3 in a time horizon of 3 seconds. The simulation results are shown in Fig. 2. Of the 256, 256 trajectories encountered during training, 10,000 are randomly sampled and plotted as *blue* lines. However, to emphasize that safety constraints *are not violated during training*, we plot the worst-case trajectory (i.e. the trajectory that gets closest to any one of the three obstacles during training) in *lime*. Notice that the optimal policy does not choose to be very close to the obstacle. This is due to the barrier function design, which allows the car to be at the obstacle boundary only if its velocity is zero. Therefore, getting closer to the boundary, slows the car down causing it to either not reach the target in the given time horizon or reach the target *late* thereby accumulating more running cost. We hypothesize that during the training process, the learnt optimal cost-to-go (value function) of states very close to the obstacle are high and therefore the optimal controller chooses to stay away from the states close to the obstacle boundary. We hypothesize this also being the reason for the optimal policy choosing to go through the larger gap in between the upper and middle obstacle than the lower gap between the middle and lower obstacle.

4.3.2. Multi-Car Collision Avoidance: We finally tested our framework in a dynamic obstacle setting. In this task, 4 cars start at 4 corners of a square grid ($2 \text{ m} \times 2 \text{ m}$) and are tasked with swapping their positions with the diagonally opposite car. To ensure safety, each car can be at a minimum distance of 0.1 m from each other. Thus, the goal is to swap positions by avoiding collisions. We used $\binom{4}{2} = 6$ barrier functions for each car pair. Since all training trajectories cannot be plotted, we find the worst-cases (i.e. closest approach) between all car pairs during training. As seen in Fig. 6, the cars stop short of their respective targets (*green* spheres) because they get too close to each other. This behavior stems from our barrier function design, which allows the cars to be at 0.1 m from each other but with zero velocity. We hypothesize that these states get associated with high value (high cost-to-go) which the optimal controller learns to avoid. The performance during testing is shown in Fig. 7 wherein each car successfully reaches its target on average. Please refer to the supplementary for 3D plots during testing to see successful collision avoidance. An animation² demonstrating the progress of the policy through the training process is available on YouTube.

5 Conclusions and Future Directions

In this paper we introduced a novel approach to safe reinforcement learning for stochastic dynamical systems. Our framework is the first to integrate Stochastic CBFs into the deep FBSDE controller framework for end-to-end safe optimal control. We extended the existing theory on Stochastic ZCBFs, developed a numerical algorithm combining deep FBSDEs, stochastic ZCBFs and the differentiable convex optimization layer (OptNet) frameworks, and successfully demonstrated its application on three systems in simulation. This initial success motivates a few directions for further investigation. Namely, we would like to explore high relative degree stochastic CBFs for underactuated systems, and combine this framework for systems in simulators for which explicit equations of motion are not available. Further directions include extending the framework to handle model uncertainty and analyzing the robustness capabilities of the controller. We believe these would serve as stepping stones to be able to deploy this algorithm on real hardware in the future.

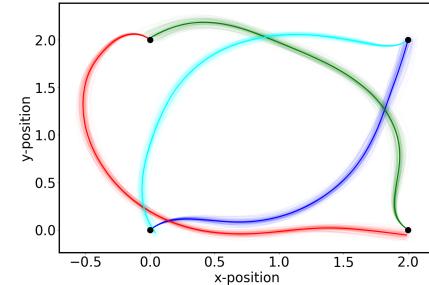


Figure 7: **Collision Avoidance:** Trajectories during testing learnt policy with darker lines indicating mean.

²<https://youtu.be/noonPzL39ic>

Acknowledgments

This work is supported by NASA Langley and the NSF-CPS award #1932288.

References

- [1] Q. Nguyen and K. Sreenath. Exponential control barrier functions for enforcing high relative-degree safety-critical constraints. In *2016 American Control Conference (ACC)*, pages 322–328. IEEE, 2016.
- [2] M. Althoff, C. Le Guernic, and B. H. Krogh. Reachable set computation for uncertain time-varying linear systems. In *Proceedings of the 14th International Conference on Hybrid Systems: Computation and Control, HSCC ’11*, page 93–102, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450306294. doi:10.1145/1967701.1967717. URL <https://doi.org/10.1145/1967701.1967717>.
- [3] S. Mitra, T. Wongpiromsarn, and R. M. Murray. Verifying cyber-physical interactions in safety-critical systems. *IEEE Security Privacy*, 11(4):28–37, 2013.
- [4] A. D. Ames, J. W. Grizzle, and P. Tabuada. Control barrier function based quadratic programs with application to adaptive cruise control. In *53rd IEEE Conference on Decision and Control*, pages 6271–6278. IEEE, 2014.
- [5] A. D. Ames, X. Xu, J. W. Grizzle, and P. Tabuada. Control barrier function based quadratic programs for safety critical systems. *IEEE Transactions on Automatic Control*, 62(8):3861–3876, 2016.
- [6] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada. Control barrier functions: Theory and applications. In *2019 18th European Control Conference (ECC)*, pages 3420–3431. IEEE, 2019.
- [7] A. Clark. Control barrier functions for complete and incomplete information stochastic systems. In *2019 American Control Conference (ACC)*, pages 2928–2935. IEEE, 2019.
- [8] T.-H. Pham, G. De Magistris, and R. Tachibana. Optlayer-practical constrained optimization for deep reinforcement learning in the real world. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6236–6243. IEEE, 2018.
- [9] S. Gros, M. Zanon, and A. Bemporad. Safe reinforcement learning via projection on a safe set: How to achieve optimality? *arXiv preprint arXiv:2004.00915*, 2020.
- [10] K. P. Wabersich, L. Hewing, A. Carron, and M. N. Zeilinger. Probabilistic model predictive safety certification for learning-based control. *arXiv preprint arXiv:1906.10417*, 2019.
- [11] W. Jin, Z. Wang, Z. Yang, and S. Mou. Neural certificates for safe control policies. *arXiv preprint arXiv:2006.08465*, 2020.
- [12] E. Todorov and W. Li. A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *Proceedings of the 2005, American Control Conference, 2005.*, pages 300–306. IEEE, 2005.
- [13] E. Theodorou, J. Buchli, and S. Schaal. A generalized path integral control approach to reinforcement learning. *journal of machine learning research*, 11(Nov):3137–3181, 2010.
- [14] I. Exarchos and E. A. Theodorou. Stochastic optimal control via forward and backward stochastic differential equations and importance sampling. *Automatica*, 87:159–165, 2018.
- [15] J. Han, A. Jentzen, and E. Weinan. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510, 2018.
- [16] M. A. Pereira, Z. Wang, I. Exarchos, and E. A. Theodorou. Learning deep stochastic optimal control policies using forward-backward sdes. In *Robotics: science and systems*, 2019.

- [17] Z. Wang, K. Lee, M. A. Pereira, I. Exarchos, and E. A. Theodorou. Deep forward-backward sdes for min-max control. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pages 6807–6814. IEEE, 2019.
- [18] M. Pereira, Z. Wang, T. Chen, E. Reed, and E. Theodorou. Feynman-kac neural network architectures for stochastic control using second-order fbsde theory. 2020.
- [19] S. E. Shreve. *Stochastic calculus for finance II: Continuous-time models*, volume 11. Springer Science & Business Media, 2004.
- [20] I. Exarchos, M. A. Pereira, Z. Wang, and E. A. Theodorou. Non-convex optimization via adaptive stochastic search for end-to-end learning and control. *arXiv preprint arXiv:2006.11992*, 2020.
- [21] B. Amos and J. Z. Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 136–145. JMLR.org, 2017.
- [22] A. Clark. Control barrier functions for stochastic systems. *arXiv preprint arXiv:2003.03498*, 2020.
- [23] J. Yong and X. Y. Zhou. *Stochastic controls: Hamiltonian systems and HJB equations*, volume 43. Springer Science & Business Media, 1999.
- [24] J. Nocedal and S. Wright. *Numerical optimization*. Springer Science & Business Media, 2006.
- [25] J. Mattingley and S. Boyd. Cvxgen: A code generator for embedded convex optimization. *Optimization and Engineering*, 13(1):1–27, 2012.
- [26] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [27] I. Karatzas and S. E. Shreve. Brownian motion. In *Brownian Motion and Stochastic Calculus (2nd ed.)*. Springer, 1998.
- [28] Z. Xie, C. K. Liu, and K. Hauser. Differential dynamic programming with nonlinear constraints. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 695–702, 2017.
- [29] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>.

6 Supplementary Material

6.1 Proof of Theorem 1

Theorem 1 is a generalization of Theorem [3] in [22], the proof of which is mimicked in construction here. We want to show that for any $t > 0$, any $\epsilon > 0$, and any $\delta \in (0, 1)$,

$$\mathbb{P}\left(\inf_{t' < t} h(\mathbf{x}_{t'}) \leq -\epsilon\right) \leq \delta,$$

where the subscript t denotes dependence on time. Let $\theta = \min\{\alpha^{-1}(\frac{\delta\epsilon}{2t}), h(\mathbf{x}_0)\}$. Note that for $t = 0$ this reduces to

$$h(\mathbf{x}_0) = \theta. \quad (11)$$

Itô's lemma then implies that

$$h(\mathbf{x}_t) = h(\mathbf{x}_0) + \int_0^t \frac{\partial h}{\partial \mathbf{x}}(\mathbf{f}(\mathbf{x}_\tau) + \mathbf{G}(\mathbf{x}_\tau)\mathbf{u}) + \frac{1}{2} \text{tr}\left(\frac{\partial^2 h}{\partial \mathbf{x}^2} \Sigma \Sigma^\top\right) d\tau + \int_0^t \Sigma(\mathbf{x}_\tau) \frac{\partial h}{\partial \mathbf{x}} d\mathbf{w}_\tau. \quad (12)$$

Consider now θ as a particular value for $h(\mathbf{x})$. As $h(\mathbf{x})$ varies, it may cross this value from above or from below. We call the times in which this occurs *stopping times*; in particular, we have stopping times η_i whenever $h(\mathbf{x}_t)$ down-crosses the value θ (i.e., coming from above) and ζ_i whenever $h(\mathbf{x}_t)$ up-crosses the value θ (i.e., coming from below). The sequence is defined as follows:

$$\begin{aligned} \eta_0 &= 0, & \zeta_0 &= \inf\{t : h(\mathbf{x}_t) > \theta\}, \\ \eta_i &= \inf\{t : h(\mathbf{x}_t) < \theta, t > \zeta_{i-1}\}, & \zeta_i &= \inf\{t : h(\mathbf{x}_t) > \theta, t > \eta_{i-1}\}, \end{aligned}$$

for $i = 1, 2, \dots$. Note that since $h(\mathbf{x}_0) = \theta$, we take $\eta_0 = 0$ by convention. We now proceed to construct a random process U_t with $U_0 = \theta$ as follows:

$$U_t = \theta + \sum_{i=0}^{\infty} \left(\int_{\eta_i \wedge t}^{\zeta_i \wedge t} -\alpha(\theta) d\tau + \int_{\eta_i \wedge t}^{\zeta_i \wedge t} \Sigma \frac{\partial h}{\partial \mathbf{x}} d\mathbf{w}_\tau \right), \quad (13)$$

in which $a \wedge b$ denotes the minimum between a and b . For $s < t$, this process satisfies

$$\begin{aligned} \mathbb{E}[U_t | U_s] &= U_s + \mathbb{E}\left[\sum_{i=0}^{\infty} \left(\int_{(\eta_i \wedge t) \vee s}^{(\zeta_i \wedge t) \vee s} -\alpha(\theta) d\tau + \int_{(\eta_i \wedge t) \vee s}^{(\zeta_i \wedge t) \vee s} \Sigma \frac{\partial h}{\partial \mathbf{x}} d\mathbf{w}_\tau \right) \right] \\ &= U_s + \mathbb{E}\left[\sum_{i=0}^{\infty} \int_{(\eta_i \wedge t) \vee s}^{(\zeta_i \wedge t) \vee s} -\alpha(\theta) d\tau \right] \leq U_s, \end{aligned}$$

where $a \vee b$ denotes the maximum between a and b , and $(\eta_i \wedge t) \vee s$ is used to avoid integration during times up to s , which are already included in U_s . This inequality implies that the stochastic process U_t is a *supermartingale*.

We will now show that (a). $h(\mathbf{x}_t) \geq U_t$ and (b). $U_t \leq \theta$ for all t . This will be done by induction. For $t = \eta_0 = 0$, both statements hold, because $h(\mathbf{x}_0) = \theta$ as per (11), and $U_0 = \theta$ by construction (eq. (13)). Now, suppose that the statements hold up to time $t = \eta_i$ for some $i \geq 0$. We will show that they remain true for the interval $[\eta_i, \zeta_i]$ first, and then that they remain true for the interval $[\zeta_i, \eta_{i+1}]$, which would complete the induction. For $t \in [\eta_i, \zeta_i]$ we have

$$\begin{aligned} h(\mathbf{x}_t) &= h(\mathbf{x}_{\eta_i}) + \int_{\eta_i}^t \frac{\partial h}{\partial \mathbf{x}}(\mathbf{f}(\mathbf{x}_\tau) + \mathbf{G}(\mathbf{x}_\tau)\mathbf{u}) + \frac{1}{2} \text{tr}\left(\frac{\partial^2 h}{\partial \mathbf{x}^2} \Sigma \Sigma^\top\right) d\tau + \int_{\eta_i}^t \Sigma(\mathbf{x}_\tau) \frac{\partial h}{\partial \mathbf{x}} d\mathbf{w}_\tau, \\ U_t &= U_{\eta_i} + \int_{\eta_i}^t -\alpha(\theta) d\tau + \int_{\eta_i}^t \Sigma(\mathbf{x}_\tau) \frac{\partial h}{\partial \mathbf{x}} d\mathbf{w}_\tau. \end{aligned}$$

We perform comparison by terms: for the first terms, $h(\mathbf{x}_{\eta_i}) \geq U_{\eta_i}$ by the main assumption of our induction process. The third terms are common, and the integrands of the second terms relate as follows:

$$\frac{\partial h}{\partial \mathbf{x}}(\mathbf{f}(\mathbf{x}) + \mathbf{G}(\mathbf{x})\mathbf{u}) + \frac{1}{2} \text{tr}\left(\frac{\partial^2 h}{\partial \mathbf{x}^2} \Sigma \Sigma^\top\right) \geq -\alpha(h(\mathbf{x})) \geq -\alpha(\theta),$$

where the first inequality is due to the CBF inequality (4) and the second inequality is by definition of stopping times η_i and ζ_i and monotonicity of $\alpha(\cdot)$: in the interval $[\eta_i, \zeta_i]$ we have that $h(\mathbf{x}_t) \leq \theta$.

Thus, for the interval $[\eta_i, \zeta_i]$, we showed that $h(\mathbf{x}_t) \geq U_t$, and as a corollary, by virtue of the stopping time definition, we furthermore have $U_t \leq \theta$. We proceed with the interval $t \in [\zeta_i, \eta_{i+1}]$. By construction, U_t remains constant during that interval and we have

$$U_t = U_{\zeta_i} \leq h(x_{\zeta_i}) = \theta \leq h(\mathbf{x}_t),$$

wherein the first inequality holds as a result of the previous interval ($t \in [\eta_i, \zeta_i]$), and the second equality and inequality hold by virtue of the definition of the stopping times ζ_i and η_{i+1} . This completes the induction and we have thus proved that for all t , we have $h(\mathbf{x}_t) \geq U_t$ and $U_t \leq \theta$.

Since $U_t \leq h(\mathbf{x}_t)$, it follows that

$$\mathbb{P}\left(\inf_{t' < t} h(\mathbf{x}_{t'}) \leq -\epsilon\right) \leq \mathbb{P}\left(\inf_{t' < t} U_{t'} \leq -\epsilon\right).$$

By Doob's supermartingale inequality [27] we have that for $\epsilon > 0$

$$\epsilon \mathbb{P}\left(\inf_{t' < t} U_{t'} \leq -\epsilon\right) \leq \mathbb{E}[U_t^+] - \mathbb{E}[U_t],$$

wherein $a^+ = \max\{a, 0\}$. Taking $s = \max\{i : \eta_i < t\}$, we can express $\mathbb{E}[U_t]$ as

$$\mathbb{E}[U_t] = \begin{cases} \theta - \alpha(\theta)\mathbb{E}[\sum_{i=0}^s (\zeta_i - \eta_i)], & t \in [\zeta_s, \eta_{s+1}], \\ \theta - \alpha(\theta)\mathbb{E}[t - \eta_s + \sum_{i=0}^{s-1} (\zeta_i - \eta_i)], & t \in [\eta_s, \zeta_s]. \end{cases}$$

Both expectations inside the bracket are bounded above by t , and thus $\mathbb{E}[U_t] \geq \theta - \alpha(\theta)t$. Furthermore, since $U_t \leq \theta$ we have $\mathbb{E}[U_t^+] \leq \theta$, so combining the two we obtain

$$\mathbb{E}[U_t^+] - \mathbb{E}[U_t] \leq \theta - (\theta - \alpha(\theta)t) = \alpha(\theta)t.$$

Thus,

$$\begin{aligned} \mathbb{P}\left(\inf_{t' < t} h(\mathbf{x}_{t'}) \leq -\epsilon\right) &\leq \mathbb{P}\left(\inf_{t' < t} U_{t'} \leq -\epsilon\right) \\ &\leq \alpha(\theta) \frac{t}{\epsilon} \leq \alpha(\alpha^{-1}(\frac{\delta\epsilon}{2t})) \frac{t}{\epsilon} = \frac{\delta\epsilon t}{2\epsilon t} < \delta, \end{aligned}$$

where we used that $\theta = \min\{\alpha^{-1}(\frac{\delta\epsilon}{2t}), h(\mathbf{x}_0)\}$ is no greater than $\alpha^{-1}(\frac{\delta\epsilon}{2t})$. The proof is concluded.

6.2 Relative Degree of Barrier Functions

Let $L_f h$ and $L_g h$ be the Lie derivatives of h , defined as $\frac{\partial h}{\partial \mathbf{x}} \mathbf{f}$ and $\frac{\partial h}{\partial \mathbf{x}} \mathbf{G}$, respectively. The relative degree r of a barrier function is defined such that $L_g L_f^{r-1} h(\mathbf{x}) \neq 0$ and $L_g h(\mathbf{x}) = L_g L_f h(\mathbf{x}) = L_g L_f^2 h(\mathbf{x}) = \dots = L_g L_f^{r-2} h(\mathbf{x}) = 0$, $\forall \mathbf{x}$. Note that relative degree 1 barrier functions are barrier functions that impose restrictions on directly actuated system states, as a result of which $L_g h(\mathbf{x}) \neq 0$.

6.3 Implementation Details

In all our simulations we pick the following form of the class \mathcal{K} function that bounds the rate of change of the barrier function, $\alpha(\cdot) = \gamma h(\mathbf{x})$, where γ is a fixed positive constant that changes the slope of the bounding function. Higher the value of γ , less conservative is the barrier function.

6.3.1 Inverted Pendulum

For the inverted pendulum system, the state vector is given by $\mathbf{x} = [\theta, \dot{\theta}]^\top$ indicating pendulum angular-position and pendulum angular-velocity. The system dynamics (f) vector and the actuation (G) matrix are given by,

$$f = \begin{bmatrix} \dot{\theta} \\ -\frac{b}{I}\dot{\theta} - \frac{g}{l} \sin \theta \end{bmatrix}, \quad G = \begin{bmatrix} 0, \frac{1}{I} \end{bmatrix}^\top$$

The stochastic dynamics are given by,

$$d\mathbf{x}(t) = f(\mathbf{x}) dt + G(\mathbf{x}) u dt + \Sigma(\mathbf{x}) dW(t)$$

where,

$$\Sigma = \begin{bmatrix} 0 & 0 \\ 0 & \sigma \end{bmatrix} \implies \Sigma \Sigma^T = \begin{bmatrix} 0 & 0 \\ 0 & \sigma^2 \end{bmatrix}$$

The parameter values used were: $b = 0.1$, $l = 0.5$ m, $g = 9.81$ m/s², $m = 2$ kg and $I = ml^2$.

Balancing with angle constraints: For this task, we impose box constraints on the pendulum angular position. The goal is to ensure that the pendulum does not fall outside of a predetermined safe region around the unstable equilibrium point of $\mathbf{x} = [\pi, 0]^T$. Although the pendulum is initialized at $\mathbf{x} = [\pi, 0]^T$, stochasticity in the dynamics will push it off this unstable equilibrium position and safe controller is then tasked to keep the pendulum inside the safe region. As stated earlier, the pendulum must remain inside the safe region *during the entire training process*.

Let θ_h and θ_l be the upper and lower bounds on the angular position. Let the corresponding barrier functions be $h_h(\mathbf{x}) = \theta_h - \theta$ and $h_l(\mathbf{x}) = \theta - \theta_l$ and the combined barrier function be $h(\mathbf{x}) = h_h(\mathbf{x}) \cdot h_l(\mathbf{x}) - \mu\dot{\theta}^2$, so that the safe set given by $\mathcal{C} = \{\mathbf{x} : h(\mathbf{x}) \geq 0\}$.

$$\begin{aligned} h(\mathbf{x}) &= (\theta_h - \theta) \cdot (\theta - \theta_l) - \mu\dot{\theta}^2 = \theta_h\theta - \theta_h\theta_l - \theta^2 + \theta_l\theta - \mu\dot{\theta}^2 \\ \frac{\partial h}{\partial \mathbf{x}} &= \begin{bmatrix} \theta_h - 2\theta + \theta_l \\ -2\mu\dot{\theta} \end{bmatrix} \implies \frac{\partial h^T}{\partial \mathbf{x}} G = \frac{-2\mu\dot{\theta}}{I} \\ \therefore C &= -\frac{\partial h^T}{\partial \mathbf{x}} G = \frac{2\mu\dot{\theta}}{I} \quad \dots \text{ for (10)} \\ \frac{\partial h^T}{\partial \mathbf{x}} f &= \dot{\theta}(\theta_h - 2\theta + \theta_l) + \left(\frac{-b}{I}\dot{\theta} - \frac{g}{l}\sin\theta \right) \cdot (-2\mu\dot{\theta}) \\ &= \dot{\theta}(\theta_h - 2\theta + \theta_l) + \left(\frac{b}{I}\dot{\theta} + \frac{g}{l}\sin\theta \right) \cdot (2\mu\dot{\theta}) \\ \therefore d &= \alpha(h(\mathbf{x})) + \frac{1}{2}\text{tr}\left(\frac{\partial^2 h}{\partial \dot{\theta}^2} \Sigma \Sigma^T\right) + \frac{\partial h^T}{\partial \mathbf{x}} f \quad \dots \text{ for (10)} \\ &= \alpha(h(\mathbf{x})) - \mu\sigma^2 + \frac{\partial h^T}{\partial \mathbf{x}} f \quad \dots \left(\because \frac{\partial^2 h}{\partial \dot{\theta}^2} = -2\mu \right) \end{aligned}$$

The above choice of barrier function is relative degree 1 because $L_g h = (\partial h / \partial \mathbf{x})^T g \neq 0$, except at $\dot{\theta} = 0$. However, given the task at hand, the only time this can occur is at initial condition $\mathbf{x} = [\pi, 0]^T$ or if the controller manages to stabilize the pendulum at the unstable equilibrium point and exactly cancel out all the stochasticity entering the system. If any one of these occur, since $\theta = \pi$ (and because we choose box-constraint like bounds eg. $(\theta_l, \theta_h) = (2\pi/3, 4\pi/3)$), we see that the gradient of the barrier function $\partial h / \partial \mathbf{x}$ is itself zero. This means that the safety constraint is trivially satisfied (as we have $0 \geq -\alpha(h(\mathbf{x}))$ in (4)). In these cases, the unconstrained optimal control, $\mathbf{u}^*(t, \mathbf{x}) = -\mathbf{R}^{-1} \mathbf{G}^T V_{\mathbf{x}}$, would be the solution to the QP problem.

Regarding values of hyper-parameters used to train the Safe FBSDE controller for this problem, we used the following,

#	Parameter Name	Parameter Value
1	batch size	128
2	training iterations	201
3	σ	1.0
4	learning rate	$1e^{-2}$
5	number of neurons per LSTM layer	16
6	optimizer	Adam
7	μ	0.05
8	γ	0.5

6.3.2 Cart Pole

For this system, the state vector $\mathbf{x} = [x_c, \theta, \dot{x}_c, \dot{\theta}]^T$, indicating cart-position, pole angular-position, cart-velocity and pole angular-velocity respectively. The system dynamics (f) vector and actuation (G) matrix are given by,

$$f = \begin{bmatrix} \dot{x}_c \\ \dot{\theta} \\ \frac{m_p \sin \theta (l\dot{\theta}^2 + g \cos \theta)}{m_c + m_p \sin^2 \theta} \\ \frac{-m_p l \dot{\theta}^2 \cos \theta \sin \theta - (m_c + m_p) g \sin \theta}{l(m_c + m_p \sin^2 \theta)} \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix}, \quad G = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \frac{m_c + m_p \sin^2 \theta}{l(m_c + m_p \sin^2 \theta)} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ g_3 \\ g_4 \end{bmatrix}$$

The stochastic dynamics are given by,

$$d\mathbf{x}(t) = f(\mathbf{x}) dt + G(\mathbf{x}) u dt + \Sigma(\mathbf{x}) dW(t)$$

where,

$$\Sigma(\mathbf{x}) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma & 0 \\ 0 & 0 & 0 & \sigma \end{bmatrix} \implies \Sigma(\mathbf{x}) \Sigma^T(\mathbf{x}) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma^2 & 0 \\ 0 & 0 & 0 & \sigma^2 \end{bmatrix}$$

The system parameters chosen were: $m_p = 0.01 \text{ kg}$, $m_c = 1.0 \text{ kg}$, $g = 9.81 \text{ m/s}^2$ and $l = 0.5 \text{ m}$.

- i) **Swing-up with cart position constraints:** Here the task is to begin from the initial position $\mathbf{x}_0 = [0, 0, 0, 0]^T$ and perform a swing-up to $\mathbf{x}_{\text{target}} = [0, \pi, 0, 0]^T$ within a fixed time horizon of 1.5 seconds. Additionally, there are constraints on the cart-position that need to be satisfied both during training (i.e. learning the policy) and testing (i.e. final deployment of the policy). These constraints are represented as $[x_{c,l}, x_{c,h}]$ where $x_{c,h} > x_{c,l}$. A choice of barrier function that combines both position and velocity constraints is as follows:

$$\tilde{h} = (x_{c,h} - x_c) \cdot (x_c - x_{c,l}) - \mu \dot{x}_c^2 = x_{c,h} x_c - x_{c,h} x_{c,l} - x_c^2 + x_c x_{c,l} - \mu \dot{x}_c^2$$

where the parameter μ controls *how fast* the cart is allowed to move in the interior of the safe set \mathcal{C} .

$$\begin{aligned} \frac{\partial \tilde{h}}{\partial \mathbf{x}} &= \begin{bmatrix} x_{c,h} - 2x_c + x_{c,l} \\ 0 \\ -2\mu \dot{x}_c \\ 0 \end{bmatrix} \implies \frac{\partial \tilde{h}}{\partial \mathbf{x}}^T G = \frac{-2\mu \dot{x}_c}{m_c + m_p \sin^2 \theta} \\ \therefore C &= -\frac{\partial \tilde{h}}{\partial \mathbf{x}}^T G \quad \text{and} \\ d &= \alpha(\tilde{h}(\mathbf{x})) + 0.5 \cdot \sigma^2 (-2\mu) + \frac{\partial \tilde{h}}{\partial \mathbf{x}}^T f \quad \dots \left(\frac{\partial^2 \tilde{h}}{\partial x_c^2} = -2\mu \right) \end{aligned}$$

The relative degree of this barrier is also 1 similar to the pendulum case above. Please see the explanation provided there. For this task we used, $\mu = 0.1$, $\gamma = 1.0$ and cart-position bounds of $(x_{c,l}, x_{c,h}) = (-15 \text{ m}, 15 \text{ m})$.

- ii) **Pole balancing with pole position constraints:** Similar to the construction above for position constrained swing-up, we have the following hybrid barrier function for the task of balancing with constrained pole angle,

$$h_\theta(\mathbf{x}) = (\theta_h - \theta) \cdot (\theta - \theta_l) - \mu_\theta \dot{\theta}^2 = \theta_h \theta - \theta_h \theta_l - \theta^2 + \theta \theta_l - \mu_\theta \dot{\theta}^2$$

$$\begin{aligned} \frac{\partial h_\theta}{\partial \mathbf{x}} &= \begin{bmatrix} 0 \\ \theta_h - 2\theta + \theta_l \\ 0 \\ -2\mu_\theta \dot{\theta} \end{bmatrix} \implies \frac{\partial h_\theta}{\partial \mathbf{x}}^T G = \frac{(-2\mu_\theta \dot{\theta}) \cdot (-\cos \theta)}{l(m_c + m_p \sin^2 \theta)} \\ \therefore C &= -\frac{\partial h_\theta}{\partial \mathbf{x}}^T G \quad \text{and} \end{aligned}$$

$$d = \alpha(h_\theta(\mathbf{x})) + 0.5 \cdot \sigma^2 (-2\mu_\theta) + \frac{\partial h_\theta}{\partial \mathbf{x}}^T f \quad \dots \left(\frac{\partial^2 h}{\partial \dot{\theta}^2} = -2\mu_\theta \right)$$

The relative degree of this barrier is also 1 similar to the pendulum case above. Please see the explanation provided there. For this task we used, $\mu = 0.1$, $\gamma = 1.0$ and pole angular-position bounds of $(\theta_l, \theta_h) = (\pi/2 \text{ rad}, 3\pi/2 \text{ rad})$.

- iii) **Pole balancing with both pole and cart position constraints:** For this task we consider two separate barrier functions i.e. two safety inequality constraints for cart-position and pole-angle. The barrier functions and corresponding terms of the inequality constraints are exactly the same as the two sub-tasks above. The bounds are also the same as those used in the above sub-tasks. The barrier function parameters used were $\mu_{x_c} = 0.001$, $\mu_\theta = 0.1$, $\gamma_{x_c} = 100$ and $\gamma_\theta = 0.5$.

Following are some of the common hyper-parameter values used for the above cart-pole simulations:

#	Parameter Name	Parameter Value
1	batch size	128
2	training iterations (tasks i.)	2001
3	training iterations (tasks ii. and iii.)	201
4	σ	1.0
4	learning rate	$1e^{-2}$
5	number of neurons per LSTM layer	16
6	optimizer	Adam

6.3.3 2-D Car

The model is same as that used in [28] and has the following dynamics:

$$\begin{aligned}\dot{x} &= v \cos(\theta), & \dot{y} &= v \sin(\theta) \\ \dot{\theta} &= vu^\theta, & \dot{v} &= u^v\end{aligned}$$

The state vector is $\mathbf{x} = [p_x, p_y, \theta, v]^T$ which corresponds to the x-position, y-position, heading angle and forward velocity of the car. The control vector is $\mathbf{u} = [u^\theta, u^v]^T$ which corresponds to the acceleration and steering-rate of the car. The system dynamics (f) vector and actuation (G) matrix are given by,

$$f = \begin{bmatrix} v \cos(\theta) \\ v \sin(\theta) \\ 0 \\ 0 \end{bmatrix}, \quad G = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ v & 0 \\ 0 & 1 \end{bmatrix}$$

The stochastic dynamics are given by,

$$d\mathbf{x} = f(\mathbf{x})dt + g(\mathbf{x})\mathbf{u}dt + \Sigma(\mathbf{x})dW$$

where

$$\Sigma(\mathbf{x}) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma & 0 \\ 0 & 0 & 0 & \sigma \end{bmatrix} \implies \Sigma(\mathbf{x})\Sigma^T(\mathbf{x}) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma^2 & 0 \\ 0 & 0 & 0 & \sigma^2 \end{bmatrix}$$

- i) **Single Car Multiple Obstacle Avoidance:** In this task a single car starts at $(p_x, p_y) = (0 \text{ m}, 0 \text{ m})$ with the goal to reach the target $(p_x, p_y) = (2 \text{ m}, 2 \text{ m})$ while avoiding obstacles shaped as circles. We use 1 barrier function for each obstacle which takes the following form:

$$\tilde{h}^i(\mathbf{x}) = (p_x - o_x^i)^2 + (p_y - o_y^i)^2 - o_r^i{}^2 - \mu v^2$$

where i is the obstacle index and o_x^i, o_y^i, o_r^i are the x-position, y-position and radius of the obstacle respectively. This is a relative degree 1 barrier function, which can be shown by

$$\begin{aligned}\frac{\partial \tilde{h}^i}{\partial \mathbf{x}} &= \begin{bmatrix} 2(p_x - o_x^i) \\ 2(p_y - o_y^i) \\ 0 \\ -2\mu v \end{bmatrix} \\ \implies \frac{\partial \tilde{h}^i}{\partial \mathbf{x}}^T G &= [0 \quad -2\mu v]\end{aligned}$$

The other terms needed to set up (10) can be calculated as:

$$\begin{aligned}\frac{\partial \tilde{h}^i}{\partial \mathbf{x}}^T f &= 2(p_x - o_x^i)v \cos(\theta) + 2(p_y - o_y^i)v \sin(\theta) \\ \frac{1}{2} \text{tr} \left(\frac{\partial^2 \tilde{h}^i}{\partial \mathbf{x}^2} \Sigma \Sigma^T \right) &= -\mu \sigma^2\end{aligned}$$

The QP can be set up with

$$\begin{aligned}C_i &= -\frac{\partial \tilde{h}^i}{\partial \mathbf{x}}^T G \\ d_i &= \alpha(\tilde{h}^i(\mathbf{x})) + \frac{1}{2} \text{tr} \left(\frac{\partial^2 \tilde{h}^i}{\partial \mathbf{x}^2} \Sigma \Sigma^T \right) + \frac{\partial \tilde{h}^i}{\partial \mathbf{x}}^T f\end{aligned}$$

ii) **Multi-car Collision Avoidance:** For multi-car collision avoidance, simply set the obstacle's positions in the previous section's barrier function to the other car's positions, the obstacle's size to be 2 times the car's size, and subtract an additional velocity of the other car. In our simulation example we used 4 cars for a total of 16 states, $\mathbf{x} = [p_{x_1}, p_{y_1}, \dots, \theta_4, v_4]^T$, dimensions and 8 control dimensions, $\mathbf{u} = [u_1^\theta, u_1^v, \dots, u_4^\theta, u_4^v]^T$, resulting in 6 barrier functions for each pair of vehicles. The initial conditions for the 4 cars are $[0, 0, \pi/4, 0.1]$, $[2, 0, 3\pi/4, 0.1]$, $[0, 2, -\pi/4, 0.1]$ and $[2, 2, -3\pi/4, 0.1]$ respectively. For example, the barrier between car 1 and car 2 looks like

$$\tilde{h}^{12}(\mathbf{x}) = (p_{x_1} - p_{x_2})^2 + (p_{y_1} - p_{y_2})^2 - (2o_c)^2 - \mu(v_1^2 + v_2^2)$$

This is a relative degree 1 barrier function, which can be shown by

$$\begin{aligned}\frac{\partial \tilde{h}^{12}}{\partial \mathbf{x}} &= \begin{bmatrix} 2(p_{x_1} - p_{x_2}) \\ 2(p_{y_1} - p_{y_2}) \\ 0 \\ -2\mu v_1 \\ -2(p_{x_1} - p_{x_2}) \\ -2(p_{y_1} - p_{y_2}) \\ 0 \\ -2\mu v_2 \\ \mathbf{0}_{8 \times 1} \end{bmatrix} \\ \implies \frac{\partial \tilde{h}^{12}}{\partial \mathbf{x}}^T G &= [0 \quad -2\mu v_1 \quad 0 \quad -2\mu v_2 \quad \mathbf{0}_{1 \times 4}]\end{aligned}$$

The other terms needed to set up (10) can be calculated as:

$$\begin{aligned}\frac{\partial \tilde{h}^{12}}{\partial \mathbf{x}}^T f &= 2(p_{x_1} - p_{x_2})v_1 \cos(\theta_1) + 2(p_{y_1} - p_{y_2})v_1 \sin(\theta_1) \\ &\quad - 2(p_{x_1} - p_{x_2})v_2 \cos(\theta_2) - 2(p_{y_1} - p_{y_2})v_2 \sin(\theta_2) \\ \frac{1}{2} \text{tr} \left(\frac{\partial^2 \tilde{h}^{12}}{\partial \mathbf{x}^2} \Sigma \Sigma^T \right) &= -2\mu \sigma^2\end{aligned}$$

The QP can be set up with

$$\begin{aligned}C_{12} &= -\frac{\partial \tilde{h}^{12}}{\partial \mathbf{x}}^T G \\ d_{12} &= \alpha(\tilde{h}^{12}(\mathbf{x})) + \frac{1}{2} \text{tr} \left(\frac{\partial^2 \tilde{h}^{12}}{\partial \mathbf{x}^2} \Sigma \Sigma^T \right) + \frac{\partial \tilde{h}^{12}}{\partial \mathbf{x}}^T f\end{aligned}$$

Regarding values of common hyper-parameters used to train the Safe FBSDE controllers for the 2D car problems, we used the following,

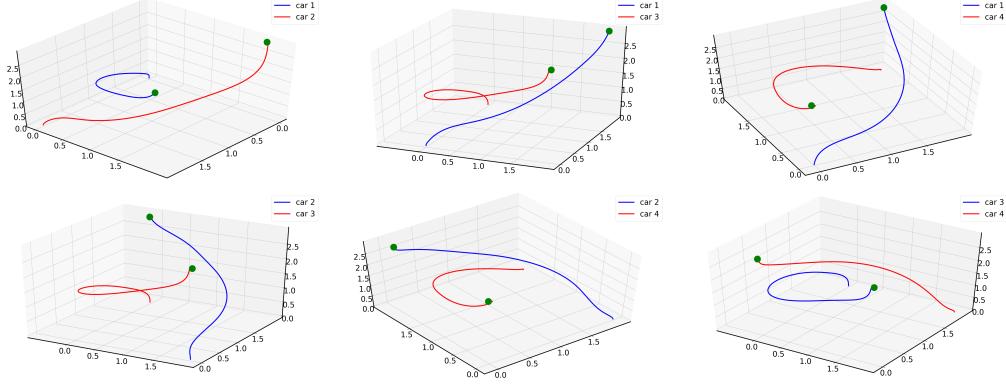


Figure 8: 2D Car Collision Avoidance Average Testing Performance: Figures above show successful collision avoidance performance of the trained policy averaged over 128 trials. Vertical axes represent *time (s)* and horizontal axes represent *(x, y) position*.

#	Parameter Name	Parameter Value
1	batch size (task i.)	128
2	batch size (task ii.)	256
3	training iterations	1000
4	σ	0.1
5	learning rate	$5e^{-3}$
6	number of neurons per LSTM layer	16
7	optimizer	Adam

For task (i.), we used $\mu = 0.05$ and $\gamma = 1$, while for task (ii.) we used, $\mu = 0.1$ and $\gamma = 1$.

6.4 Animation of Multi-Car Collision Avoidance

Attached with this supplementary material is an animated video of the multi-car collision avoidance problem. For this animation we used the first batch element (from a batch of 128) from the respective iterations seen in the video. However, when demonstrating performance during testing, we use average performance. As seen in the plot Fig.7, the variance of the trajectories is considerably small and therefore it suffices to show the mean performance. The mean cost is also the cost functional used in the problem formulation. The colored circles indicate the diagonally opposite targets for each car. The dotted circles show the 2 car radii minimum allowable safe distance.

6.5 Computational Resources

All the computational graphs were implemented in PyTorch [29] on a system with Intel(R) Xeon(R) CPU E5-1607 v3 @ 3.10GHz, 64GB RAM and a NVIDIA Quadro K5200 GPU.

6.6 Algorithms

The Alg.1 below, is very similar to work in [16], with the following changes: computation of the optimal control (Hamiltonian QP instead of closed form expression), propagation of the value function (no need for Girsanov's theorem), computation of an additional target for training the deep FBSDE network (i.e. the true gradient of the value function) and additional terms in the loss function. The colored circles in the video are the respective targets of each car.

In Alg.2, for details regarding **Affine Scale**, **Centering-Corrector**, **Residul** and **Step-Size** please see the primal-dual interior point method detailed in [24] or refer to code provided on github in [21].

Algorithm 1: Finite Horizon Safe FBSDE Controller

Given:

$\mathbf{x}_0 = \xi$, \mathbf{f} , \mathbf{G} , Σ : Initial state and system dynamics;
 ϕ , q , \mathbf{R} : Cost function parameters;
 a, b, c, d : Loss function parameters;
 N : Task horizon, K : Number of iterations, M : Batch size; Δt : Time discretization; λ : weight-decay parameter;

Parameters:

$\psi = V(\mathbf{x}_0, 0)$: Value function at $t = 0$;
 θ : Weights of all Linear and LSTM layers;

Initialize neural network parameters;

Initialize (for every batch element): $\{\mathbf{x}_0^i\}_{i=1}^M$, $\mathbf{x}_0^i = \xi$; $\{\psi_0^i\}_{i=1}^M$, $\psi_0^i = V(\mathbf{x}_0^i, 0)$

for $k = 1$ **to** K **do**

for $i = 1$ **to** M **do**

for $t = 1$ **to** $N - 1$ **do**

 Predict value function gradient:

$$V_{\mathbf{x},t}^i = f_{FBSDE}(\mathbf{x}_t^i; \theta^k)$$

 Solve constrained Hamiltonian minimization quadratic program:

$$\bar{\mathbf{u}}_t^{*i} = f_{safe}(\mathbf{x}_t^i, V_{\mathbf{x},t}^i)$$

 Sample Brownian noise:

$$\Delta \mathbf{w}_t^i \sim \mathcal{N}(0, \Sigma \Delta t)$$

 Propagate value function: $V_{t+1}^i = V_t^i - (q(\mathbf{x}_t) + \frac{1}{2} \bar{\mathbf{u}}_t^{*i T} R \bar{\mathbf{u}}_t^{*i}) \Delta t + V_{\mathbf{x},t}^{i T} \Sigma_t^i \Delta \mathbf{w}_t^i$

 Propagate system state: $\mathbf{x}_{t+1}^i = \mathbf{x}_t^i + \mathbf{f}_t^i \Delta t + \mathbf{G}_t^i \bar{\mathbf{u}}_t^{*i} \Delta t + \Sigma_t^i \Delta \mathbf{w}_t^i$

end for

 Compute targets: true terminal value and true value function gradient,

$$V_N^{*i} = \phi(\mathbf{x}_N^i); V_{\mathbf{x},N}^{*i} = \phi_{\mathbf{x}}(\mathbf{x}_N^i)$$

end for

 Compute mini-batch loss:

$$\mathcal{L} = \frac{1}{M} \sum_{i=1}^M a \|V_N^{*i} - V_N^i\|_2^2 + b \|V_{\mathbf{x},N}^{*i} - V_{\mathbf{x},N}^i\|_2^2 + c \|V_N^{*i}\|_2^2 + d \|V_{\mathbf{x},N}^{*i}\|_2^2 + \lambda \|\theta^k\|_2^2$$

 Gradient step:

$$\theta^{k+1}, \psi^{k+1} \leftarrow \text{Adam.step}(\mathcal{L}, \theta^k, \psi^k)$$

end for

return θ^K, ψ^K

Algorithm 2: Stochastic CBF layer (Safe OptNet layer)

Given:

h : Safe set characterizing barrier function; $\alpha(h(\mathbf{x})) = \gamma h(\mathbf{x})$;

$\mathbf{f}, \mathbf{G}, \Sigma$: System dynamics; L : Maximum QP iterations, R : control cost positive definite matrix

Input: Current system state (\mathbf{x}_t) and current predicted value function gradient ($V_{\mathbf{x},t}$)

Set up the QP problem:

$$Q = R, q = V_{\mathbf{x}}^T G, C = -\frac{\partial h}{\partial \mathbf{x}}^T \mathbf{G}(\mathbf{x}), d = \alpha(h(\mathbf{x})) + \frac{\partial h}{\partial \mathbf{x}}^T \mathbf{f}(\mathbf{x}) + \frac{1}{2} \text{tr} \left(\Sigma(\mathbf{x})^T \frac{\partial^2 h}{\partial \mathbf{x}^2} \Sigma(\mathbf{x}) \right)$$

Initialize solution:

initialize feasible \mathbf{u} , slack variable: $s \in \mathbb{R}_+^1$ and Lagrange multiplier for CBF inequality constraint:

$\lambda \in \mathbb{R}_+$ and Residue Tolerance: ϵ

for $j = 1$ to L **do**

$(\Delta \mathbf{u}_j^{aff}, \Delta s_j^{aff}, \Delta \lambda_j^{aff}) \leftarrow \text{Affine Scale } (\mathbf{u}_j, s_j, \lambda_j)$

$(\Delta \mathbf{u}_j^{cc}, \Delta s_j^{cc}, \Delta \lambda_j^{cc}) \leftarrow \text{Centering-Corrector } (\mathbf{u}_j, s_j, \lambda_j)$

$\zeta \leftarrow \text{Residual}(\mathbf{u}_j, s_j, \lambda_j)$

$\alpha \leftarrow \text{Step Size}(\mathbf{u}_j, s_j, \lambda_j)$

$\mathbf{u}_{(j+1)} = \mathbf{u}_j + \alpha(\Delta \mathbf{u}_j^{aff} + \Delta \mathbf{u}_j^{cc})$

$s_{(j+1)} = s_j + \alpha(\Delta s_j^{aff} + \Delta s_j^{cc})$

$\lambda_{(j+1)} = \lambda_j + \alpha(\Delta \lambda_j^{aff} + \Delta \lambda_j^{cc})$

if $\zeta \leq \epsilon$ **then**

 Break

end if

end for

Return $\bar{\mathbf{u}}^* \leftarrow \mathbf{u}_j$
