Deep $\mathcal{L}^1$ Stochastic Optimal Control Policies for Planetary Soft-landing

Marcus Aloysius Pereira [*]
The Institute for Robotics and Intelligent Machines, Georgia Institute of Technology, Atlanta, GA 30332

Camilo A. Duarte[†]
School of Aerospace Engineering, Georgia Institute of Technology, Atlanta, GA 30332

Ioannis Exarchos[‡]
Microsoft

Evangelos A. Theodorou[§]
School of Aerospace Engineering, Georgia Institute of Technology, Atlanta, GA 30332

***Abstract*: In this paper, a novel deep-learning-based solution is introduced to the Powered Descent Guidance (PDG) problem, grounded in principles of nonlinear Stochastic Optimal Control (SOC) and Feynman-Kac theory. Our algorithm solves the problem by framing it as an $\mathcal{L}^1$-SOC problem for minimum fuel consumption. Additionally, it can handle practically useful control constraints, nonlinear dynamics and enforces state constraints as soft-constraints. This is achieved by building off of recent work on Deep Forward-Backward Stochastic Differential Equations (Deep FBSDEs) and differentiable neural-network layers for non-convex optimization based on stochastic search. In contrast to previous approaches, our algorithm does not require convexification of the constraints or linearization of the dynamics and is empirically shown to be robust to stochastic disturbances and the initial conditions of the spacecraft. After training offline, our control policy can be activated once the spacecraft is within a pre-specified radius of the landing zone and at a pre-specified altitude, i.e., the base of an inverted cone with the tip at the landing zone. We demonstrate empirically that our controller can successfully and safely land all trajectories initialized in the vicinity of the base of this cone as well as with randomization in the starting velocities and spacecraft mass while minimizing fuel consumption.**

I. Introduction

The PDG problem addresses the final stage of the entry, descent, and landing sequence wherein a spacecraft uses its rocket engines to maneuver from some initial position to a safe landing at a desired landing location. It can be framed as

[*]**Corresponding Author** and a Ph.D. student in Robotics at Georgia Tech, email address: mpereira30@gatech.edu
[†]Master's student in the School of Aerospace Engineering at Georgia Tech, email address: candresdu@gmail.com
[‡]Work done during a postdoctoral fellowship at Stanford University, email address: exarchos@gatech.edu
[§]Associate Professor at the Daniel Guggenheim School of Aerospace Engineering, email address: evangelos.theodorou@gatech.edu

a finite time-horizon optimal control problem where the ultimate goal is to achieve a safe landing while minimizing the amount of fuel consumed during descent. The definition of a safe landing is provided in terms of state constraints (i.e., the terminal velocity and position). As a consequence, PDG is regarded as a control- and state-constrained optimization problem, with state constraints imposed by stringent mission requirements and control constraints imposed by the thrusting capabilities of the spacecraft.

The original solution to the PDG problem dates back to the 1960s, during the Apollo era and consisted of representing a reference trajectory as a vector of polynomial functions of time. Initially, Cherry [1] solved the Apollo Powered Descent Guidance (APDG) problem explicitly, wherein a vector of polynomial functions that evolve forward in time and intersect the current and target states, is repetitively solved on-board as the mission progresses. Later, Klumpp [2] derived a generalized implicit guidance equation, wherein the reference trajectory defined by the vector of polynomial functions satisfies a constrained Two-Point Boundary Value Problem (TPBVP). In this case, the solution is a state feedback policy where the feedback gains can be computed offline by simulating the mission. These two methodologies laid the theoretical foundations for several planetary landing studies thereafter. However, there are aspects of these guidance approaches that make them unfit for modern, more sophisticated missions. For instance, these approaches are not fuel-optimal since in both cases, a quadratic performance index on the thrust magnitude is minimized. The fallacy of the assumption that *quadratic costs minimize fuel-consumption* is proved in [3] wherein the author demonstrates how the choice of the norm of the thrust in the cost function is dependent on the type of rocket and which norms actually measure fuel consumption. It is shown that the well-known quadratic cost (or $\mathcal{L}^2$-norm) does not measure (and therefore does not minimize) fuel consumption and that an optimal control policy for quadratic costs will be sub-optimal with respect to other control costs that do measure fuel-consumption. For future crewed missions to planets with greater mass and stronger gravitational forces, the spacecraft mass will be significantly larger than that of a robotic mission. In such cases, propellant optimality may determine the feasibility of the mission. Another important limitation is that the Apollo-era methods do not consider hard constraints imposed on the thrust magnitude and direction, and require the time-to-go to the target condition to be pre-specified. In fact, an inaccurate choice for the time-to-go may result in solutions outside of the physical capabilities of the propulsion system, thereby rendering any pre-computed solutions sub-optimal and infeasible. The use of quadratic costs for fuel minimization can yield sub-optimal conditions for certain types of missions. For instance, as mentioned in [3], quadratic costs will provide continuous thrusting control policies which can cause undesirable effects such as perturbing the microgravity conditions or actively injecting noise on position pointing payloads. For such payloads, *bang-off-bang* controllers are preferable so that scientific experiments can be conducted during the *off* periods. Thus, the $\mathcal{L}^1$-norm is the *de facto* choice for designing optimal controllers for chemical space propulsion systems.

In [4], the author provides a framework to solve different versions of the fuel-optimal PDG problem which all use the $\mathcal{L}^1$-norm of the thrust as the control-cost over the entire time horizon. The approach is called an indirect method which relies on Pontryagin Maximum/Minimum Principle (PMP), and proves that a fuel-optimal solution is characterized by a *bang-bang* control profile with at most two optimal switching times for the 3D PDG problem. The method leads to a multivariate root finding problem whose solution provides the initial values of the co-state variables, the optimal switching times and time-to-go to the target condition. The aforementioned method, dubbed Universal Powered Guidance (UPG), boasts of its simplicity and considers hard constraints on the thrust explicitly in its formulation. However, UPG's solution relies on the assumption that the vehicle dynamics are deterministic and requires the root finding problem to be solved onboard or *on-the-fly* to handle stochasticity. This can be problematic since the proposed algorithm does not enjoy theoretically guaranteed convergence. An important contribution of UPG is the Powered Descent Initiation (PDI) condition to determine *when to transition* from the engine idle to engine burn phase to prevent large divert requirements or problem infeasibility. Recent work, dubbed the Augmented Apollo Powered Descent Guidance (A^2PDG) [5] combines this PDI condition with a tunable version of the APDG. By means of a tunable gain parameter, one can recover the original APDG law while other values allow trading-off between trajectory shaping and fuel consumption. However, the resulting guidance laws are not necessarily fuel-optimal because no such cost is optimized for and control-constraints on the thrust vector are ignored. Nevertheless, it equips the nearly 50-year old APDG with a capability for fuel-saving and ability to determine a PDI condition.

Another recent indirect approach [6], leverages the function approximation capabilities of DNNs (Deep Neural Networks) to learn so-called critical mission parameters which are the same parameters that UPG solves for using nonlinear root-finding. The inputs to the DNN are the initial position and velocity vectors sampled uniformly from pre-specified ranges and the targets for the DNN are obtained from the solution of an Nonlinear Program (NLP) solver. The approach is promising for on-board application since DNN inference is very fast and the predicted critical parameters can be fed into the PMP problem to quickly obtain the optimal control by forward propagating Ordinary Differential Equations (ODEs). However, the main drawbacks of the approach are - (i.) it does not account for stochasticity and (ii.) potentially biased input dataset. The later is due to the choice of the authors to reject those solutions of the NLP solver that do not appear *bang-bang-like* and is clear from their training description [6, Section III.B.3] which rejects about 53% of the sampled cases. In [7], the authors employ an imitation learning-like procedure wherein PMP is used to solve optimal control problems for soft-landing and generate training data. This data is then used to train DNNs that directly predict the optimal control via supervised learning. However, the approach is demonstrated only on 2D problems and does not consider any state constraints.

In contrast to indirect methods, direct methods discretize time converting the original PDG problem into a nonlinear optimization problem. Attempting to solve such a problem using brute force approaches can be done using NLP solvers; however, convergence is not guaranteed and they can be computationally inefficient making them unfit for onboard applications. As a result, existing direct approaches seek to provide a solution by performing a lossless convexification of the nonlinear problem [8] so that convergence is guaranteed. Another noteworthy work based on the direct approach presents a solution to the 6-DoF PDG problem with free final time, considers aerodynamics effects and introduces a continuous version of state-triggered constraints [9]. The proposed non-convex optimization algorithm relies on iterative convexification, solving local Second Order Cone Programming problems (SOCPs) and repeating the process until convergence. Despite the impressive results of this method demonstrated on real flight hardware, the approach has the following drawbacks - (i.) it regards the PDG problem as a feedforward trajectory generation problem and does not address the topic of feedback control, (ii.) has no convergence guarantee, (iii.) requires extra machinery such as virtual-control and trust region to handle artificial infeasibility and artificial unboundedness that are induced due to the convexification process, and (iv.) relies on iteratively solving the SOCP problem onboard to handle uncertainties arising from aerodynamic forces. Finally, recent efforts to develop real-time onboard trajectory optimization with full consideration of the aerodynamic forces [10] model angle-of-attack as an extra control input that influences the aerodynamic forces. Although, the obtained solutions are more robust to modeling errors, the addition of aerodynamic forces results in a more complicated lossless convexification analysis which is only applicable to the 2D PDG problem.

Incidentally, the approaches mentioned thus far, assume deterministic dynamics and ignore uncertainty that could arise due to modeling errors or atmospheric disturbances. In order to handle uncertainty, they have to resort to recomputing optimal solutions *on-the-fly* or employing a disturbance rejecting tracking controller. The former is computationally expensive and slow to perform on-board, while for the latter to work, conservative throttle-margins must be built into the pre-computed open-loop commands [11]. This is because a pre-computed fuel-optimal PDG throttle command trajectory will always have *bang-bang* profile [4] leading to saturation at the maximum and minimum possible values of throttle commands. Without built-in throttle margins, it would be impossible to employ additional control for disturbance rejection. The problem with overly conservative throttle-margins is that it leads to high fuel-cost (due to not applying maximum or minimum thrust commands) and therefore a pre-computed fuel-optimal control sequence will cease to be fuel-optimal or will be sub-optimal when combined with a feedback controller. To bridge this gap, minimum-fuel PDG controllers that explicitly account for stochasticity have been proposed [11, 12] which consider dynamics modelled by Stochastic Differential Equations (SDEs). To minimize the impact on fuel-cost caused by throttle margins, an approach based on covariance steering [11] solves for feedback gains using a linear model of the residual dynamics around a pre-computed mean trajectory and a user-defined terminal state covariance constraint. These gains are then used to optimize for the smallest possible throttle margin while ensuring that throttle constraints are respected with a user-defined minimum probability. Although minimizing fuel-cost as compared to deterministic approaches in a principled manner by explicitly accounting for stochasticity, this approach relies on two main assumptions - (i.) the open-loop (mean) control term is much larger than the feedback control term and (ii.) the mass-rate dynamics are deterministic. Both assumptions are required to eliminate mass from the residual state and render the corresponding dynamics linear thereby allowing one to use covariance steering to compute feedback gains. In the cases where (i.) is violated, the dynamics are no longer control-affine (i.e., linear w.r.t control) and covariance steering cannot be employed.

One scenario where this can arise is high noise-variance in the SDE (to account for high uncertainty), resulting in higher feedback gains. This in turn also leads to increasing the throttle-margin and thereby increasing the fuel-cost [11, Section III.D]. In contrast to (ii.), a realistic model of stochasticity should consider the noise in the mass-rate dynamics to be inversely correlated to that affecting the acceleration dynamics [12, Section II.B]. The approach based on Forward-Backward Stochastic Differential Equations (FBSDEs) [12] does not rely on such assumptions but has only been applied to the one-dimensional PDG problem. The closed-form optimal control expression presented for the 1D problem does not hold for the general 3D constrained PDG problem as well as the proposed numerical algorithm is prone to compounding errors from least-squares approximations at every time step. Nevertheless, the results demonstrate superior performance in terms of crash percentages when compared to deterministic controllers and show a comparable fuel consumption to the venerable APDG law.

To the best of our knowledge, our work is the first to propose a deep learning based solution to the stochastic 3D constrained PDG problem. Our work is inspired by [12] and builds off of recent work [13] that uses DNNs to solve systems of FBSDEs subject to stochastic dynamics with non-affine controls and hard non-convex control constraints. These so called deep FBSDE architectures are scalable solutions to high-dimensional parabolic Partial Differential Equations (PDEs) such as the Hamilton-Jacobi-Bellman PDE (HJB-PDE) that one encounters in continuous-time SOC problems. These do not suffer from compounding least-squares errors and do not require backpropagating SDEs. By treating the initial-value of the BSDE (Backward SDE) as a learnable parameter of the DNN, the BSDE can be forward propagated and the deviation from the given terminal-value can be used as a loss function to train the DNN. This approach has been used to successfully solve high-dimensional problems in finance [13] and safety-critical control problems [14]. Compared to all the aforementioned work thus far, our main contributions are as follows:

1) An HJB-PDE based indirect approach to the stochastic 3D constrained PDG problem, in contrast to PMP-based methods in literature for deterministic versions of the PDG problem.
2) Ability to solve the nonlinear $\mathcal{L}^1-$SOC PDG problem using deep FBSDEs without relying on customized convexification analysis and convex solvers in an end-to-end differentiable manner. This allows for future extensions on incorporating more complicated dynamics models without any change to the theory.
3) Incorporated *first-exit* time capability into the deep FBSDE framework for the PDG problem to handle uncertainty in the total time of flight.
4) A framework that can be trained to be invariant of the initial position of the spacecraft and handle stochastic disturbances. The trained network can be deployed as a feedback policy without having to recompute the optimal solution *on-the-fly*.

With regards to computational burden, similar to [7], our approach is also based on training a policy network offline. The online computation comprises of a forward pass through a neural network and one-step parallel simulation of the dynamics. These computations can be performed entirely on a CPU (using vectorized operations) or a modest GPU.

II. Problem Formulation

A. Spacecraft Dynamics and Constraints

For this study, we consider the dynamics of the final stage of the PDG problem only, as a result we make the following assumptions: (1) aerodynamic forces are neglected such that only gravity and thrust forces act on the vehicle, (2) the spacecraft is at a relatively low altitude such that a flat planet model can be assumed, and at a reasonable distance to the desired landing zone; (3) similar to [8] we assume high bandwidth attitude control so that we can decouple translational and rotational dynamics of the vehicle and (4) we consider the initial velocity to be in the subsonic regime. Due to the assumption (3), we completely neglect rotational dynamics of the spacecraft in this formulation and assume that the attitude of the vehicle needed to produce the required thrust profile can be achieved instantaneously. Therefore, it is sufficient to define the dynamics of the vehicle by its three-dimensional translational dynamics which are as follows:

$$\begin{aligned}\dot{\mathbf{r}}(t) &= \mathbf{v}(t),\\ \dot{\mathbf{v}}(t) &= \frac{\mathbf{T}(t)}{m(t)} - \mathbf{g}\\ \dot{m}(t) &= -\frac{1}{c}||\mathbf{T}(t)||\end{aligned} \tag{1}$$

where, at any time t, $\mathbf{r}(t) \in \mathbb{R}^3$ is the position of the spacecraft with respect to a defined inertial frame, $\mathbf{v}(t) \in \mathbb{R}^3$ is the velocity defined in the same frame and $m(t) \in \mathbb{R}^+$ is the spacecraft's total mass. $\mathbf{T} \in \mathbb{R}^3$ is the thrust vector generated by the propulsion system, $\mathbf{g} \in \mathbb{R}^3$ is the acceleration vector due to the gravitational force exerted by the planet (we consider Mars for our simulations) on the spacecraft, and $c \in \mathbb{R}^+$ is the effective exhaust speed of the propulsion system which governs the rate at which fuel is consumed proportional to the generated thrust.

In a stochastic setting, as described in [12], we assume that stochastic disturbances enter the acceleration channels due to unmodeled environmental disturbances and error induced by limitations of the thrust modulation mechanism. Moreover, these disturbances are negatively correlated with the noise that enters the mass-rate channel. The SDEs representing the dynamics of the vehicle as a function of time are as follows,

$$\begin{aligned}\mathrm{d}\mathbf{r}(t) &= \mathbf{v}(t)\mathrm{d}t,\\ \mathrm{d}\mathbf{v}(t) &= \left[\frac{\mathbf{T}(t)}{m(t)} - \mathbf{g}\right]\mathrm{d}t + \frac{\Gamma}{m(t)}\mathrm{d}W(t),\\ \mathrm{d}m(t) &= -\frac{1}{c}\left[||\mathbf{T}(t)||\mathrm{d}t + \mathbf{1}_{1\times 3}^{\mathrm{T}}\,\Gamma\,\mathrm{d}W(t)\right]\end{aligned} \tag{2}$$

where, $\mathrm{d}W \in \mathbb{R}^3$ is a vector of mutually independent Brownian motions and $\Gamma \in \mathbb{R}^{3\times 3}$ is the covariance matrix which in our case is a diagonal matrix containing the uncorrelated noise variances entering the three acceleration channels. A vector of ones ($\mathbf{1}_{1\times 3}$) is used to combine the Brownian motions entering the acceleration channels to obtain a Brownian motion that enters the mass-rate channel which is negatively correlated with those that enter the acceleration channels (due to the $-1/c$ coefficient). We can rewrite the dynamics concisely in state-space form as follows:

$$\mathrm{d}\mathbf{x}(t) = f(\mathbf{x}(t), \mathbf{T}(t))\,\mathrm{d}t + \Sigma(\mathbf{x}(t))\,\mathrm{d}W(t), \tag{3}$$

where, $\mathbf{x}(t) \in \mathbb{R}^7$ is the state vector, $f(\mathbf{x}(t), \mathbf{T}(t))$ is the *drift* vector representing the deterministic component and $\Sigma(\mathbf{x}(t)) \triangleq H(\mathbf{x}(t))\Gamma$ is the *diffusion* matrix representing the stochastic component of the dynamics. The components of the state vector are, $\mathbf{x} = [\mathbf{r}(t)^{\mathrm{T}}, \mathbf{v}(t)^{\mathrm{T}}, m(t)]^{\mathrm{T}}$ and $H(\mathbf{x})$ is a 7×3 matrix defined as follows,

$$H(\mathbf{x}(t)) = \begin{bmatrix} \mathbf{0}_{3\times3} & \frac{1}{m(t)}\mathbf{I}_{3\times3} & -\frac{1}{c}\mathbf{1}_{3\times1} \end{bmatrix}^T$$

Similar to other works in PDG literature, we consider hard constraints on the thrust vector $\mathbf{T}(t)$ that are imposed by the physical limitations of the spacecraft's propulsion system. In order for the propulsion system to operate reliably, the engines may not operate below a certain thrust level. In addition, the thrusters are only capable of producing finite thrust. We represent these constraints by the following inequality constraint,

$$0 < \rho_1 \leq ||\mathbf{T}(t)|| \leq \rho_2 \tag{4}$$

The presence of a lower bound in the constraint above yields a non-convex set of feasible thrust values which could lead to a non-convex optimization problem. The conventional approach [8] is to convexify the problem to handle the non-convex constraints and show that the convexification is lossless. In this paper, we show that our method is able to work directly with the non-convex thrust constraints.

Additionally, a constraint on the direction in which thrust can be applied is also imposed. The so-called *thrust-pointing* constraint is given by,

$$\hat{\mathbf{n}} \cdot \mathbf{T}(t) \geq ||\mathbf{T}(t)|| \cos\theta \tag{5}$$

where, $\hat{\mathbf{n}} \in \mathbb{R}^3$ is a unit vector describing a desired pointing direction, and $\theta \in [0, \pi]$ is a fixed pre-specified maximum angle between the thrust vector $\mathbf{T}(t)$ and $\hat{\mathbf{n}}$. Intuitively, this constraint is required for sensors such as cameras to ensure that the ground is always in the field-of-view. For values of $\theta > \pi/2$ radians, this also leads to a non-convex set of feasible thrust values. Now, although our proposed method can handle such a constraint, to ensure the practical usefulness of maintaining the ground in the field-of-view, we limit θ to be strictly less than $\pi/2$ radians.

Next, we introduce state constraints that are necessary to ensure a soft-landing at a pre-specified landing zone. Our strategy is to handle these as soft-constraints and heavily penalize violations. These soft-constraints are represented by adding extra terms to the terminal and running cost functions of our proposed stochastic optimal control problem formulation. Incidentally, the goal of our proposed algorithm is to minimize the expected running and terminal costs, where the expectation is evaluated using sampled trajectories according to (2). Similar to [12], because the approach discussed in this paper requires trajectory sampling, it is imperative to impose an upper bound on the duration of each trajectory. This is necessary because it is possible to encounter trajectories with very large or infinite duration that cannot be simulated. On the other hand, it is practically meaningless to continue the simulation if a landing or crash occurs prior to reaching this upper bound. Thus, we formulate a *first-exit* problem with a finite upper bound on flight time where the simulation is terminated when one of the following two conditions is met: 1) we reach the ground, i.e., $r_3 = 0$ (or more realistically some threshold $r_3 \leq h_{\text{tol}}$ where h_{tol} is some arbitrarily small number defining a height at which shutting off the thrusters would be considered safe), or 2) the time elapsed during simulation is equal to or greater than a predetermined maximum simulation time (t_f seconds), whichever occurs first. Mathematically, the *first-exit* time, $\mathcal{T}$, is defined as follows,

$$\begin{aligned} \text{Let, } \tau &= \inf_s \left\{ s \in [0,\, t_f] \,\middle|\, r_3(s) \leq h_{\text{tol}} \right\} \\ \mathcal{T} &= \min(\tau,\, t_f). \end{aligned} \tag{6}$$

The vehicle is required to perform a safe landing which is characterized by a zero terminal velocity at a predetermined landing zone. However, in a stochastic setting, the probability of a continuous random variable being exactly equal to a specific value is zero. Thus, under stochastic disturbances, it is unrealistic to impose exact terminal conditions. Our strategy is to penalize the mean-squared deviations from the desired positions and velocities at $t = \mathcal{T}$ seconds and thus approach the target positions and velocities on average. As it will be shown later, our simulations demonstrate controlled trajectories that terminate in the vicinity of the desired terminal conditions. We define the following components of our proposed terminal cost function,

1) $\phi_x = \big(r_1(\mathcal{T})\big)^2$ and $\phi_y = \big(r_2(\mathcal{T})\big)^2$, which means, without loss of generality, that we consider the x and y coordinates of the landing zone to be at the origin of the interial frame.
2) $\phi_z = \big(r_3(\mathcal{T})\big)^2$, which means that we penalize the residual altitude at $t = \mathcal{T}$ seconds to discourage hovering.
3) $\phi_{v_x} = \big(\dot{r}_1(\mathcal{T})\big)^2$ and $\phi_{v_y} = \big(\dot{r}_2(\mathcal{T})\big)^2$, which means that we penalize the residual x and y velocities at $t = \mathcal{T}$ seconds to discourage landing with unsafe lateral terminal velocities.
4) $\phi_{v_z} = \begin{cases} c_{v_z+}\big(\dot{r}_3(\mathcal{T})\big)^2, & \dot{r}_3(\mathcal{T}) > 0 \text{ m/s} \\ c_{v_z-}\big(\dot{r}_3(\mathcal{T})\big)^2, & \dot{r}_3(\mathcal{T}) \leq 0 \text{ m/s} \end{cases}$
   where the constants c_{v_z+} and c_{v_z-} represent penalty terms for the residual vertical velocity. Note that the terminal vertical velocity is penalized such that $c_{v_z+} >> c_{v_z-}$ in order to discourage hovering around the landing zone.

An inequality constraint on the spacecraft's total mass given by, $m(\mathcal{T}) \geq m_d$, is used to ensure that the dry mass (m_d

kgs) of the vehicle is lower than the total mass at terminal time $\big(m(\mathcal{T})\big)$. We enforce this constraint as follows:

$$\phi_m = \exp\left(-\frac{m(\mathcal{T}) - m_d}{m(0) - m_d} \right)$$

wherein, the penalty increases exponentially if the terminal mass $(m(\mathcal{T}))$ falls below the dry mass m_d. Additionally, this also encourages minimum fuel consumption as higher values of $\big(m(\mathcal{T}) - m_d\big)$ lead to lower values of ϕ_m.

The terminal cost function can now be stated as a weighted sum of the terms described above,

$$\phi(\mathbf{x}(\mathcal{T})) = Q_x \cdot \phi_x + Q_y \cdot \phi_y + Q_z \cdot \phi_z + Q_{v_x} \cdot \phi_{v_x} + Q_{v_y} \cdot \phi_{v_y} + Q_{v_z} \cdot \phi_{v_z} + Q_m \cdot \phi_m \tag{7}$$

where, the coefficients (Q_i) allow the user to assign relative importance to each term in the terminal cost function.

A glide-slope constraint is also employed to keep the vehicle in an inverted cone with the tip of the cone at the landing zone [8]. This is given by,

$$\tan\gamma \cdot \Big\| \big(r_1(t),\, r_2(t)\big) \Big\| \leq r_3(t), \tag{8}$$

where $\gamma \in [0,\, \pi/2)$ is the minimum admissible glideslope angle. We can convert this inequality constraint to an equality constraint as follows,

$$\Delta_{\text{glide}} = \tan\gamma \cdot \sqrt{r_1(t)^2 + r_2(t)^2} - r_3(t) \tag{9}$$

Since this constraint is imposed at every point in time, we define it as the running cost function, thus,

$$l(t,\, \mathbf{x}(t)) = \begin{cases} q_+ \cdot \Delta_{\text{glide}}^2, & \Delta_{\text{glide}} > 0 \\ q_- \cdot \Delta_{\text{glide}}^2, & \Delta_{\text{glide}} \leq 0 \end{cases} \tag{10}$$

where, $q_+ >> q_-$ to heavily penalize trajectories from leaving the glide-slope cone. Note that we do not set q_- to zero as this encourages hovering around the landing zone at high altitudes by making Δ_{glide} highly negative. Thus, a non-zero value for q_- encourages landing.

Finally, concerning the initial conditions, we assume that separate navigation systems onboard the spacecraft take care of the main flight segment (e.g., from planet to planet) and will navigate the spacecraft to a position that is within a reasonable distance from the landing zone for the final descent stage to begin. Specifically, we assume that the final descent stage is initialized when the spacecraft reaches a certain altitude. We sample from a normal distribution around a pre-specified mean trigger altitude, to ensure that the control policy can handle small perturbations around the trigger value. As far as the corresponding initial x,y coordinates are concerned, we assume that these lie on the base of an inverted cone as defined by (8). Sampling on the base of the cone allows our control policy to be able to handle the spacecraft approaching from any direction towards a pre-specified landing zone. The initial vertical velocity (v_z) and initial mass are also sampled from normal distributions to ensure tolerance to perturbations. Finally, the horizontal velocities (v_x and v_y) are initialized such that they are directed towards the axis of the inverted cone with the directions being perturbed and the magnitude set proportional to the distance from the axis. Specifically, we assume that the main navigation system is aware of the landing zone and will adjust the speed of the spacecraft proportional to its horizontal distance from the landing zone. Therefore, if the spacecraft starts (the descent stage) on the rim of the base of the inverted cone, it will have a higher horizontal velocity as compared to if it starts very close to the axis of the cone.

B. The Minimum Fuel or $\mathcal{L}^1$ Stochastic Optimal Control Problem

We can now formulate the three-dimensional PDG stochastic optimal control problem as a constrained non-convex minimization problem where the goal is to minimize the amount of fuel needed to achieve a safe landing. As motivated in the introduction and in [3], we consider the $\mathcal{L}^1$-norm of the thrust as the running control cost (as opposed to the conventional quadratic cost or $\mathcal{L}^2$-norm) to correctly measure and hence minimize the total fuel consumption. The optimization problem is formally stated as,

minimize: $$\mathcal{J} = \mathbb{E}\left[\phi\big(x(\mathcal{T})\big) + \int_0^{\mathcal{T}} \Big(l\big(s,\, \mathbf{x}(s)\big) + q_{\mathcal{L}^1}||\mathbf{T}(t)||\Big)\mathrm{d}s\right]$$

subject to:

$$\begin{aligned}
&\mathrm{d}\mathbf{r}(t) = \mathrm{d}\mathbf{v}(t)\mathrm{d}t,\\
&\mathrm{d}\mathbf{v}(t) = \frac{\mathbf{T}(t)}{m(t)}\mathrm{d}t - \mathbf{g}\mathrm{d}t + \frac{\Gamma}{m(t)}\mathrm{d}W(t),\\
&\mathrm{d}m(t) = -\frac{1}{c}\left[||\mathbf{T}(t)||\mathrm{d}t + \mathbf{1}_{1\times3}^{\mathrm{T}}\Gamma\mathrm{d}W(t)\right],\\
&0 < \rho_1 \le ||\mathbf{T}(t)|| \le \rho_2, \quad \hat{\mathbf{n}}\cdot\mathbf{T}(t) \ge ||\mathbf{T}(t)||\cos\theta
\end{aligned} \tag{11}$$

where, $\phi : \mathbb{R}^n \to \mathbb{R}^+$ is defined in (7), $l : \mathbb{R}^n \to \mathbb{R}^+$ is defined in (10), and $q_{\mathcal{L}^1}$ is a positive scalar weight assigned to the $\mathcal{L}^1$-norm of the thrust vector. With the presented constraints, we now have three sources of non-convexity in the problem formulation, 1) the relationship between the mass-rate $\big(\dot{m}(t)\big)$ and the thrust vector $\big(\mathbf{T}(t)\big)$ in the dynamics, 2) the lower bound on the norm of the thrust vector $\big(\rho_1 \le ||\mathbf{T}(t)||\big)$, and, 3) the thrust-pointing constraint when $\theta > \pi/2$. Existing work in literature either attempts to convexify the original problem so that customized convex solvers can be used, or relies on sequential convex programming to iteratively convexify and solve the original nonlinear problem. In contrast to these methods, we propose and develop an approach, in the subsequent sections of the paper, that can handle the nonlinear dynamics and does not require complex reformulations of the problem at hand.

C. Solution using Forward-Backward Stochastic Differential Equations

In this section, we describe our methodology to solve the $\mathcal{L}^1$ stochastic optimal control problem described in (11). We seek to minimize the expected cost with respect to the set of all admissible controls $\mathcal{U}$. We begin by defining the value function (V) (i.e., the *minimum cost-to-go*) as follows,

$$\begin{cases} V\big(\mathbf{x}(t),\, t\big) = \inf_{\mathbf{T}(\cdot)\in\mathcal{U}[0,\mathcal{T}]} \mathcal{J} \\ V\big(\mathbf{x}(\mathcal{T}),\, \mathcal{T}\big) = \phi\big(\mathbf{x}(\mathcal{T}),\, \mathcal{T}\big) \end{cases} \tag{12}$$

Using Bellman's principle of optimality and applying Ito's lemma [15], one can derive the HJB-PDE given by,

$$\begin{cases} V_t + \inf_{\mathbf{T}(\cdot)\in\mathcal{U}[0,\mathcal{T}]} \left\{ \frac{1}{2}\mathrm{tr}\big(V_{\mathbf{xx}}\Sigma\Sigma^{\mathrm{T}}\big) + V_{\mathbf{x}}^{\mathrm{T}} f\big(\mathbf{x}(t),\, \mathbf{T}(t),\, t\big) + l\big(\mathbf{x}(t),\, t\big) + q_{\mathcal{L}^1}\big\|\mathbf{T}(t)\big\| \right\} = 0 \\ V(\mathbf{x}(\mathcal{T}), \mathcal{T}) = \phi(\mathbf{x}(\mathcal{T}), \mathcal{T}) \end{cases} \tag{13}$$

where the subscripts t and $\mathbf{x}$ are used to denote partial derivatives with respect to time and the state vector, respectively. The term inside the infimum operator is known as the Hamiltonian (denoted $\mathcal{H}$). The HJB-PDE is a backward, nonlinear parabolic PDE and solving it using grid-based methods is known to suffer from the well-known *curse-of-dimensionality*. Among some of the recent scalable methods to solve nonlinear parabolic PDEs, the Deep FBSDEs [14, 16, 17] based solution is the most promising and has been used successfully for high-dimensional problems in finance [13]. Deep FBSDEs leverage the function approximation capabilities of deep neural networks to solve systems of FBSDEs which in turn solve the corresponding nonlinear parabolic PDE. The connection between the solutions of nonlinear parabolic PDEs and FBSDEs is established via the nonlinear Feynman-Kac lemma [18, Lemma 2]. Thus, applying the nonlinear Feynman-Kac lemma to (13) yields the following system of FBSDEs,

$$\mathbf{x}(t) = \mathbf{x}(0) + \int_0^t f\big(\mathbf{x}(t),\, \mathbf{T}^*(t),\, t\big)\,\mathrm{d}t + \int_0^t \Sigma\big(\mathbf{x}(t),\, t\big)\,\mathrm{d}W(t) \quad \text{[FSDE]} \tag{14}$$

$$V\big(\mathbf{x}(t),\, t\big) = \phi\big(\mathbf{x}(\mathcal{T})\big) + \int_t^{\mathcal{T}} \Big(l\big(\mathbf{x}(t),\, t\big) + q_{\mathcal{L}^1}\big\|\mathbf{T}^*(t)\big\|\Big)\mathrm{d}t - \int_t^{\mathcal{T}} V_{\mathbf{x}}^{\mathrm{T}}\Sigma\big(\mathbf{x}(t),\, t\big)\,\mathrm{d}W(t) \quad \text{[BSDE]} \tag{15}$$

$$\mathbf{T}^*(t) = \underset{\mathbf{T}\in\mathcal{U}}{\mathrm{argmin}}\ \mathcal{H}\big(\mathbf{x}(t),\, \mathbf{T}(t),\, V_{\mathbf{x}},\, V_{\mathbf{xx}}\Sigma\Sigma^{\mathrm{T}}\big) \quad \text{[Hamiltonian minimization]} \tag{16}$$

*(Note: from hereon, the superscript * will be used to denote a solution to the respective optimization problem)*

Because of the terminal condition $V\big(\mathbf{x}(\mathcal{T}),\, \mathcal{T}\big) = \phi\big(\mathbf{x}(\mathcal{T})\big)$, the value-function $V\big(\mathbf{x}(t),\, t\big)$ evolves backward in time while $\mathbf{x}(t)$ evolves forward in time. As a result, the above system of FBSDEs yields a TPBVP. Although simulating $\mathbf{x}(t)$ might be trivial, $V\big(\mathbf{x}(t),\, t\big)$ cannot be naively simulated by backward integration like an ODE. This is because within the Ito-calculus framework, in order for solutions to be adapted, the process should be non-anticipating; which means that in this case, naive backward integration of $V\big(\mathbf{x}(t),\, t\big)$ would result in it depending explicitly on future values of noise making it an anticipating stochastic process. One solution to solve BSDEs is to backward-propagate the conditional expectation of the process as it is done in [18]. However, the least-squares-based algorithm to approximate the conditional expectation suffers from compounding approximation errors at every time step and does not scale for long time horizons. To overcome this, the deep FBSDE method [16] parameterizes the unknown value function $V\big(\mathbf{x}(0),\,0;\,\xi\big)$ at the initial time step using trainable weights of a neural network and parameterizes the gradient of the value-function $V_{\mathbf{x}}\big(\mathbf{x}(t),\,t;\,\xi\big)$ using an LSTM-based deep neural network. The parameters ξ of the network are trained using Adam [19] or any variant of the stochastic gradient descent algorithm. By introducing an initial condition, the BSDE is forward propagated as if it were a forward SDE and the known terminal condition $\Big(V\big(\mathbf{x}(\mathcal{T}),\,\mathcal{T}\big) = \phi\big(\mathbf{x}(\mathcal{T})\big)\Big)$ is used as a training loss for the deep neural network. This solution has been demonstrated to be immune to compounding errors and can scale to high-dimensional problems [14, 16, 17]. The Hamiltonian minimization at every time step computes the optimal control (i.e., the optimal thrust) that is used in the drifts of the FSDE and the BSDE. For numerical simulations, the system of FBSDEs is discretized in time using an Euler-Maruyama discretization [20] to yield the following set of equations,

$$\mathbf{x}[k+1] = \mathbf{x}[k] + f\big(\mathbf{x}[k],\,\mathbf{T}^*[k],\,k\big)\,\Delta t + \Sigma\big(\mathbf{x}[k],\,k\big)\,\Delta W[k] \tag{17}$$

$$V\big(\mathbf{x}[k+1],\,k+1\big) = V\big(\mathbf{x}[k],\,k\big) - \Big(l\big(\mathbf{x}[k],\,k\big) + q_{\mathcal{L}^1}\big\|\mathbf{T}^*[k]\big\|\Big)\Delta t + V_{\mathbf{x}}^{\mathrm{T}}\Sigma\big(\mathbf{x}[k],\,k\big)\,\Delta W[k] \tag{18}$$

$$\mathbf{T}^*[k] = \operatorname*{argmin}_{\mathbf{T}\in\mathcal{U}} \mathcal{H}\big(\mathbf{x}[k],\,\mathbf{T}[k],\,V_{\mathbf{x}},\,V_{\mathbf{xx}}\Sigma\Sigma^{\mathrm{T}}\big) \tag{19}$$

where k denotes the discrete-time index and Δt denotes the time-interval (in continuous-time) between any two discrete-time indices k and $k+1$.

For systems with control-affine dynamics and quadratic running control costs (i.e., the $\mathcal{L}^2$ norm of the control) as in [16], the minimization step (19) has a closed form expression for the optimal control (i.e., the optimal thrust profile), $\mathbf{T}^*(t)$. For the one dimensional soft-landing problem as in [12], although the $\mathcal{L}^1$ norm is used, a closed form expression (i.e., the *bang-bang* controller) can be derived because the dynamics are affine w.r.t the control. However, for the general soft-landing problem in three dimensions, as presented in this paper, the dynamics are non-affine with respect to the controls. As a result, a closed-form *bang-bang* optimal control cannot be derived and the Hamiltonian minimization step must be solved numerically. Additionally, as described in equation (11), the general problem has non-trivial control constraints with non-affine dynamics. In the following section, we build off of recent work [13] that embeds a non-convex optimizer into the deep FBSDE framework to solve non-convex Hamiltonian minimization problems at each time step. We extend this framework to handle the aforementioned control constraints as well as the first-exit problem formulation. Moreover, as stated in [13] this non-convex optimizer is differentiable and can facilitate end-to-end learning making it a good fit for the deep FBSDE framework.

III. Proposed Solution using NOVAS-FBSDE

The presence of the Euclidean-norm $||\cdot||$ in the equation for $\dot{m}(t)$ makes the dynamics a non-affine function of the thrust vector, $\mathbf{T}(t)$. Additionally, the control constraints given by equations (4) and (5) are non-convex as described in the previous sections. As a result, (19) is a non-convex optimization problem. The general Hamiltonian ($\mathcal{H}$) takes the following form,

(Note: henceforth the dependence of $V_{\mathbf{x}}$, $V_{\mathbf{xx}}$ and Σ on $\mathbf{x}$ and t will be dropped for ease of readability)

$$\mathcal{H}\big(\mathbf{x}(t),\, \mathbf{T}(t),\, V_{\mathbf{x}},\, V_{\mathbf{xx}}\Sigma\Sigma^{\mathrm{T}}\big) \triangleq \frac{1}{2}tr\big(V_{\mathbf{xx}}\Sigma\Sigma^{\mathrm{T}}\big) + V_{\mathbf{x}}^{\mathrm{T}} f\big(\mathbf{x}(t), \mathbf{T}(t)\big) + l\big(t, \mathbf{x}(t), \mathbf{T}(t)\big)$$

However, since in this problem the diffusion matrix Σ is not dependent on the control $\mathbf{T}(t)$ (i.e., we do not consider control-multiplicative noise), the trace-term can be ignored from the above expression and unlike [13] we do not require an extra neural network to predict the terms of the hessian of the value function $V_{\mathbf{xx}}$. Thus, the simplified Hamiltonian for our problem is given by,

$$\mathcal{H}\big(\mathbf{x}(t),\, \mathbf{T}(t),\, V_{\mathbf{x}}\big) = V_{\mathbf{x}}^{\mathrm{T}} f\big(\mathbf{x}(t), \mathbf{T}(t)\big) + q_{\mathcal{L}^1}\big\|\mathbf{T}(t)\big\| \tag{20}$$

Recently, a new framework [13] to handle non-convex Hamiltonian minimization problems using deep FBSDEs has been developed wherein the authors employ the Gradient-based Adaptive Stochastic Search (GASS) [21] algorithm to solve problems such as (19) while allowing efficient back-propagation of gradients to train the deep FBSDE network. This framework is called NOVAS-FBSDE wherein NOVAS stands for *Non-Convex Optimization Via Adaptive Stochastic Search*. NOVAS has been demonstrated to recover the closed-form optimal control profile in the case of control-affine dynamics and quadratic control costs. Additionally, it has been successfully tested in simulation on high-dimensional systems such as portfolio optimization with 100 stocks [13] in simulation. In a nutshell, at each time step, the Hamiltonian ($\mathcal{H}$) is minimized using the GASS algorithm. Briefly stated, Adaptive Stochastic Search first converts the original deterministic problem into a stochastic problem by introducing a parameterized distribution $\rho(\mathbf{T}(t);\, \theta)$ on the control $\mathbf{T}(t)$ and reformulating the minimization of $\mathcal{H}$ with respect to $\mathbf{T}(t)$ as the minimization of its expectation, $\mathbb{E}[\mathcal{H}]$, with respect to θ. This allows for $\mathcal{H}$ to be an arbitrary function of $\mathbf{T}(t)$ (potentially non-differentiable) and $\mathbb{E}[\mathcal{H}]$ is approximated by sampling from $\rho(\mathbf{T}(t);\, \theta)$. By minimizing $\mathbb{E}[\mathcal{H}]$, the upper bound on $\mathcal{H}$ is minimized. We invite the interested reader to refer to the appendix for a derivation of the equations of the NOVAS optimizer.

Notice that the general problem (11) has hard control constraints (i.e. equations (4) and (5)). To enforce these constraints, we employ a novel sampling scheme that allows us to efficiently sample feasible control inputs while guaranteeing that the control constraints are satisfied. To define the constrained sampling procedure, we make the following assumptions,

Assumption 1. *The horizontal thrust components* $\big(\mathbf{T}_1(t),\, \mathbf{T}_2(t)\big)$ *are bounded based on the lower bound of the norm of* *the thrust ρ_1, so that $|\mathbf{T}_1(t)| \leq \frac{\rho_1}{2}$ and $|\mathbf{T}_2(t)| \leq \frac{\rho_1}{2}$.*

Assumption 2. *The maximum angle θ between the thrust vector $\mathbf{T}(t)$ and $\hat{\mathbf{n}}$ belongs to the interval $\left[\frac{\pi}{6}, \frac{\pi}{2}\right)$.*

Assumption 3. *The bounds ρ_1, ρ_2 and the angle θ satisfy, $\sqrt{\frac{\rho_1^2}{2 \cdot \sin^2\theta}} \leq ||\mathbf{T}(t)|| \leq \rho_2$.*

The assumption 2 is justified because values of $\theta \geq \pi/2$ will result in the camera sensors loosing the ground from their field of view, while very low values of θ will restrict horizontal motion.

We choose $\hat{\mathbf{n}} = [0,\ 0,\ 1]^{\mathrm{T}}$ and therefore we can show that if $\hat{\mathbf{n}} \cdot \mathbf{T} = \mathbf{T}_3 \geq ||\mathbf{T}|| \cos\theta$, the *thrust-pointing* control constraint is always satisfied.

Lemma 1. *Given that assumptions 1– 3 hold, the thrust-pointing constraint $\mathbf{T}_3 \geq ||\mathbf{T}|| \cos\theta$ is satisfied.*

Proof. Given that, $\sqrt{\frac{\rho_1^2}{2 \cdot \sin^2\theta}} \leq ||\mathbf{T}|| \leq \rho_2$, we have $\frac{\rho_1^2}{2 \cdot \sin^2\theta} \leq ||\mathbf{T}||^2 \leq \rho_2^2$.

Therefore, $\frac{\rho_1^2}{2} \leq ||\mathbf{T}||^2 \sin^2\theta = ||\mathbf{T}||^2(1 - \cos^2\theta) = ||\mathbf{T}||^2 - ||\mathbf{T}||^2 \cos^2\theta$

Based on assumption 1, we know that, $\mathbf{T}_1^2 + \mathbf{T}_2^2 \leq \frac{\rho_1^2}{2}$.

Combining this with the inequality above i.e., $\frac{\rho_1^2}{2} \leq ||\mathbf{T}||^2 - ||\mathbf{T}||^2 \cos^2\theta$, we have,

$\mathbf{T}_1^2 + \mathbf{T}_2^2 \leq \frac{\rho_1^2}{2} \leq ||\mathbf{T}||^2 - ||\mathbf{T}||^2 \cos^2\theta$

Therefore, re-arranging the terms we have, $||\mathbf{T}||^2 \cos^2\theta \leq ||\mathbf{T}||^2 - \mathbf{T}_1^2 - \mathbf{T}_2^2 = \mathbf{T}_3^2$

which implies that, $||\mathbf{T}|| \cos\theta \leq \mathbf{T}_3$ ∎

Thus, for lemma 1 to hold, we need to satisfy assumptions 1– 3. Assumption 2 is satisfied by design decisions. For assumptions 1 and 3, we first sample the horizontal thrust components $\big(\mathbf{T}_1(t),\ \mathbf{T}_2(t)\big)$ and the norm of the thrust $||\mathbf{T}(t)|| = \sqrt{\mathbf{T}_1^2(t) + \mathbf{T}_2^2(t) + \mathbf{T}_3^2(t)}$ and then we project these samples onto closed intervals such that both assumptions along with the original thrust bounds in (4) are satisfied. Defining $\rho_3 = \sqrt{\frac{\rho_1^2}{2 \cdot \sin^2\theta}}$ and projecting the samples of $||\mathbf{T}(t)||$ onto the interval $\big[\max(\rho_1,\ \rho_3),\ \rho_2\big]$, both control constraints (equations (4) and (5)) can be satisfied. A pseudo-code of this sampling scheme is presented in Algorithm 1 wherein the inputs $\mu_{\mathbf{T}}$ and $\Sigma_{\mathbf{T}}$ are the sampling mean and variance used by the NOVAS algorithm. Note that the subscript $\mathbf{T}$ has been used to differentiate the distribution $\rho(\mathbf{T}; \theta)$ parameters for sampling thrust values from the diffusion matrix Σ. The algorithm outputs feasible values of horizontal thrust and thrust norm (i.e., $[\mathbf{T}_1,\ \mathbf{T}_2,\ ||\mathbf{T}||]$) from which the feasible vertical thrust can be computed.

IV. Algorithmic Details

In this section, we present algorithmic details concerning the capability to handle random initial positions and training of the NOVAS-FBSDE network with *first-exit* times, which differentiate the proposed framework from algorithms

Algorithm 1 Sampling with control constraints for NOVAS

```
1: function SAMPLE($\mu_{\mathbf{T}}, \Sigma_{\mathbf{T}}$)
2:     Given: $\rho_1,\ \rho_2,$ and $\theta$
3:     Compute: $\rho_3 \leftarrow \sqrt{\frac{\rho_1^2}{2\cdot\sin^2\theta}}$
4:     Sample: $x \sim \mathcal{N}(\mu_{\mathbf{T}},\ \Sigma_{\mathbf{T}})$, where the components of $x$ are:   $x_1 = \mathbf{T}_1, \quad x_2 = \mathbf{T}_2, \quad x_3 = ||\mathbf{T}||$
5:     Project samples to satisfy hard control constraints:
           $\bar{x}_1 = \text{Proj}_{[-\rho_1/2,\rho_1/2]}(x_1)$
           $\bar{x}_2 = \text{Proj}_{[-\rho_1/2,\rho_1/2]}(x_2)$
           $\bar{x}_3 = \text{Proj}_{[\max(\rho_1,\rho_3),\rho_2]}(x_3)$
6:     $\bar{x} \leftarrow (\bar{x}_1,\ \bar{x}_2,\ \bar{x}_3)$
7:     $\delta\bar{x} \leftarrow \bar{x} - \mu_{\mathbf{T}}$
8:     return $(\bar{x}, \delta\bar{x})$
9: end function
```

presented in past works [13, 16] on deep FBSDEs. A diagram incorporating architectural changes of the deep neural network to enable these new capabilities is also presented.

A. Training a policy network invariant of initial conditions

So far in the deep FBSDEs literature [13, 14, 16, 17, 22] a fixed initial state $\mathbf{x}[0]$ has been used leading to the policy network only being able to solve the problem starting from $\mathbf{x}[0]$. However, this is a very limiting assumption in practice, more so for the planetary soft-landing problem as the probability of the spacecraft being in a specific initial state is zero. To tackle this, we relax this assumption of constant $\mathbf{x}[0]$ and consider random initial states as briefly described at the end of Section II-A of the problem formulation. We provide specific details regarding random sampling of the spacecraft's initial state in the pseudo-code Alg. 2. The inputs of Alg. 2 are: the batch-size (B), the radius of the base of the glide-slope cone (rad), the respective means and the standard deviations of the trigger altitude ($\mu_{z(0)},\ \sigma_{z(0)}$), the initial vertical velocity ($\mu_{v_z(0)},\ \sigma_{v_z(0)}$) and the initial mass ($\mu_{m(0)},\ \sigma_{m(0)}$). For sampling random horizontal velocities, the inputs σ_{hvel} and $hvel_{max}$ are the standard deviation for directions of horizontal velocity unit-vectors and the maximum horizontal velocity magnitude respectively. The boldfaced symbols $\mathbf{0}$ and $\mathbf{1}$ are vectors of zeros and ones respectively of length B and $\mathbf{I}$ is an identity matrix of size B.

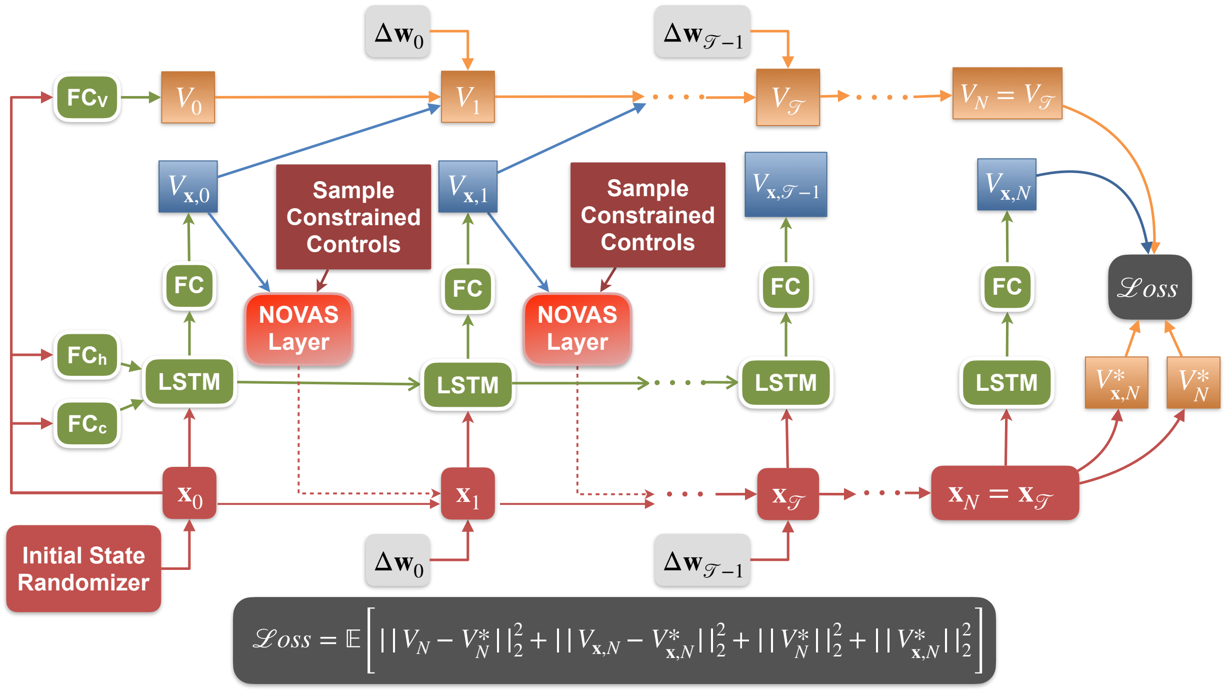
B. The Proposed Network Architecture and Pseudo-Code

The new architecture proposed in Fig. 1 features additional Fully-Connected (FC) neural network layers to enable training from multiple randomly sampled initial states. Given a batch of randomly sampled initial states, the layers FC_V, FC_h and FC_c are used to generate corresponding initial values for the value function, the hidden states of the LSTM layers and the cell states of the LSTM layers respectively.

The random initial state generation procedure not only makes our proposed approach practically meaningful as discussed in the previous subsection, but also leads to better exploration of the state-space around the landing zone.

Algorithm 2 Sampling random initial conditions

```
1:  function SAMPLE_INITIAL_STATES($B$, $rad$, $\mu_{z(0)}$, $\sigma_{z(0)}$, $\mu_{v_z(0)}$, $\sigma_{v_z(0)}$, $\mu_{m(0)}$, $\sigma_{m(0)}$, $\sigma_{hvel}$, $hvel_{max}$)
2:      $\text{radii} = rad \cdot \sqrt{\epsilon_1}$, $\epsilon_1 \sim \mathcal{U}(\mathbf{0}, \mathbf{1})$          ▷ sample $B$ uniformly distributed variables
3:      $\theta = 2\pi \cdot \epsilon_2$, $\epsilon_2 \sim \mathcal{U}(\mathbf{0}, \mathbf{1})$
4:      $x = \text{radii} \cdot \cos(\theta)$, $y = \text{radii} \cdot \sin(\theta)$
5:      $z \sim \mathcal{N}(\mu_{z(0)}\mathbf{1}, \sigma_{z(0)}\mathbf{I})$
6:      $\bar{x} = \frac{-x}{\sqrt{x^2+y^2}}$, $\bar{y} = \frac{-y}{\sqrt{x^2+y^2}}$, $\theta_{hvel} = \text{atan2}(\bar{y}, \bar{x})$ and $\bar{\theta}_{hvel} \sim \mathcal{N}(\theta_{hvel}, \sigma_{hvel}\mathbf{I})$
7:      $(v_x, v_y) = \left(hvel_{max} \cos(\bar{\theta}_{hvel}) \frac{\sqrt{x^2+y^2}}{\mu_{z(0)}}, hvel_{max} \sin(\bar{\theta}_{hvel}) \frac{\sqrt{x^2+y^2}}{\mu_{z(0)}}\right)$
8:      $v_z \sim \mathcal{N}(\mu_{v_z(0)}\mathbf{1}, \sigma_{v_z(0)}\mathbf{I})$
9:      $m \sim \mathcal{N}(\mu_{m(0)}\mathbf{1}, \sigma_{m(0)}\mathbf{I})$
10:     return $(x, y, z, v_x, v_y, v_z, m)$
11: end function
    • The above square-root, cosine and sine operations are element-wise operations
```



$\Delta \mathbf{w}_0$
$\Delta \mathbf{w}_{\mathcal{T}-1}$
FCv
V_0
V_1
$V_{\mathcal{T}}$
$V_N = V_{\mathcal{T}}$
$V_{\mathbf{x},0}$
Sample Constrained Controls
$V_{\mathbf{x},1}$
Sample Constrained Controls
$V_{\mathbf{x},\mathcal{T}-1}$
$V_{\mathbf{x},N}$
FC
NOVAS Layer
FC
NOVAS Layer
FC
FC
$\mathscr{L}oss$
FCh
LSTM
LSTM
LSTM
LSTM
$V^*_{\mathbf{x},N}$
V^*_N
FCc
$\mathbf{x}_0$
$\mathbf{x}_1$
$\mathbf{x}_{\mathcal{T}}$
$\mathbf{x}_N = \mathbf{x}_{\mathcal{T}}$
Initial State Randomizer
$\Delta \mathbf{w}_0$
$\Delta \mathbf{w}_{\mathcal{T}-1}$
$\mathscr{L}oss = \mathbb{E}\left[||V_N - V^*_N||_2^2 + ||V_{\mathbf{x},N} - V^*_{\mathbf{x},N}||_2^2 + ||V^*_N||_2^2 + ||V^*_{\mathbf{x},N}||_2^2 \right]$

Fig. 1 A new DNN architecture to handle hard control constraints, first-exit times and random initial conditions as compared to the architecture in [13]

This was found to significantly improve the performance of the trained policy when subject to stochasticity in the initial positions and the network can be deployed as a *feedback policy*. The output of Alg. 2 serves as an input to the FC_V, FC_h and FC_c neural network layers which contain Rectified Linear Unit (ReLU) nonlinearities. The LSTM layers predict $V_\mathbf{x}$ (i.e., the gradient of the value function) which is then used to compute and minimize $\mathcal{H}$ (i.e., the Hamiltonian) at each timestep within the NOVAS layer. In order to satisfy hard control constraints, a batch of constrained control samples generated by Alg. 1 are fed to the NOVAS layer during every NOVAS iteration to minimize $\mathcal{H}$ (i.e., step 2 of Alg. 5). Similar to [16], we also choose LSTM layers in the proposed architecture in order to provide robustness against the vanishing gradient problem, to reduce memory requirements by avoiding individual neural networks to approximate $V_{\mathbf{x}}$ at every time step, to generate a temporally coherent control policy and to avoid the need to feed the time step as an explicit input to the policy network. The output of the NOVAS layer is the optimal thrust that minimizes the Hamiltonian. This is fed to the dynamics model (17) to forward-propagate the state. This process is repeated for all time steps until the *first-exit* termination criteria is met.

So far, deep FBSDEs have been successfully implemented for fixed finite time-horizon problems (i.e., $\mathcal{T} = t_f$ is constant). In order to incorporate first-exit times we use a mask such that,

$$\text{mask} = \begin{cases} 1, & r_3(t) > h_{\text{tol}} \\ 0, & r_3 \leq h_{\text{tol}} \end{cases}$$

where, $h_{\text{tol}} > 0$ m, is a user-defined fixed tolerance for the altitude to determine if a landing has occurred. In the deep FBSDEs framework, due to stochastic dynamics, each trajectory could potentially have a different first-exit time. To keep track of these different first-exit times, we maintain a vector of masks of the same size as the mini-batch which is then incorporated into the equations of the forward and backward SDEs. If a particular trajectory is found to terminate early, its state, value function, and gradient of the value function are propagated forward using an identity map for the remaining time steps until the maximum simulation time. This *freezes* the components of the state vector and the value-function to the values they take on at the *first-exit* time $\mathcal{T}$. This allows gradients to freely flow from the final time step t_f to the *first-exit* time step $\mathcal{T}$. Once the end of the time horizon t_f is reached, we compute targets for the value function and its gradient using the propagated terminal states and terminal cost function (7). These are fed to a loss function that is used to train the LSTM layers and the additional neural network layers at the initial time step by minimizing the difference between the predicted and target value functions and their respective gradients. The mathematical description of the loss function is shown at the bottom of Fig. 1.

The pseudo-code (Alg. 3) provides further details regarding the forward pass of the NOVAS-FBSDE architecture and also contains discretized equations of the FSDE and the BSDE. The discretization interval (Δt seconds) is fixed and is user-defined. The total number of time steps (or discrete time intervals) is computed as $N = t_f/\Delta t$ such that when $t = t_f$ seconds, the discrete-time index $k = N$ where $k \in \{0, 1, \ldots, N\}$.

We would like to emphasize here that our framework is computationally demanding only during the training stage which does not occur onboard the spacecraft. Once fully trained, the policy can be deployed onboard the spacecraft, and it is expected to use minimal computational resources to predict an optimal thrust at every time step. This is because predicting the optimal thrust only requires a forward-pass through the trained networks which can be performed using basic linear algebra operations such as matrix-vector products and element-wise application of non-linear functions such as ReLU, tanh and sigmoid. Moreover, during flight the batch size $B = 1$ allowing all computations to be performed on

Algorithm 3 NOVAS-FBSDE with first-exit times (using Python convention for indexing and for-loops)

```
1: function FORWARD_PASS(number of time steps ($N$), altitude threshold ($h_{\text{tol}}$), mini-batch size ($B$), discretization time interval ($\Delta t$), LSTM neural-network to predict $V_{\mathbf{x}}$ ($f_{\text{LSTM}}$), diffusion matrix ($\Sigma$), drift function ($f$), Hamiltonian function ($\mathcal{H}$), running cost function ($l$), terminal cost function ($\phi$), inputs and hyperparameters for NOVAS Layers ($NOVAS\_inputs$), initial value-function network ($\text{FC}_{\text{V}}$), number of LSTM hidden layers ($H$), neural networks to predict initial LSTM-states $\{\text{FC}_{\text{c}}{}^{i},\ \text{FC}_{\text{h}}{}^{i}\}_{i=1}^{H}$, radius of base of the inverted glide-slope cone ($rad$))
2:     Initialize: mask $\leftarrow 0_{B\times 1}$
                   $\mathbf{x}[:,\ 0,\ :] \leftarrow$ sample_initial_states $(\ldots)$          ▷ Algorithm 2
                   where, $\mathbf{x}$ is a tensor of shape $(B \times N \times 7)$
   (Predict the initial value-function and the initial cell and hidden states of the LSTM for all batch elements)
3:     for $b = 0 : B-1$ (in parallel) do
4:         $V[b,0] = \text{FC}_{\text{V}}(\mathbf{x}[b,0])$
5:         for $i = 1 : H+1$ (in parallel) do
6:             $h_i[b,0] = \text{FC}_{\text{h}}{}^{i}(\mathbf{x}[b,0])$
7:             $c_i[b,0] = \text{FC}_{\text{c}}{}^{i}(\mathbf{x}[b,0])$
8:         end for
9:     end for

   (Forward propagate the dynamics and value function trajectories)
10:    for $k = 0 : N-1$ do
11:        for $b = 0 : B-1$ (in parallel) do
12:            if $r_3 > h_{\text{tol}}$ then
13:                mask$[b] \leftarrow 1$
14:            else
15:                mask$[b] \leftarrow 0$
16:            end if
17:            sample noise, $\Delta w[b,k] \sim \mathcal{N}(\mathbf{0},\ \sqrt{\Delta t}\mathbf{I})$     ▷ zero mean vector which has same dimensionality as $\mathbf{x}$
18:            $\Big(V_{\mathbf{x}}[b,k],\ \big\{h_i[b,k+1],\ c_i[b,k+1]\big\}_{i=1}^{H}\Big) \leftarrow f_{\text{LSTM}}\Big(\mathbf{x}[b,k],\ \big\{h_i[b,k],\ c_i[b,k]\big\}_{i=1}^{H}\Big)$
19:            $\mathbf{T}^*[b,k] \leftarrow$ NOVAS_Layer $(\mathbf{x}[b,k],\ V_{\mathbf{x}}[b,k],\ \mathcal{H},\ f,\ NOVAS\_inputs)$     ▷ Algorithm 4
                                                    ▷ warm-start NOVAS with $\mathbf{T}^*[b,k-1]$ for $k > 0$
20:            FSDE: $\mathbf{x}[b,k+1] = \mathbf{x}[b,k] + \mathbf{mask}[b] \odot \Big(f\big(\mathbf{x}[b,k],\ \mathbf{T}^*[b,k]\big)\,\Delta t + \Sigma\big(\mathbf{x}[b,k],\ k\big)\,\Delta w[b,k]\Big)$
21:            BSDE: $V[b,k+1] = V[b,k] + \mathbf{mask}[b] \odot \Big(-l\big(\mathbf{x}[b,k],\ \mathbf{T}^*[b,k]\big)\,\Delta t + V_{\mathbf{x}}[b,k]^{\text{T}}\Sigma\big(\mathbf{x}[b,k],\ k\big)\,\Delta w[b,k]\Big)$
22:        end for
23:    end for

   (Compute loss function using the predicted value and its gradient)
24:    for $b = 0 : B-1$ (in parallel) do
25:        $V^*[b,N] = \phi\big(\mathbf{x}[b,N]\big),\ V^*_{\mathbf{x}}[b,N] = \dfrac{\partial \phi\big(\mathbf{x}[b,N]\big)}{\partial \mathbf{x}}$
26:        $V_{\mathbf{x}}[b,N] = f_{\text{LSTM}}\Big(\mathbf{x}[b,N],\ \big\{h_i[b,N],\ c_i[b,N]\big\}_{i=1}^{H}\Big)$
27:    end for
28:    $\mathcal{L}oss = \dfrac{1}{B}\sum_{b=1}^{B}\Big\{\big\|V[b,N]-V^*[b,N]\big\|_2^2 + \big\|V_{\mathbf{x}}[b,N]-V^*_{\mathbf{x}}[b,N]\big\|_2^2 + \big\|V^*[b,N]\big\|_2^2 + \big\|V^*_{\mathbf{x}}[b,N]\big\|_2^2\Big\}$
29:    return $\mathcal{L}oss$
30: end function
```

- In steps 20 and 21, for a given batch index b, the same **mask**$[b]$ is applied to all 7 elements of the state vector **x**.
- The parallel for-loops over $b = 0 : B-1$ and $i = 1 : H+1$ can be easily implemented with vectorized operations and batched operations, using any deep learning framework such as PyTorch [23] or TensorFlow [24].

a CPU and thus eliminating the need for a GPU onboard the spacecraft.

C. The NOVAS Layer

In this section, we summarize the algorithmic implementation of the NOVAS Layer from [13] for the convenience of the reader. Since the goal of the NOVAS Layer is to solve the problem proposed in (16), we must provide all the necessary inputs for the computation of $\mathcal{H}$ such as the current state vector $\mathbf{x}(t)$, the gradient of the value-function $V_{\mathbf{x}}(t)$ and the system's drift vector, $f(\mathbf{x},\, \mathbf{T})$. In addition to these quantities, we also have tunable hyperparameters, referred to in Algorithms 3 and 4 as $NOVAS\_inputs$, that directly affect the performance of the algorithm. These values include the initial sampling mean and variance $(\mu_{\mathbf{T}}, \Sigma_{\mathbf{T}})$, a scalar learning rate (α), number of NOVAS samples (M), number of NOVAS inner-loop iterations (N_{iter}), some arbitrarily small positive number (ε) indicating minimum variance, and a user-defined shape function (S). The quantity ε and the function S are chosen in such a way as to improve the stability of the algorithm while all other values directly affect the convergence-rate and the accuracy of the output control solution. The hyperparameter values used to obtain the simulation results are presented in Table 1.

Algorithm 4 NOVAS_LAYER

```
1: function NOVAS_LAYER($\mathbf{x}[b,t]$, $V_{\mathbf{x}}[b,t]$, $\mathcal{H}$, $f$, $NOVAS\_inputs \rightarrow \big[$ initial sampling mean and variance
   $(\mu_{\mathbf{T}}, \Sigma_{\mathbf{T}})$, learning rate ($\alpha$), shape function ($S$), number of samples ($M$), number of iterations ($N_{\text{iter}}$), minimum
   variance ($\epsilon$)$\big]$)

2:     for $n = 0 : N_{\text{iter}} - 2$ (off-graph operations) do
3:         $(\mu_{\mathbf{T}},\, \Sigma_{\mathbf{T}}) \leftarrow$ NOVAS_STEP $(\mathbf{x}[b,t],\, V_{\mathbf{x}}[b,t],\, \mathcal{H},\, f,\, \mu_{\mathbf{T}},\, \Sigma_{\mathbf{T}},\, \alpha,\, S,\, M,\, \epsilon)$     ▷ Algorithm 5
4:     end for

       (final iteration is on-graph)
5:     $(\mu_{\mathbf{T}},\, \Sigma_{\mathbf{T}}) \leftarrow$ NOVAS_STEP $(\mathbf{x}[b,t],\, V_{\mathbf{x}}[b,t],\, \mathcal{H},\, f,\, \mu_{\mathbf{T}},\, \Sigma_{\mathbf{T}},\, \alpha,\, S,\, M,\, \epsilon)$
6:     $\mathbf{T}^* \leftarrow \left(\mu_{\mathbf{T},1}, \mu_{\mathbf{T},2}, \sqrt{(\mu_{\mathbf{T},3})^2 - (\mu_{\mathbf{T},1})^2 - (\mu_{\mathbf{T},2})^2}\right)$
7:     return $\mathbf{T}^*$
8: end function
```

During each iteration of the NOVAS Layer (Algorithms 4 and 5), we approximate the gradient (A.4)(see appendix for derivation) of the stochastic approximation of the objective w.r.t $\mu_{\mathbf{T}}$ through sampling. To do this, we sample M different values of horizontal thrust and thrust-norm using univariate Gaussian distributions with mean $\mu_{\mathbf{T}}$ and covariance $\Sigma_{\mathbf{T}}$. These are then projected to appropriate intervals to satisfy the hard control constraints specified in the problem formulation (as shown in Alg. 1). For initialization, the mean vector can be populated using random values within the admissible control set. However, in our case, we set the initial $\mu_{\mathbf{T}} = (0, 0, \rho_1)$ for the first ($k = 0$) time step, and warm-start the NOVAS Layers for subsequent time steps ($k > 0$) using the optimal thrust values from the respective previous time steps (where "time step" refers to the for-loop $k = 0 : N - 1$ in Alg. 3). Note that the first $N_{\text{iter}} - 1$ iterations of the NOVAS Layer are *off-graph operations*, meaning that they are not part of the deep learning framework's

Algorithm 5 NOVAS_STEP

```
1:  function NOVAS_STEP( $\mathbf{x}[b,t]$, $V_{\mathbf{x}}[b,t]$, $\mathcal{H}$, $f$, $\mu_{\mathbf{T}}$, $\Sigma_{\mathbf{T}}$, $\alpha$, $S$, $M$, $\epsilon$)
2:      Generate M control samples: $(\bar{x}^m,\ \delta\bar{x}^m) \leftarrow \mathbf{SAMPLE}(\mu_{\mathbf{T}}, \Sigma_{\mathbf{T}})$,    $m = 1, ..., M$        ▷ Algorithm 1
3:      Transform: $\mathbf{T}^m \leftarrow \left(\bar{x}_1^m, \bar{x}_2^m, \sqrt{(\bar{x}_3^m)^2 - (\bar{x}_1^m)^2 - (\bar{x}_2^m)^2}\right)$
4:      for $m = 0 : M - 1$ (in parallel) do
5:          Evaluate: $F^m = -\mathcal{H}(\ \mathbf{x}[b,t],\ V_{\mathbf{x}}[b,t],\ \mathbf{T}^m, f)$        ▷ using eqn. (20)
6:          Shift: $F^m = F^m - \min_m F^m$
7:          Apply (element-wise) the shape function: $S^m = S(F^m)$
8:          Normalize: $S^m = S^m / \sum_{m=1}^{M} S^m$
9:      end for
    (Perform thrust mean and variance update)
10:     $\mu_{\mathbf{T}} = \mu_{\mathbf{T}} + \alpha \sum_{m=1}^{M} S^m \delta\bar{x}^m$        ▷ derived in appendix (A.7)
11:     $\delta\bar{x}^m \leftarrow \bar{x}^m - \mu_{\mathbf{T}}$
12:     $\Sigma_{\mathbf{T}} = diag\left(\sqrt{\sum_{m=1}^{M} S^m (\delta\bar{x}^m)^2 + \epsilon}\right)$
13:     return $(\mu_{\mathbf{T}}, \Sigma_{\mathbf{T}})$
14: end function
```

compute graph and therefore not considered during backpropagation. A compute graph is built to approximate gradients, by means of automatic differentiation, of the loss function with respect to the weights of the deep neural network. Taking the first $N_{\text{iter}} - 1$ iterations *off-the-graph* can be seen as "warm-starting" the last iteration which is performed *on-the-graph*. This procedure has negligible effect on the training of the neural network and can be performed because NOVAS does not overfit to the specific number of inner-loop iterations as demonstrated in [13]. By performing the first $N_{\text{iter}} - 1$ iterations within the NOVAS Layers at each time step *off-the-graph*, we significantly reduce the length of the overall compute graph (i.e., over all N time steps) which speeds up training and enables us to use this approach to train policies for long time horizons.

V. Simulation Results

We demonstrate the capabilities of the NOVAS-FBSDE algorithm by training it to perform a safe landing maneuver on the Martian surface in simulation. A full list of the system and algorithm parameters is presented in Table 1. For our simulations, we consider the discretized equations of motion (17), and assume a constant value for the Martian acceleration due to gravity, $g = 3.72 m/s^2$. During training, we allow a maximum simulation time of $t_f = 20$ seconds and set a time discretization of $\Delta t = 0.05$ seconds. To achieve the results presented below, the network was trained for 16,500 iterations adding complexity to the task progressively to facilitate learning. First, a model was trained until convergence for 9,800 iterations and a fixed learning rate of 0.0005. During training of this first model, only the initial position (i.e., downrange and altitude) was randomly sampled. Using this pre-trained model, a new training run was initiated for 6,700 iterations and a learning rate schedule of [0.0001, 0.00002]. For the second run, the entire state was randomly sampled according to Alg. 2. We found this two-step training procedure to be the most effective as the pre-trained policy serves as a good starting point and allows for more efficient exploration than starting from scratch. For the computation of the discretized-BSDE in (18), the parameters used to compute the terminal cost (7), the running state cost i.e., the glide-slope constraint cost in (10), and the control cost are also provided in Table 1. Based on the mass-flowrate equation for gimbaled rockets [3], our control cost takes the form of an $\mathcal{L}^1$-norm to penalize fuel consumption which is given by,

$$||\mathbf{T}(t)||_{\mathcal{L}^1} = \int_0^{\mathcal{T}} \sqrt{\mathbf{T}_1^2(t) + \mathbf{T}_2^2(t) + \mathbf{T}_3^2(t)}\ \mathrm{d}t$$

Furthermore, we considered a *first-exit* formulation, where a particular simulated trajectory is terminated if the spacecraft is found to be within a user-defined altitude tolerance value of $h_{\text{tol}} <= 10^{-3}m$ from the Martian surface. Similar to [12], we assume that a touchdown speed of higher than 5 ft/s* (1.52 m/s) in any direction leads to a crash landing. The cost function hyperparameters (i.e., $Q_x, Q_y, Q_z, Q_{v_x}, Q_{v_y}, Q_{v_z}, Q_m, q_-, q_+, q_{\mathcal{L}^1}$) were determined through experimentation until desirable results were obtained. In practice, one could carry out a grid search over the hyperparameters space. Table 2 provides the logic to categorize each trajectory as either *No Landing, Safe Landing or Crash Landing*.

During testing, to allow the spacecraft to get close enough to the ground (i.e., below an altitude of h_{tol}), we increased the maximum simulation time to $t_f = 30$ seconds. Note that this is 10 seconds higher than the maximum simulation time considered during training (i.e., $t_{f,\,\text{training}} = 20$ seconds). We hypothesize that because our policy behaves like a *feedback policy*, we can deploy the learnt policy for a much longer duration than what it was trained for. As shown in Figures 2, 3, and 4 our policy can successfully steer all trajectories towards the landing location and safely land while staying as much as possible inside the glide-slope cone enforced as a soft-constraint. Specifically, Fig. 2 shows trajectories of position states (top row) and the corresponding velocity states (bottom row) of 1024 independent test trials. The random initial states are sampled as per Alg. 2. Each trial uses 50 NOVAS samples and 6 NOVAS iterations per time step. Since we have formulated this problem as a first-exit problem, all velocities are zeroed out once a landing or a crash happens. To gain more insight on the true performance of the learned policy, we have computed the landing statistics based on the 1024 test trials. We summarize our observations in Table 3 wherein the correct category is chosen based on the logic provided in Table 2. We noticed that just increasing the NOVAS iterations by 1 (per time step) we were able to achieve 100% safe landing at test time.

Finally, we demonstrate through empirical evidence that our constrained sampling scheme satisfies the hard constraints on the thrust. For our simulations, we chose $\theta = \pi/4$ to ensure that the ground always is in the field of view of the camera and other sensors on the base of the spacecraft. Thus, $\rho_3 = \sqrt{\dfrac{\rho_1^2}{2 \cdot \sin^2(\pi/4)}} = \rho_1$. In Fig. 5, we show randomly sampled instances of the control-norm $\Big($i.e., $||\mathbf{T}(t)||\Big)$ to demonstrate satisfaction of hard constraints on the thrust-norm (i.e., the solid blue lines never cross the bounds indicated by dashed blue lines). We additionally plot

*NASA specifications: `https://www.nasa.gov/mission_pages/station/structure/elements/soyuz/landing.html`

Table 1 System and Algorithm Parameters

Description	Symbol	Value	Units
System Parameters			
Thrust bounds	$(\rho_1,\ \rho_2)$	$(4.97\times10^3,\ 1.334\times10^4)$	Newtons
Minimum Admissible Glide-Slope Angle	γ	$\pi/4$	rad
Maximum Allowable Angle between **T** and $\hat{\mathbf{n}}$	θ	$\pi/4$	rad
Glide-Slope Cone Base Radius	rad	80	m
Acceleration due to Gravity for Mars	g	3.7144	m/s^2
Effective Exhaust Velocity	c	553.31	m/s
Altitude Tolerance for Landing	h_{tol}	10^{-3}	m
Dynamics Diffusion Matrix	Σ	$10^{-4}\cdot\mathbf{I}_{3\times3}$	-
Initial Trigger Altitude Mean	$\mu_{z(0)}$	80	m
Initial Trigger Altitude Standard Deviation	$\sigma_{z(0)}$	2.5	m
Initial Horizontal Velocity Orientation Standard Deviation	σ_{hvel}	5	degrees
Initial Horizontal Velocity Maximum Magnitude	$hvel_{max}$	10	m/s
Initial Vertical Velocity Mean	$\mu_{v_z(0)}$	-10	m/s
Initial Vertical Velocity Standard Deviation	$\sigma_{v_z(0)}$	2.5	m/s
(Dry Mass, Initial Mass Mean)	$(m_d,\ \mu_{m(0)})$	(1505, 1905)	kg
Initial Mass Standard Deviation	$\sigma_{m(0)}$	10	kg
Maximum Simulation Time (Training)	t_f	20	s
Maximum Simulation Time (Testing)	t_f	30	s
Discretization Time Interval	Δt	0.05	s
NOVAS-FBSDE Algorithm Parameters			
Cost on Terminal Position	(Q_x, Q_y, Q_z)	(2.5, 2.5, 2.5)	-
Cost on Terminal Velocity	$(Q_{v_x}, Q_{v_y}, Q_{v_z})$	(5.0, 5.0, 10.0)	-
Cost on Terminal Mass	Q_m	10.0	-
Glide-Slope Violation Penalty	$(q_-,\ q_+)$	(1.0, 0.005)	-
$\mathcal{L}^1$-norm Fuel Consumption Coefficient	$q_{\mathcal{L}^1}$	0.00055	-
Hidden and Cell State Neurons per Layer	-	8	-
Optimizer	-	Adam	-
NOVAS Shape Function	$S(\cdot)$	$\exp(\cdot)$	-
NOVAS Initial Sampling Variance	$\Sigma_{\mathbf{T}}$	$\mathbf{diag}(500^2,\ 500^2,\ 1000^2)$	-
NOVAS Initial Sampling Mean	$\mu_{\mathbf{T}}$	(0.0, 0.0, 5000)	-
NOVAS Iteration Learning Rate	α	1.0	-
Mini-batch size (training)	B	32	-
Mini-batch size (testing)	B	1024	-
Number of LSTM Hidden Layers	H	2	-

the norm of the velocity vector in red and indicate the time step after which the distance to the landing zone is less than $10\,m$ with the dashed green line. Available research literature affirms that the optimal thrust profile (i.e., one that consumes the least amount of fuel overall) for the deterministic 3D PDG problem is *bang-bang* in nature; implying that our controller has converged to a *sub-optimal* control policy. A majority of the samples illustrated in Fig. 5 show a

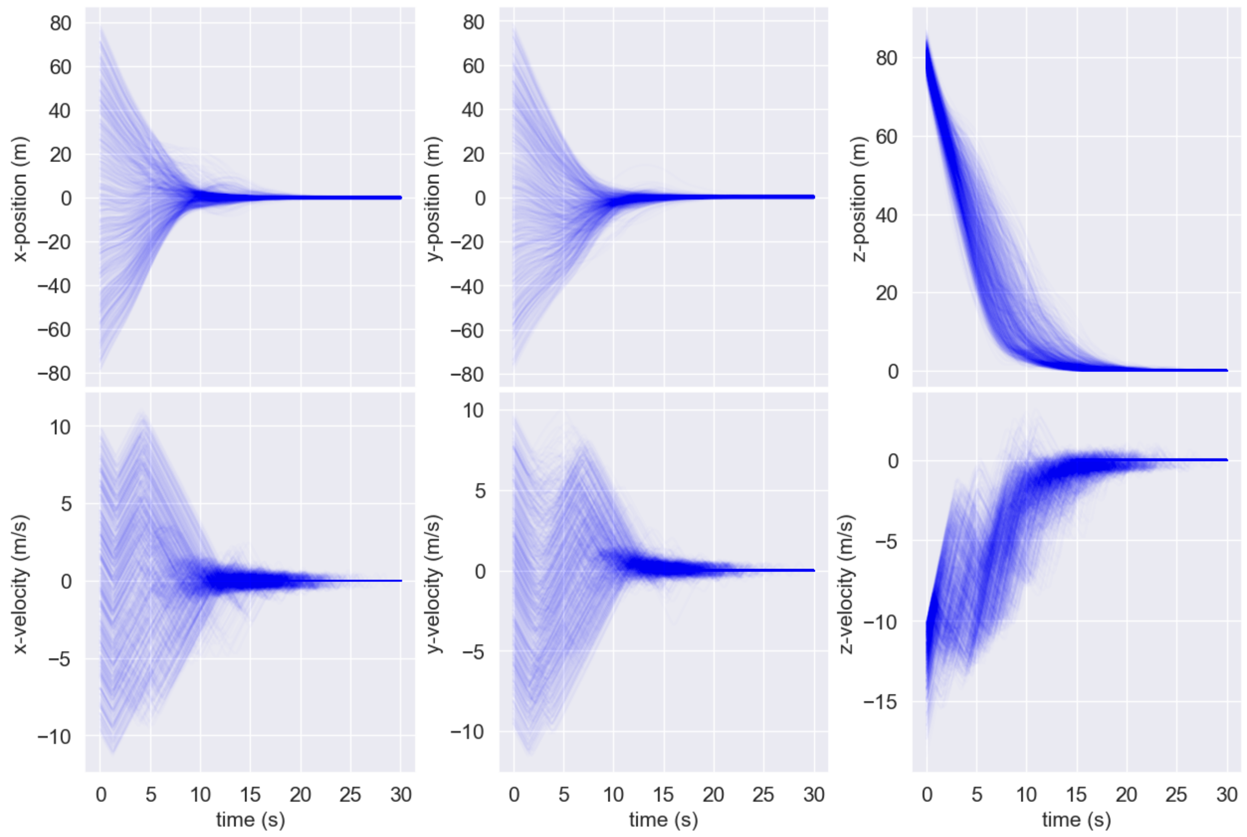
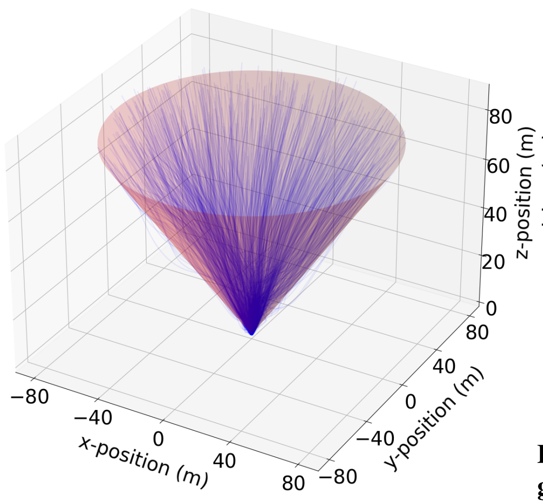


Fig. 2 Position and velocity trajectories of 1024 test trials showing safe landing in all instances



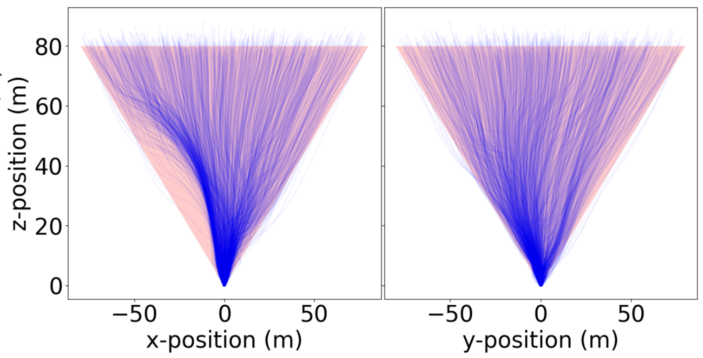


Fig. 4 xz (*left*) and yz (*right*) cross-sectional views of the glide-slope soft-constraint satisfaction

Fig. 3 Three-dimensional view of glide-slope soft-constraint satisfaction

highly undesirable thrust profile to fly on actual hardware. Although we acknowledge this outcome, we would like to point out that in most cases, the undesired oscillations arise at low speeds and low altitudes which we hypothesize as

Table 2 Logic to determine the type of landing

Simulation Outcome	Height and Speed Criteria
No Landing	height > $10^{-3}m$
Safe Landing	height ≤ $10^{-3}m$ AND speed ≤ $1.52m/s$
Crash Landing	height ≤ $10^{-3}m$ AND speed > $1.52m/s$

Table 3 Landing statistics for 1024 trials with maximum simulation time of $t_{f,\,\text{test}} = 30$ seconds

NOVAS inner-loop iterations	NOVAS samples	Not landed	Safely landed	Crashed
5	50	0.2%	99.8%	0.0%
6	50	0.0%	100.0%	0.0%

the controller's response in trying to achieve the softest landing possible (i.e., it prioritizes minimizing the terminal state cost over minimizing the fuel consumption when very close to the landing zone). In some instances, a highly sub-optimal (i.e., a non *bang-bang-like*) thrust profile is also observed before the $10\,m$ mark. This is a limitation of modeling the thrust-norm as a continuous random variable with a truncated Gaussian distribution (see line 4 of Alg.1), instead of a discrete random variable. This allows for a continuous thrusting profile which in the presence of high levels of stochasticity can lead to a *sub-optimal* control policy. We suspect this to be the primary reason for the departure from the optimal *bang-bang* thrust profile. This can be overcome by modeling the thrust-norm using discrete distributions such as the categorical distribution wherein the discrete values are ρ_1 and ρ_2, while using the truncated Gaussian distribution to model the horizontal thrust components similar to Alg.1. Because the categorical distribution is also part of the exponential family, the theory of Stochastic Search applies and it can be used to optimize the parameters of the distribution. We invite the interested reader to refer to the work in [25] for more details, and we postpone the investigation of extending NOVAS to use a discrete distribution for the thrust-norm to future work.

VI. Conclusion

In this paper, we presented a novel approach to solve the constrained three-dimensional stochastic soft-landing problem using Long-Short Term Memory-based deep recurrent neural networks and the differentiable non-convex optimization layer called the Non-Convex Optimization Via Adaptive Stochastic Search layer, within the deep Forward-Backward Stochastic Differential Equation framework for end-to-end differentiable $\mathcal{L}^1$ stochastic optimal control. Our approach does not rely on convexification of the constraints or linearization of the dynamics. Through simulations, we demonstrated satisfaction of hard thrusting (i.e., control) constraints and soft state constraints, as well as robustness to the spacecraft's initial conditions and external disturbances. Not only does our policy safely land all test cases, but it also exhibits properties of a feedback policy, thereby allowing it to be deployed for longer than the maximum simulation

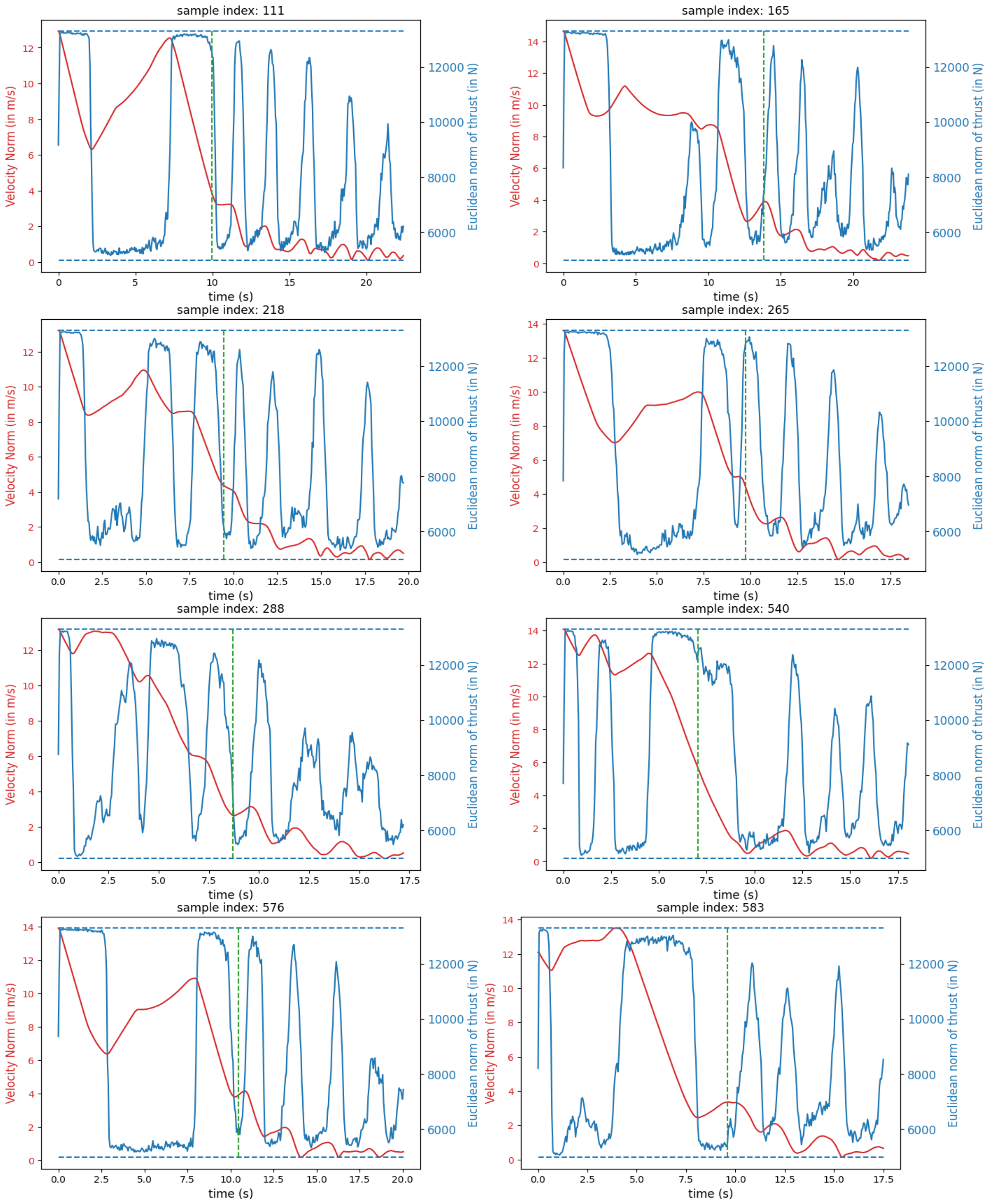


Fig. 5 Randomly sampled test instances wherein the dashed blue lines are ρ_1 and ρ_2 showing satisfaction of hard constraints on the thrust-norm. The dashed green line indicates the time step when the distance to the landing zone is less than $10\,m$ to distinguish the thrust-norm profiles in the two intervals.

time during training. Thus, once trained offline, our policy does not require *on-the-go* re-planning as compared to other deterministic methods in literature and can output an optimal control by performing a forward-pass through a neural network and the Non-Convex Optimization Via Adaptive Stochastic Search layer. By making the framework robust to the initial state at the start of the descent stage, not only is the glide-slope of the descent trajectory regulated, but our trained policy also has a higher tolerance for errors made by the pre-descent stage controllers and can takeover from the previous stage starting in a wide radius around the landing zone. We believe these successful results serve as a stepping stone for future research directions in using deep-learning-based stochastic optimal control for planetary soft-landing.

References

[1] Cherry, G. W., "A general, explicit, optimizing guidance law for rocket-propelled spaceflight," *American Institute of Aeronautics and Astronautics*, 1964. https://doi.org/10.2514/6.1964-638.

[2] Klumpp, A. R., "Apollo Lunar Descent Guidance," *Automatica*, Vol. 10, No. 2, 1974, pp. 133–146. https://doi.org/10.1016/0005-1098(74)90019-3.

[3] Ross, M. I., "How to Find Minimum-Fuel Controllers," *AIAA Guidance, Navigation, and Control Conference and Exhibit, Providence, Rhode Island*, 16 - 19 August, 2004.

[4] Lu, P., "Propellant-optimal powered descent guidance," *Journal of Guidance, Control, and Dynamics*, Vol. 41, No. 4, 2018, pp. 813–826. https://doi.org/10.2514/1.G003243.

[5] Lu, P., "Augmented apollo powered descent guidance," *Journal of Guidance, Control, and Dynamics*, Vol. 42, No. 3, 2019, pp. 447–457. https://doi.org/10.2514/1.G004048.

[6] You, S., Wan, C., Dai, R., Lu, P., and Rea, J. R., "Learning-based optimal control for planetary entry, powered descent and landing guidance," *AIAA Scitech 2020 Forum*, 2020, p. 0849. https://doi.org/10.2514/1.G004928.

[7] Sánchez-Sánchez, C., and Izzo, D., "Real-time optimal control via deep neural networks: study on landing problems," *Journal of Guidance, Control, and Dynamics*, Vol. 41, No. 5, 2018, pp. 1122–1135. https://doi.org/10.2514/1.G002357.

[8] Dueri, D., Açıkmeşe, B., Scharf, D. P., and Harris, M. W., "Customized real-time interior-point methods for onboard powered-descent guidance," *Journal of Guidance, Control, and Dynamics*, Vol. 40, No. 2, 2017, pp. 197–212. https://doi.org/10.2514/1.G001480.

[9] Szmuk, M., Reynolds, T. P., and Açıkmeşe, B., "Successive convexification for real-time six-degree-of-freedom powered descent guidance with state-triggered constraints," *Journal of Guidance, Control, and Dynamics*, Vol. 43, No. 8, 2020, pp. 1399–1413. https://doi.org/10.2514/1.G004549.

[10] Liu, X., "Fuel-optimal rocket landing with aerodynamic controls," *Journal of Guidance, Control, and Dynamics*, Vol. 42, No. 1, 2019, pp. 65–77. https://doi.org/10.2514/1.G003537.

[11] Ridderhof, J., and Tsiotras, P., "Minimum-Fuel Closed-Loop Powered Descent Guidance with Stochastically Derived Throttle Margins," *Journal of Guidance, Control, and Dynamics*, Vol. 44, No. 3, 2021, pp. 537–547. https://doi.org/10.2514/1.G005400.

[12] Exarchos, I., Theodorou, E. A., and Tsiotras, P., "Optimal Thrust Profile for Planetary Soft Landing Under Stochastic Disturbances," *Journal of Guidance, Control, and Dynamics*, Vol. 42, No. 1, 2019, pp. 209–216. https://doi.org/10.2514/1.G003598.

[13] Exarchos, I., Pereira, M. A., Wang, Z., and Theodorou, E. A., "NOVAS: Non-convex Optimization via Adaptive Stochastic Search for End-to-End Learning and Control," *Published as a conference paper at the International Conference on Learning Representations (ICLR)*, 2021.

[14] Pereira, M. A., Wang, Z., Exarchos, I., and Theodorou, E. A., "Safe optimal control using stochastic barrier functions and deep forward-backward sdes," *Published as a conference paper at the Conference on Robot Learning (CoRL)*, 2020.

[15] Itô, K., "Stochastic Integral," *Proceedings of the Imperial Academy*, 8, Vol. 20, 1944, pp. 519, 524. https://doi.org/10.3792/pia/1195572786.

[16] Pereira, M., Wang, Z., Exarchos, I., and Theodorou, E. A., "Learning deep stochastic optimal control policies using forward-backward sdes," *Published as a conference paper at Robotics: Science and Systems (RSS)*, 2019.

[17] Pereira, M., Wang, Z., Chen, T., Reed, E., and Theodorou, E., "Feynman-Kac Neural Network Architectures for Stochastic Control Using Second-Order FBSDE Theory," *Learning for Dynamics and Control*, PMLR, 2020, pp. 728–738.

[18] Exarchos, I., and Theodorou, E. A., "Stochastic optimal control via forward and backward stochastic differential equations and importance sampling," *Automatica*, Vol. 87, 2018, pp. 159–165.

[19] Kingma, D. P., and Ba, J., "Adam: A method for stochastic optimization," *Published as a conference paper at the International Conference on Learning Representations (ICLR)*, 2015.

[20] Gisiro, M., "Continuous Markov processes and stochastic equations," *Rendiconti del Circolo Matematico di Palermo*, Vol. 4, No. 1, 1955, pp. 48–90. https://doi.org/10.1007/BF02846028.

[21] Zhou, E., and Hu, J., "Gradient-based adaptive stochastic search for non-differentiable optimization," *IEEE Transactions on Automatic Control*, Vol. 59, No. 7, 2014, pp. 1818–1832.

[22] Wang, Z., Lee, K., Pereira, M. A., Exarchos, I., and Theodorou, E. A., "Deep forward-backward sdes for min-max control," *2019 IEEE 58th Conference on Decision and Control (CDC)*, IEEE, 2019, pp. 6807–6814. https://doi.org/10.1109/CDC40024.2019.9028871.

[23] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al., "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, Vol. 32, 2019, pp. 8026–8037.

[24] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al., “Tensorflow: A system for large-scale machine learning,” *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, 2016, pp. 265–283.

[25] Chen, X., Zhou, E., and Hu, J., “Discrete optimization via gradient-based adaptive stochastic search methods,” *IISE Transactions*, Vol. 50, No. 9, 2018, pp. 789–805.

Appendix A - NOVAS Derivation

In this paper, we consider non-convex optimization problems where the optimal control cannot be computed as an analytical solution and therefore we rely on the use of a novel approach introduced by Exarchos et. al. in [13], by the name of NOVAS. NOVAS stands for *Non-convex Optimization Via Adaptive Stochastic Search*. NOVAS is designed to tackle very general non-convex optimization problems, and is inspired by a well-researched method used across the field of stochastic optimization known as Gradient-based Adaptive Stochastic Search (GASS) [21]. We summarize, in this section, the main ideas and the derivation from the work [13] for the convenience of the reader. For more details and other applications of NOVAS, we invite the interested reader to refer to [13]. In general, GASS addresses a maximization problem (for minimization we consider the negative of the objective function) of the following form,

$$\mathbf{T}^* \in \arg\max_{\mathbf{T}\in\mathcal{U}} \mathcal{H}(\mathbf{T}), \quad \mathcal{U} \subseteq \mathbb{R}^n \tag{A.1}$$

where, $\mathcal{U}$ is non-empty and compact, and $\mathcal{H} : \mathcal{U} \to \mathbb{R}$ is a real-valued function that may be non-convex, discontinuous and non-differentiable. Given that $\mathcal{H}(\mathbf{T})$ is allowed to be very general, this function may be defined by an analytical expression or a neural network. GASS allows us to solve the above maximization problem through a stochastic approximation. For this we first convert the deterministic problem above into a stochastic one in which $\mathbf{T}$ is a random variable. Moreover, we assume that $\mathbf{T}$ has a probability distribution $\rho(\mathbf{T};\theta)$ from the exponential family and is parameterized by θ. Using this approximation, we now shift from explicitly maximizing $\mathcal{H}(\mathbf{T})$ to maximizing a lower bound on $\mathcal{H}(\mathbf{T})$ given by,

$$\theta^* = \arg\max_{\theta} \underbrace{\int \mathcal{H}(\mathbf{T})\rho(\mathbf{T};\theta)\,\mathrm{d}\mathbf{T}}_{\triangleq\mathbb{E}[\mathcal{H}(\mathbf{T})]} \tag{A.2}$$

It is common practice to introduce a natural log and a shape function, $S(\cdot)$ with properties of being a continuous, non-negative and non-decreasing function. Due to these properties, the optima of the new problem remain unchanged. The problem then becomes,

$$\theta^* = \arg\max_{\theta} \left\{ \underbrace{\ln \int S\big(\mathcal{H}(\mathbf{T})\big)\rho(\mathbf{T};\theta)\,\mathrm{d}\mathbf{T}}_{=\ln\mathbb{E}\left[S\left(\mathcal{H}(\mathbf{T})\right)\right]} \right\} \tag{A.3}$$

Notice that the optimization is not with respect to $\mathbf{T}$ anymore and is instead with respect to the parameters of the distribution on $\mathbf{T}$. Thus, we can attempt to solve the above problem with gradient-based approaches since the non-differentiability of $\mathcal{H}$ with respect to $\mathbf{T}$ has now been circumvented. Taking the gradient of the objective we have,

$$\begin{aligned}
\nabla_\theta \ln \int S\big(\mathcal{H}(\mathbf{T})\big)\, \rho(\mathbf{T};\theta)\, \mathrm{d}\mathbf{T} &= \frac{\int S\big(\mathcal{H}(\mathbf{T})\big)\, \nabla_\theta \rho(\mathbf{T};\theta)\, \mathrm{d}\mathbf{T}}{\int S\big(\mathcal{H}(\mathbf{T})\big)\, \rho(\mathbf{T};\theta)\, \mathrm{d}\mathbf{T}} \\
&= \frac{\int S\big(\mathcal{H}(\mathbf{T})\big)\, \nabla_\theta \rho(\mathbf{T};\theta)\, \frac{\rho(\mathbf{T};\theta)}{\rho(\mathbf{T};\theta)} \mathrm{d}\mathbf{T}}{\int S\big(\mathcal{H}(\mathbf{T})\big)\, \rho(\mathbf{T};\theta)\, \mathrm{d}\mathbf{T}} \\
&= \frac{\int S\big(\mathcal{H}(\mathbf{T})\big)\, \nabla_\theta \ln \rho(\mathbf{T};\theta)\, \rho(\mathbf{T};\theta) \mathrm{d}\mathbf{T}}{\int S\big(\mathcal{H}(\mathbf{T})\big)\, \rho(\mathbf{T};\theta)\, \mathrm{d}\mathbf{T}} \quad \text{(also known as the log-trick)} \\
&= \frac{\mathbb{E}\big[S\big(\mathcal{H}(\mathbf{T})\big)\, \nabla_\theta \ln \rho(\mathbf{T};\theta)\big]}{\mathbb{E}\Big[S\big(\mathcal{H}(\mathbf{T})\big)\Big]} \qquad \text{(A.4)}
\end{aligned}$$

The *log-trick* allows us to approximate the gradient by sampling. This makes this method amenable to GPUs or vectorized operations. Since $\rho(\mathbf{T};\theta)$ belongs to the exponential family we can compute an analytical form for the gradient inside the expectation. Distributions belonging to the exponential family generally take the following form,

$$\rho(\mathbf{T};\theta) = h(\mathbf{T})\ \exp(\theta^{\mathrm{T}} Z(\mathbf{T}) - A(\theta))$$

where, θ is the vector of natural parameters, Z is the vector of sufficient statistics and A is the log-partition function. For a multivariate Gaussian we can obtain analytical expressions for each of these as follows:

$$\begin{aligned}
P(\mathbf{T};\mu_\mathbf{T},\,\Sigma_\mathbf{T}) &= \frac{1}{\sqrt{(2\pi)^n|\Sigma_\mathbf{T}|}} \exp\Big(-\frac{1}{2}(\mathbf{T}-\mu_\mathbf{T})^{\mathrm{T}}\Sigma_\mathbf{T}^{-1}(\mathbf{T}-\mu_\mathbf{T})\Big) \qquad \text{(A.5)} \\
&= \underbrace{\frac{1}{\sqrt{(2\pi)^n|\Sigma_\mathbf{T}|}} \exp\Big(-\frac{1}{2}\mathbf{T}^{\mathrm{T}}\Sigma_\mathbf{T}^{-1}\mathbf{T}\Big)}_{h(\mathbf{T})} \exp\Big(\mathbf{T}^{\mathrm{T}}\Sigma_\mathbf{T}^{-1}\mu_\mathbf{T} - \frac{1}{2}\mu_\mathbf{T}^{\mathrm{T}}\Sigma_\mathbf{T}^{-1}\mu_\mathbf{T}\Big) = h(\mathbf{T})\exp\big(\theta^{\mathrm{T}}Z(\mathbf{T}) - A(\theta)\big) \qquad \text{(A.6)}
\end{aligned}$$

where, $\theta = \Sigma_\mathbf{T}^{-1/2}\mu_\mathbf{T}$, $Z = \Sigma_\mathbf{T}^{-1/2}\mathbf{T}$ and $A(\theta) = \frac{1}{2}\mu_\mathbf{T}^{\mathrm{T}}\Sigma_\mathbf{T}^{-1}\mu_\mathbf{T}$. Before we compute the gradient we observe the following regarding the log-partition function A,

$$P(\mathbf{T};\mu_\mathbf{T},\,\Sigma_\mathbf{T}) = h(\mathbf{T})\exp\big(\theta^{\mathrm{T}}\mathbf{Z}(\mathbf{T})\big)\cdot\exp\big(-A(\theta)\big) = \frac{h(\mathbf{T})\exp\big(\theta^{\mathrm{T}}\mathbf{Z}(\mathbf{T})\big)}{\exp\big(A(\theta)\big)}$$

For this to be a valid probability distribution, we must have,

$$\begin{aligned}
&\exp\big(A(\theta)\big) = \int h(\mathbf{T})\exp\big(\theta^{\mathrm{T}}\mathbf{Z}(\mathbf{T})\big)\,\mathrm{d}\mathbf{T} \\
&\implies A(\theta) = \ln\int h(\mathbf{T})\exp\big(\theta^{\mathrm{T}}\mathbf{Z}(\mathbf{T})\big)\,\mathrm{d}\mathbf{T} \quad \text{(hence the name \textit{log-partition} function)}
\end{aligned}$$

We can verify that the expression for $A(\theta)$ obtained above for the Gaussian distribution agrees with this definition of the log-partition function.

$$\begin{aligned}
A(\theta) &= \ln \int h(\mathbf{T}) \exp(\theta^{\mathrm{T}} Z)\, \mathrm{d}\mathbf{T} \\
&= \ln \int \frac{1}{\sqrt{(2\pi)^n |\Sigma_{\mathbf{T}}|}} \exp\left(-\frac{1}{2}\mathbf{T}^{\mathrm{T}}\Sigma_{\mathbf{T}}^{-1}\mathbf{T}\right) \exp\left(\mathbf{T}^{\mathrm{T}}\Sigma_{\mathbf{T}}^{-1}\mu_{\mathbf{T}}\right) \mathrm{d}\mathbf{T} \\
&= \ln \int \underbrace{\frac{1}{\sqrt{(2\pi)^n |\Sigma_{\mathbf{T}}|}} \exp\left(-\frac{1}{2}\mathbf{T}^{\mathrm{T}}\Sigma_{\mathbf{T}}^{-1}\mathbf{T} + \mathbf{T}^{\mathrm{T}}\Sigma_{\mathbf{T}}^{-1}\mu_{\mathbf{T}} - \frac{1}{2}\mu_{\mathbf{T}}^{\mathrm{T}}\Sigma_{\mathbf{T}}^{-1}\mu_{\mathbf{T}}\right)}_{\rho(\mathbf{T};\theta)} \exp\left(\frac{1}{2}\mu_{\mathbf{T}}^{\mathrm{T}}\Sigma_{\mathbf{T}}^{-1}\mu_{\mathbf{T}}\right) \mathrm{d}\mathbf{T} \\
&= \ln \int \rho(\mathbf{T};\theta) \exp\left(\frac{1}{2}\mu_{\mathbf{T}}^{\mathrm{T}}\Sigma_{\mathbf{T}}^{-1}\mu_{\mathbf{T}}\right) \mathrm{d}\mathbf{T} = \ln \exp\left(\frac{1}{2}\mu_{\mathbf{T}}^{\mathrm{T}}\Sigma_{\mathbf{T}}^{-1}\mu_{\mathbf{T}}\right) \underbrace{\int \rho(\mathbf{T};\theta)\, \mathrm{d}\mathbf{T}}_{=1} \\
&= \frac{1}{2}\mu_{\mathbf{T}}^{\mathrm{T}}\Sigma_{\mathbf{T}}^{-1}\mu_{\mathbf{T}}
\end{aligned}$$

Now it is common practice to simply optimize the mean $\mu_{\mathbf{T}}$ alone and update the variance using an empirical estimate, which is what we adopt in our algorithm as well. In that case, we are interested in the gradient with respect to $\mu_{\mathbf{T}}$ alone. Returning back to the derivation of the gradient update and considering the $\rho(\mathbf{T};\theta)$ to be the Gaussian distribution, we have the following derivation for the gradient,

$$\begin{aligned}
\nabla_\theta \ln \rho(\mathbf{T};\theta) &= \nabla_\theta (\ln h(\mathbf{T}) + \theta^{\mathrm{T}} Z - A(\theta)) \\
&= Z - \nabla_\theta A(\theta) \\
&= \Sigma_{\mathbf{T}}^{-1/2}\mathbf{T} - \frac{1}{2}\nabla_\theta\left\{(\Sigma_{\mathbf{T}}^{-1/2}\mu_{\mathbf{T}})^{\mathrm{T}}(\Sigma_{\mathbf{T}}^{-1/2}\mu_{\mathbf{T}})\right\} \\
&= \Sigma_{\mathbf{T}}^{-1/2}\mathbf{T} - \Sigma_{\mathbf{T}}^{-1/2}\mu_{\mathbf{T}} \quad \text{(because } \theta = \Sigma_{\mathbf{T}}^{-1/2}\mu_{\mathbf{T}}\text{)} \\
&= \Sigma_{\mathbf{T}}^{-1/2}(\mathbf{T} - \mu_{\mathbf{T}})
\end{aligned}$$

Substituting this back into the expression for the gradient of the objective we get the following gradient ascent update for the parameter θ,

$$\theta^{k+1} = \theta^k + \alpha \frac{\mathbb{E}\left[S(\mathcal{H}(\mathbf{T}))\left(\Sigma_{\mathbf{T}}^{-1/2}(\mathbf{T} - \mu_{\mathbf{T}})\right)\right]}{\mathbb{E}\left[S(\mathcal{H}(\mathbf{T}))\right]}$$

Using $\theta = \Sigma_{\mathbf{T}}^{-1/2}\mu_{\mathbf{T}}$, we have, $\Sigma_{\mathbf{T}}^{-1/2}\mu_{\mathbf{T}}^{k+1} = \Sigma_{\mathbf{T}}^{-1/2}\mu_{\mathbf{T}}^{k} + \alpha\Sigma_{\mathbf{T}}^{-1/2}\dfrac{\mathbb{E}\left[S(\mathcal{H}(\mathbf{T}))(\mathbf{T} - \mu_{\mathbf{T}})\right]}{\mathbb{E}\left[S(\mathcal{H}(\mathbf{T}))\right]}$

Therefore, $\mu_{\mathbf{T}}^{k+1} = \mu_{\mathbf{T}}^{k} + \alpha\dfrac{\mathbb{E}\left[S(\mathcal{H}(\mathbf{T}))(\mathbf{T} - \mu_{\mathbf{T}})\right]}{\mathbb{E}\left[S(\mathcal{H}(\mathbf{T}))\right]}$ (A.7)