POLITECNICO
MILANO 1863

RE.PUBLIC@POLIMI
Research Publications at Politecnico di Milano

Post-Print

This is the accepted version of:

R. Furfaro, A. Scorsoglio, R. Linares, M. Massari
Adaptive Generalized ZEM-ZEV Feedback Guidance for Planetary Landing via a Deep Reinforcement Learning Approach
Acta Astronautica, Vol. 171, 2020, p. 156-171
doi:10.1016/j.actaastro.2020.02.051

The final publication is available at https://doi.org/10.1016/j.actaastro.2020.02.051

Access to the published version may require subscription.

When citing this work, cite the original published paper.

© 2020. This manuscript version is made available under the CC-BY-NC-ND 4.0 license
http://creativecommons.org/licenses/by-nc-nd/4.0/

Permanent link to this version
http://hdl.handle.net/11311/1133506

Adaptive Generalized ZEM-ZEV Feedback Guidance for Planetary Landing via a Deep Reinforcement Learning Approach

Roberto Furfaro[1,*]

Department of Systems & Industrial Engineering, Department of Aerospace and Mechanical Engineering, University of Arizona, Tucson, AZ 85721, USA

Andrea Scorsoglio[2]

Department of Systems & Industrial Engineering, University of Arizona, Tucson, AZ 85721, USA

Richard Linares[3]

Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, MA, 02139, USA

Mauro Massari[4]

Department of Aerospace Science and Technology, Politecnico di Milano, Milan, 20156, ITA

Abstract

Precision landing on large and small planetary bodies is a technology of utmost importance for future human and robotic exploration of the solar system. In this context, the Zero-Effort-Miss/Zero-Effort-Velocity (ZEM/ZEV) feedback guidance algorithm has been studied extensively and is still a field of active research. The algorithm, although powerful in terms of accuracy and ease of

*Corresponding author: Roberto Furfaro

Email addresses: robertof@email.arizona.edu (Roberto Furfaro), andreascorsoglio@email.arizona.edu (Andrea Scorsoglio), linaresr@mit.edu (Richard Linares), mauro.massari@polimi.it (Mauro Massari)

[1]Professor, Department of System & Industrial Engineering, Department of Aerospace and Mechanical Engineering, University of Arizona, Tucson, AZ 85721, USA

[2]PhD student, Department of System & Industrial Engineering, University of Arizona, Tucson, AZ 85721, USA

[3]Charles Stark Draper Assistant Professor, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, MA, 02139, USA

[4]Associate Professor, Department of Aerospace Science and Technology, Politecnico di Milano, Milan, 20156, ITA

implementation, has some limitations. Therefore with this paper we present an adaptive guidance algorithm based on classical ZEM/ZEV in which machine learning is used to overcome its limitations and create a closed loop guidance algorithm that is sufficiently lightweight to be implemented on board spacecraft and flexible enough to be able to adapt to the given constraint scenario. The adopted methodology is an actor-critic reinforcement learning algorithm that learns the parameters of the above-mentioned guidance architecture according to the given problem constraints.

Keywords: Optimal Landing Guidance, Deep Reinfocement Learning, Closed-loop Guidance

2019 MSC: 00-01, 99-00

---

1. Introduction

Precision landing on large and small planetary bodies is a technology of utmost importance for future human and robotic exploration of the solar system. Over the past two decades, landing systems for robotic Mars missions have been developed and successfully deployed robotic assets on the Martian surface (e.g. rovers, landers)[1, 2]. Considering the strong interest in sending humans to Mars within the next few decades, as well as the renewed interest in building infrastructure in the Earth-Moon system for easy access to the Lunar surface [3], the landing system technology will need to progress to satisfy the demand for more stringent requirements. The latter will call for guidance systems capable of delivering landers and/or rovers to the selected planetary surface with higher degree of precision. In the case of robotic Mars landing, the 3-sigma ellipse, which describes the landing accuracy, has seen a dramatic improvement from the 100 km[1] required by the Phoenix mission to 5 km featured by the newly developed "Sky Crane" system which delivered the Mars Science Laboratory (MSL) to the martian surface in 2012[4]. Although such improvements were needed to deliver a better science through robotic devices, future missions may require delivering cargo (including humans) in specified

location with pinpoint accuracy (somewhere between 10 and 100 meters). Importantly, delivering scientific packages in geologically interesting locations may require guidance systems that are fuel-optimal while satisfying stringent flight constraints (e.g. do not crash on the surfaces with elevated slope).

One of the most important enabling technology for planetary landing is the powered descent guidance algorithm. Generally, powered descent indicates a phase in the landing concept of operation where rockets provide the necessary thrust to steer the spacecraft trajectory toward the desired location on the planetary surface. The corresponding guidance algorithm must determine in real-time both thrust magnitude, directions and time of flight. The original Apollo guidance algorithm, which was used to drive the Lunar Exploration Module (LEM) to the lunar surface, was based on an iterative approach that computed on the ground a flyable reference trajectory in the form of a quartic polynomial[5]. The real-time guidance algorithm generated an acceleration command that targets the final condition of the trajectory. A variation of the Apollo guidance was also employed for the MSL powered descent phase [6]. Over the past two decades, there has been a tremendous interest in developing new classes of guidance algorithms for powered descent that improve performance over the classical Apollo algorithm both in precision and fuel-efficiency. Trajectory optimization methods are currently playing a major role in generating feasible, fuel-efficient trajectories that can be potentially computed in real-time. Much effort has been placed in transforming a fuel-optimal constrained landing problem in a convex optimization problem that can guarantee finding the global optimal solution in a polynomial time [7, 8]. Such approach yielded the G-FOLD algorithm [10] which has been recently tested in real landing systems. Importantly, the convexification methodology has been recently applied to other aerospace guidance problems. A review of the application areas can be found in [11]. Conversely, another class of popular methods generally employed to solve optimal guidance problems, rely on the application of Pontryagin Minimum Principle (PMP). Named indirect methods, such algorithms solve the Two-Point Boundary Value Problems (TPBVP) arising from the necessary conditions for optimality. Re-

cently, a three-dimensional, fuel-optimal, powered descent guidance algorithm based on indirect methods has been developed [12]. The approach, generally named Universal Powered Guidance (UPG), relies on a general powered descent methodology which has been developed and applied to ascent and orbital transfer problems by Ping Lu over the past decade [14, 15, 16]. The algorithm is capable of delivering both human and robotic device on planetary surfaces efficiently and accurately[13]. UPG provides a robust approach based on indirect methods to 1) analyze the thrust profile structure (i.e. analyze the bang-bang profile)and 2) find the optimal numbers of burn times. Importantly, the advantage over G-FOLD is due to its simplicity and flexibility as it does not require customization of the algorithm[12]. However, UPG has the disadvantage that both inequality and thrust direction constraints are generally difficult to enforce[12].

Besides the above mentioned methods, over the past few years, researchers have been exploring the performances of the generalized Zero-Effort-Miss/Zero-Effort-Velocity (ZEM/ZEV) feedback guidance algorithm [23, 24] in the context of landing on large and small bodies of the solar system. The feedback ZEM/ZEV guidance law is analytical in nature and derived by a straightforward application of the optimal control theory to the power descent landing problem. The algorithm generates a closed-loop acceleration command that minimizes the overall system energy (i.e. the integral of the square of the acceleration norm). The ZEM/ZEV feedback guidance is attractive because of its analytical simplicity and accuracy: guidance mechanization is straightforward and it can theoretically drive the spacecraft to a target location on the planetary surface both autonomously and with minimal guidance errors. Moreover, it has been shown to be globally finite time stable and robust to uncertainties in the model if a proper sliding parameter is added (Optimal Sliding Guidance)[26]. Although attractive because of its simplicity and analytical structure, the algorithm is not generally capable of enforcing either thrust constraints and/or flight constraints. There have been attempts to incorporate constraints in the classical ZEM/ZEV algorithm with the utilization of intermediate waypoints

[17, 25]. Although they report good performances, they lack of flexibility and ability to adapt in real-time.

In this paper, we propose a ZEM/ZEV-based guidance algorithm for powered descent landing that can adaptively change both guidance gains and time-to-go to generate a class of closed-loop trajectories that 1) are quasi-optimal (w.r.t. the fuel-efficiency) and 2) satisfy flight constraints (e.g. thrust constraints, glide slope). The proposed algorithm exploit recent advancements in deep reinforcement learning (e.g. deterministic policy gradient [29]), and machine learning (e.g. Extreme Learning Machines, ELM [27, 28]). The overall structure of the guidance algorithm is unchanged with respect to the classical ZEM/ZEV, but the optimal guidance gains are determined at each time step as function of the state via a parametrized learned policy. This is achieved using a deep reinforcement learning method based on an actor-critic algorithm that learns the optimal policy parameters minimizing a specific cost function. The policy is stochastic, but only its mean, expressed as a linear combination of radial basis functions, is updated by stochastic gradient descent. The variance of the policy is kept constant and is used to ensure exploration of the state space. The critic is an Extreme Learning Machine (ELM) that approximates the value function. The approximated value function is then used by the actor to update the policy. The power of the method resides in its capability, if an adequate cost function is introduced, of satisfying virtually any constraint and in its model-free nature that, given an accurate enough dynamics simulator for the generation of sample trajectories, allows learning of the guidance law in any environment, regardless of its properties. This greatly expands the capabilities of classical ZEM/ZEV guidance, allowing for its use in a wide variety of environment and constraint combinations, giving results that are generally close to the constrained fuel optimal off-line solution. Additionally, because the guidance structure is left virtually unchanged, we are able to ensure that the adaptive algorithm is maintained globally stable regardeless of the gain adaptation.

The paper is organized as follows. In section 2 the landing problem set-up is described. In section 3, the theoretical background is provided, including

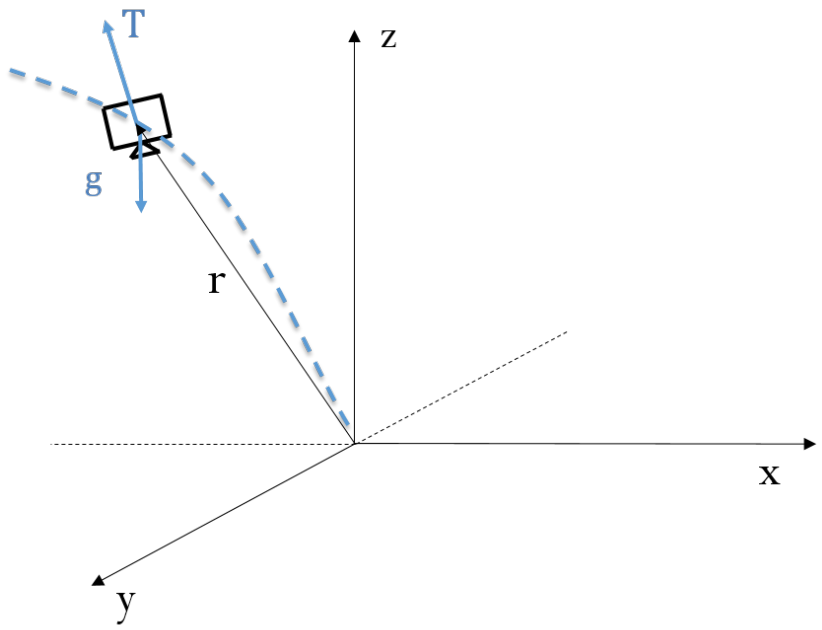


Figure 1: Problem setup

derivation of the classical ZEM/ZEV algorithm, deep Actor-Critic method and ELM. In section 4, the proposed adaptive ZEM/ZEV algorithm is described. In section 5, numerical results for Mars landing scenarios are reported. In section 6, a stability analysis is conducted to show the global stability properties of the proposed algorithm. Conclusions are reported in section 7.

2. Problem setup

The algorithm is being developed for a Mars soft landing scenario described in Figure 1. The problem is described in a orthonormal reference system centered on the nominal landing target on the ground. The equations of motion governing the dynamics of the problem expressed in the above mentioned refer-

ence frame are:

$$\ddot{\mathbf{x}} = \mathbf{g}(\mathbf{r}) + \frac{\mathbf{T}}{m} \tag{1}$$

$$\dot{m} = -\frac{\|\mathbf{T}\|}{I_{sp} g_0} \tag{2}$$

where $\mathbf{x} = [\mathbf{r}, \mathbf{v}]^T$ is the state, $\mathbf{g}(\mathbf{r})$ is the gravity vector at position $\mathbf{r}$ and $\mathbf{T}$ is the thrust vector:

$$\mathbf{T} = \Big[T_x \, , \, T_y \, , \, T_z \Big]^T \tag{3}$$

It should be noted that the control policy is based on Zero-Effort-Miss/Zero-Effort-Velocity guidance which outputs an acceleration command rather than a thrust command. The thrust $\mathbf{T}$ is recovered indirectly knowing engine specifications and mass. The gravitational acceleration is aligned with the vertical direction at all times. The rotation of the planet is neglected as we consider only the terminal guidance of the power descent phase for a pinpoint landing problem where the altitude is small with respect to the radius of the planet. The interaction with the thin martian atmosphere is also neglected. The spacecraft is constrained to remain above the ground which has a constant slope angle of 4°with respect to the horizontal direction except for 5 meter radius flat area around the target.

3. Theoretical Background

3.1. Classical Zero-Effort-Miss/Zero-Effort-Velocity algorithm

Consider the problem of landing on a large planetary body of interest with mission from time t_0 to t_f. The unconstrained, energy-optimal guidance problem can be formulated as follows: Find the overall acceleration $\mathbf{a}$ that minimizes the performance index:

$$J = \frac{1}{2} \int_{t_0}^{t_f} \mathbf{a^T a} \quad \mathrm{dt} \tag{4}$$

for a spacecraft subjected to the following general dynamic equations, valid in any case, even for non-inertial systems:

$$\begin{aligned} \dot{\mathbf{r}} &= \mathbf{v} \\ \dot{\mathbf{v}} &= \mathbf{a} + \mathbf{g} \\ \mathbf{a} &= \mathbf{T}/m \end{aligned} \tag{5}$$

with $\mathbf{r}$, $\mathbf{v}$, $\mathbf{T}$ and $\mathbf{a}$ being position, velocity, thrust and acceleration command vectors respectively and $\mathbf{g}$ is the gravitational acceleration. In the remainder of the paper, $\mathbf{g}$ is assumed to be constant. The latter works well for modeling the powered descent guidance starting close to the planetary surface of a large body. Additionally, note that additional forces (e.g. aerodynamics forces experienced by bodies close to the Mars surface) are considered negligible. The following boundary conditions are given:

$$\mathbf{r}(t_0) = \mathbf{r_0}, \qquad \mathbf{r}(t_f) = \mathbf{r_f} \tag{6}$$

$$\mathbf{v}(t_0) = \mathbf{v_0}, \qquad \mathbf{v}(t_f) = \mathbf{v_f} \tag{7}$$

Importantly, no constraints on acceleration and on the spacecraft state are assumed. The necessary conditions can be derived by a straightforward application of the PMP. Indeed, the Hamiltonian function for this problem is then defined as

$$\mathbf{H} = \frac{1}{2}\mathbf{a^T a} + \mathbf{p_r}^T\mathbf{v} + \mathbf{p_v}^T(\mathbf{g} + \mathbf{a}) \tag{8}$$

where $\mathbf{p_r}$ and $\mathbf{p_v}$ are the costate vectors associated with position and velocity vector respectively. The time-to-go is defined as: $t_{go} = t_f - t$. The optimal acceleration at any time t, can be found by directly applying the optimality condition as

$$\mathbf{a} = -t_{go}\mathbf{p_r}(t_f) - \mathbf{p_v}(t_f) \tag{9}$$

By substituting equation 9 into the dynamics equations to solve for $\mathbf{p_r}(t_f)$ and $\mathbf{p_v}(t_f)$, the optimal control solution with specified $\mathbf{r}_f$ and $\mathbf{v}_f$ and t_{go} is obtained

as:

$$\mathbf{a} = \frac{6[\mathbf{r_f} - (\mathbf{r} + t_{go}\mathbf{v})]}{t_{go}^2} - \frac{2(\mathbf{v_f} - \mathbf{v})}{t_{go}} + \frac{6\int_t^{t_f}(\tau - t)\mathbf{g}d\tau}{t_{go}^2} - \frac{4\int_t^{t_f}\mathbf{g}d\tau}{t_{go}} \tag{10}$$

The Zero-Effort-Miss (ZEM) and the Zero-Effort-Velocity (ZEV) are defined, respectively, as the distance between the desired final position and velocity and the projected final position and velocity if no additional control is commanded from time t onward. Consequently, ZEM and ZEV have the following expressions:

$$\begin{aligned} \mathbf{ZEM} &= \mathbf{r}_f - \left[\mathbf{r} + t_{go}\mathbf{v} + \int_t^{t_f}(t_f - \tau)\mathbf{g}(\tau)d\tau\right] \\ \mathbf{ZEV} &= \mathbf{v}_f - \left[\mathbf{v} + \int_t^{t_f}\mathbf{g}(\tau)d\tau\right] \end{aligned} \tag{11}$$

Then the optimal control law 10 can be expressed as:

$$\mathbf{a} = \frac{6}{t_{go}^2}\mathbf{ZEM} - \frac{2}{t_{go}}\mathbf{ZEV} \tag{12}$$

Note that the solution holds also in the case where $\mathbf{g} = \mathbf{g}(t)$. In any other case in which $\mathbf{g}$ is neither constant nor time dependant, the control law is still usable but it will not be necessarily optimal. In case the equations of motion are non-linear and in general when 11 do not apply, ZEM and ZEV are expressed in a slightly different way. The projected position and velocity cannot be recovered analytically: they must be obtained through an integration of the equations of motion from the current time instant to the end of the mission with control actions set to zero.

$$\begin{aligned} \mathbf{ZEM} &= \mathbf{r}_f - \mathbf{r}_{nc} \\ \mathbf{ZEV} &= \mathbf{v}_f - \mathbf{v}_{nc} \end{aligned} \tag{13}$$

where $\mathbf{r}_{nc}$ and $\mathbf{v}_{nc}$ are, respectively, the position and velocity at the end of mission if *no control action* is given from the considered time onward. It should be noted that using the formulation in 12, which will be called classical ZEM/ZEV from now on, can result in valid trajectories even for cases when the generalized acceleration term is arbitrary. In these types of environment however, using a definition of ZEM and ZEV as in 13, the control gains that solve the

optimal problem are no longer the ones in 12. This leads to the definition of the *Generalized-ZEM/ZEV* algorithm [23], which is valid in any environment and will be used as starting point for the development of the proposed adaptive algorithm:

$$\mathbf{a} = \frac{K_R}{t_{go}^2}\mathbf{ZEM} + \frac{K_V}{t_{go}}\mathbf{ZEV} \tag{14}$$

3.2. Reinforcement learning

Reinforcement Learning (RL) can be conceived as the formalization of learning by trial and error: it is based on the idea that a machine can autonomously learn the optimal behavior, or policy, to carry out a particular task, given the environment, by maximizing (or minimizing) a cumulative reward (or cost). RL algorithms work on systems that are formalized as *Markov Decision Processes* [30, 31, 29].

3.2.1. Markov decision processes

The reinforcement learning problem is generally modeled as a *Markov Decision Process* (MDP) which is composed by: a state space X, an action space U, an initial state distribution with density $p_1(x_1)$ representing the initial state of the system, a transition dynamics distribution with conditional density

$$p(x_{t+1}|x_t, u_t) = \int_{x_{t+1}} f(x_t, u_t, x')dx' \tag{15}$$

representing the dynamic relationship between a state and the next, given action u and a *reward* function r: $S \times U \to R$ that depends in general on the previous state, the current state and the action taken. It should be noted that if the dynamics of the system is considered completely deterministic, this probability is always 0 except when action u_t brings the state from x_t to x_{t+1}. The reward function r is assumed to be bounded. A *policy* is used to select actions by the agent given a certain state. The policy is stochastic and denoted by $\pi_\theta : X \to P(U)$ where $P(U)$ is the set of probability measures of U, $\theta \in \mathbb{R}^n$ is a vector of n parameters and $\pi_\theta(u_t|x_t)$ is the probability of selecting action u_t given state x_t. The agent uses the policy to interact with the MDP and generate

a trajectory made of a sequence of states, actions and rewards. The return

$$r_t^\gamma = \sum_{k=t}^{\infty} \gamma^{k-t} r(x_k, u_k) \tag{16}$$

is the discounted reward along the trajectory from time step t onward, with $0 < \gamma \leq 1$. The agent's goal is to obtain a policy that maximizes the discounted cumulative reward from the start state to the end state, denoted by the performance objective $J(\pi) = \mathbb{E}[r_1^\gamma | \pi]$. By denoting the density at state x' after transitioning for t time steps from state x by $p(x \to x', t, \pi)$ and the discounted state distribution by

$$\rho^\pi(x') := \int_X \sum_{t=1}^{\infty} \gamma^{t-1} P_1(x) p(x \to x', t, \pi) dx \tag{17}$$

The performance objective can then be written as an expectation:

$$J(\pi_\theta) = \int_X \rho^\pi(x) \int_U \pi_\theta(x, u) r(x, u) du dx = \mathbb{E}_{x \sim \rho^\pi, u \sim \pi_\theta} [r(x, u)] \tag{18}$$

where $\mathbb{E}_{x \sim \rho^\pi}$ denotes the expected value with respect to discounted state distribution $\rho(x)$.

During training, the agent will have to estimate the reward-to-go function J for a given policy π: this procedure is called *policy evaluation*. The resulting estimate of J is called value function. The latter may depend either on the state or both on state and action, yielding two different possible definitions. The state value function

$$V^\pi(x) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{k+1} | x_0 = x, \pi\right] \tag{19}$$

only depends on the state x. The state-action value function

$$Q^\pi(x, u) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{k+1} | x_0 = x, u_0 = u, \pi\right] \tag{20}$$

depends on the state x but also on the action u. The relationship between the two is:

$$V^\pi(x) = \mathbb{E}\left[Q^\pi(x, u) | u \sim \pi(x, \cdot)\right] \tag{21}$$

The above mentioned V and Q in recursive form become:

$$V^\pi(x) = \mathbb{E}\left[r(x, u, x') + \gamma V^\pi(x')\right] \tag{22}$$

and

$$Q^{\pi}(x,u) = \mathbb{E}\left[r(x,u,x') + \gamma Q^{\pi}(x',u')\right] \tag{23}$$

which are called *Bellman Equations*. Optimality for both V^{π} and Q^{π} is governed by the *Bellman optimality equation*. Let $V^{*}(x)$ and $Q^{*}(x,u)$ be the optimal value and action-value functions respectively, the corresponding Bellman optimality equations are:

$$\begin{aligned} V^{*}(x) &= \max_{u} \mathbb{E}\left[r(x,u,x') + \gamma V^{*}(x')\right] \\ Q^{*}(x,u) &= \mathbb{E}\left[r(x,u,x') + \gamma \max_{u'} Q^{*}(x',u')\right] \end{aligned} \tag{24}$$

The goal of reinforcement learning is to find the policy π that maximizes V^{π}, Q^{π} or $J(\pi_{\theta})$, or in other words, find V^{*} or Q^{*} that satisfy the Bellman optimality equation.

3.2.2. Stochastic policy gradient theorem

Policy gradient algorithms are among the most popular classes of continuous action and state space reinforcement learning algorithms. The fundamental idea on which they are based on is to adjust the parameters θ of the policy π_{θ} in the direction of the performance objective gradient $\nabla_{\theta} J(\pi_{\theta})$. The biggest challenge is to compute effectively the gradient $\nabla_{\theta} J(\pi_{\theta})$ so that at each iteration the policy becomes better than the one at the previous iteration. It turns out, from the work by Williams [32] who theorized the REINFORCE algorithms, that the gradient of the performance objective can be *estimated* using samples from experience, so without actually computing it and without a complete knowledge of the environment (sometimes referred to as *model-free* algorithms). A direct implication of [32] is the *policy gradient theorem*:

$$\begin{aligned} \nabla_{\theta} J(\pi_{\theta}) = \int_{X} \rho^{\pi}(x) \int_{U} \nabla_{\theta} \pi_{\theta}(u|x) Q^{\pi}(x,u) du dx = \\ = \mathbb{E}_{x\sim\rho^{\pi}, u\sim\pi_{\theta}}\left[\nabla_{\theta} \log \pi_{\theta}(u|x) Q^{\pi}(x,u)\right] \end{aligned} \tag{25}$$

where $Q(x,u)$ is the state-action value function expressing the expected total discounted reward being in state x taking action u. The theorem is important

because it reduces the computation of the performance gradient, which could be hard to compute analytically, to an expectation that can be estimated using a sample-based approach. It is important to note that this estimate is demonstrated to be unbiased so it assures that a policy is at least as good as the one in the previous iteration. Once $\nabla_\theta J(\pi_\theta)$ is computed, the policy update is simply done in the direction of the gradient

$$\theta_{k+1} = \theta_k + \alpha_k \nabla_\theta J_k \tag{26}$$

where α is the learning rate and is supposed to be bounded.

One important issue to be addressed is how to estimate the Q function effectively; all of the above is in fact valid in the case Q represent the true action-value function. In case of continuous action and states spaces, obtaining an unbiased estimate of this is difficult. One of the simplest approach is to use the single sample discounted return r_t^γ to estimate Q which is the idea behind the REINFORCE algorithm [32]. This is demonstrated to be unbiased[5], but the variance is high, which leads to slow convergence. One way of estimating the action-value function in a way that reduces the variance while keeping the error contained is the introduction of a critic in the algorithm.

3.2.3. Stochastic actor-critic algorithm

The *actor-critic* is a widely used architecture based on policy gradient. It consists of two major components. The actor adjusts the parameters θ of the stochastic policy $\pi_\theta(x)$ by stochastic gradient ascent (or descent). The critic evaluates the goodness of the generated policy by estimating some kind of value function. If a critic is present, instead of the true action-value function $Q^\pi(x, u)$, an estimated action-value function $Q^\omega(x, u)$ is used in equation 25. Provided, in fact, that the estimator is compatible with the policy parametrization, meaning

[5]The discounted return is unbiased because it comes directly from experience and no approximation is introduced.

that

$$\frac{\partial Q^w(x,u)}{\partial w} = \frac{\partial \pi(x,u)}{\partial \theta}\frac{1}{\pi(x,u)} \tag{27}$$

then $Q^w(x,u)$ can be substituted to $Q^\pi(x,u)$ in 25 and the gradient would still assure improvement by moving in that direction. It is important to note that $Q^w(x,u)$ is required to have zero mean for each state: $\sum_u \pi(x,u)Q^w(x,u) = 0, \quad \forall x \in X$. In this sense it is better to think of $Q^w(x,u)$ as an approximation of the *advantage function* $A^\pi(x,u) = Q^\pi(x,u) - V^\pi(x)$ rather than $Q^\pi(x,u)$. This is in fact what will be used in the following.

In general introducing an estimation on the action-value function may introduce bias but the overall variance of the method is decreased which ultimately leads to faster convergence. The critic goal is to estimate the action-value function, providing a better estimate of the expectation of the reward with respect to using the single sample reward-to-go given state x and action u. This happens because the action-value function is estimated from an average over all the samples, not just from the ones belonging to a single trajectory. This will become clearer in section 4 where the details of the algorithm will be discussed. It should be noted that both the critic and the deterministic part of the policy are represented by a Single Layer Feedforward Network (SLFN). Specifically the critic is represented by an Extreme Learning Machine which is a particular instance of them and will be presented in the following section.

3.3. Extreme learning machines

Extreme Learning Machines (ELM) are a particular kind of Single Layer Feedforward Networks (SLFN) with a single layer of hidden neurons which do not make use of back-propagation as learning algorithm. Backpropagation is a multiple step iterative process; ELM instead uses a learning method which allows for learning in a single step. The concepts behind ELM had already been in the scientific community for years before Huang theorized and formally introduced them as Extreme Learning Machines in 2004 [28, 27]. According to their creator, they can produce very good results with a learning time that is a fraction of the time needed for algorithms based on back-propagation.

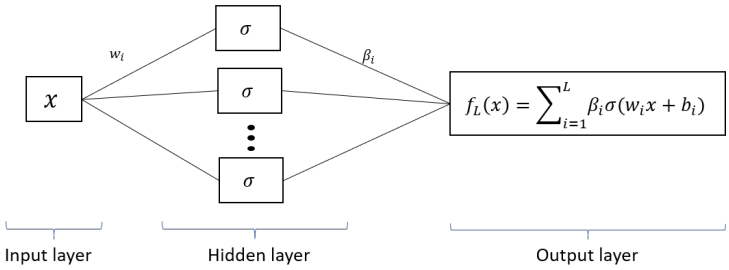


Figure 2: Single layer feedforward network

Consider a simple SLFN, the universal approximation theorem states that any continuous target function $f(x)$ can be approximated by SLFNs with a set of hidden nodes and appropriate parameters. Mathematically speaking, given any small positive ϵ, for SLFNs with enough number of neurons L, it is verified that:

$$\|f_L(x) - f(x)\| < \epsilon \tag{28}$$

where

$$f_L(x) = \sum_{i=1}^{L} \beta_i h_i(x) = \mathbf{H}(x)\beta \tag{29}$$

is the output of the SLFN, β being the output weights matrix and $\mathbf{H}(x) = \sigma(\mathbf{W}x + \mathbf{b})$ the output of the hidden layer for input x, with $\mathbf{W}$ and $\mathbf{b}$ being the input weights and biases vectors respectively, σ is the activation function of the hidden neurons. A representation of an SLFN can be seen in Figure 2. In conventional SLFN, input weights w_i, biases b_i and output weights β_i are learned via backpropagation[6]. ELM are a particular type of SLFN that have the same structure but only β_i are learned, while input weights and biases are assigned randomly at the beginning of training without the knowledge of the training data and are never changed. It is demonstrated that, for any randomly

[6]Backpropagation is an optimization technique based on the concept of updating iteratively weights and biases of a neural network according to the gradient of the loss function to be minimized. It is called backpropagation because the error is calculated at the output and distributed back through the network layers. Details in [34]

generated set $\{\mathbf{W}, \mathbf{b}\}$ of input weights and biases,

$$\lim_{L\to\infty} \|f(x) - f_L(x)\| = 0 \tag{30}$$

holds if the output weights matrix β is chosen so that it minimizes 28, which is equivalent to saying that it minimizes the loss function $\|f(x)-f_L(x)\|$. Equation 28 after some manipulation, becomes

$$\|\mathbf{H}\beta - \mathbf{Y}\| < \epsilon \tag{31}$$

where $\mathbf{Y} = [\mathbf{y_1}, ..., \mathbf{y_N}]^T$ are the target labels and $\mathbf{H} = [\mathbf{h^T}(\mathbf{x_1}), ..., \mathbf{h^T}(\mathbf{x_N})]$ the hidden layer output. Given N training samples $\{x_i, y_i\}_{i=1}^N$, the training problem is reduced to:

$$\mathbf{H}\beta = \mathbf{Y} \tag{32}$$

The output weights are then simply:

$$\beta = \tilde{\mathbf{H}}\mathbf{Y} \tag{33}$$

Where $\tilde{\mathbf{H}}$ is the Moore-Penrose generalized inverse matrix[7] of $\mathbf{H}$. This is demonstrated to minimize the loss 31 given a large enough sets of training points and neurons. This is another way of saying that $\tilde{\mathbf{H}}$ are the weights that represent the minimum norm least square solution of 32. This will be used in the actor-critic algorithm and will be explained in section 4.

4. Adaptive-ZEM/ZEV algorithm

The Adaptive-ZEM/ZEV (A-ZEM/ZEV) is based on the idea of learning the parameters K_R, K_V and the time of flight T_f, which is related to the time-to-go t_{go} of the generalized ZEM/ZEV algorithm in Equation 14. The overall idea is that the guidance gains and the time-to-go can be adapted during the powered descent phase to satisfy specific constraints while maintaining quasi-fuel

[7] Used here because the numbers of neurons and samples are not equal so the system is not squared.

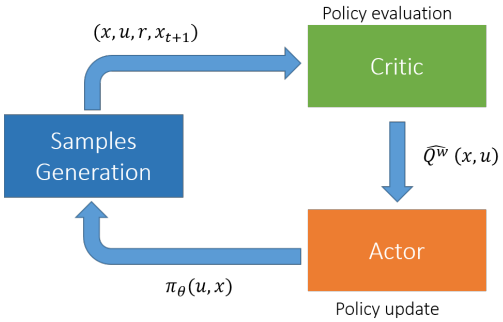


Figure 3: Schematic representation of the actor-critic algorithm

optimality and close-loop characteristics. The guidance adaptation is achieved by using a customization of the actor-critic algorithm described in Section 3.2. More specifically, we have developed a fast RL framework based on a combination of the REINFORCE algorithm [32] and a critic network based on Extreme Learning Machines for estimating the value function. The goal is to show that learning of the adaptive generalized ZEM/ZEV guidance can occur fastly and effciently. The proposed RL-based learning algorithm can be broken down in three major blocks:

1. Samples generation
2. Critic neural network fitting
3. Policy update

A high level schematic representation of the algorithm can be seen in Figure 3. Overall, at each global iteration, a batch of sample trajectories is generated giving a set of states, actions, costs and next states (x, u, c, x_{t+1}). These are then fed to the critic that outputs an approximation of the expected cost-to-go given a particular action and state $\hat{Q}^w(x, u)$ that is then used to update the policy by the actor. Importantly, we do not aim at learning the full guidance policy, which is represented by the generalized ZEM/ZEV algorithm (Eq. 14). Here, we call *policy* specifically K_R, K_V and T_f as function of the lander state and parametrized/approximated by a neural networks (see Figure 4 for a description of the policy network). Indeed, the parameters of such networks are learned

during the training phase. Details of the three phases are reported in the next sections.

4.1. Samples generation

At each global iteration, a batch of trajectories are generated by letting the agent interact with the environment using policy $\pi_\theta(u|x)$, which is a representation of the guidance gains in equation 14, giving a series of samples $(x_{i,t}, u_{i,t}, c_{i,t}, x_{i,t+1})$, where i represents the trajectory number and t is the time-step along that trajectory. At the start of each episode, the time of flight T_f is sampled using the policy and kept constant for the entire episode. The starting position is randomly chosen by sampling a gaussian distribution around the nominal starting position. This ensures exploration of the state space around the nominal starting state and also avoids singularities in the policy evaluation step[8]. The time is discretized in a fixed number of time steps: at the beginning of each time step the policy is sampled and K_R and K_V obtained, the acceleration command calculated with 14, and the equations of motion integrated forward in time. The acceleration command is kept constant during the time interval. The cost, whose value depend on the particular case addressed, is assigned at each time step. It should be noted that here the *reward* in the definitions in Section 3.2 in substituted with a *cost* for reasons that will become clearer later. Importantly, the whole machinery described in Section 3.2 is valid also in case a cost is used to evaluate actions instead of a reward. The final time for each episode is also fixed and the agent runs until the end time is reached unless an impact with the ground is detected in which case the episode ends.

4.1.1. Policy

The policy is described by a gaussian distribution with fixed variance σ^2 and variable mean from which actions are sampled. The mean is parametrized

[8]The critic network works well only if each state is associated with a single value. If each episode starts from the exacts same position, there are equal states associated with different costs, which makes the regression perform poorly.

over a certain weight vector θ which is learned through gradient descent. The stochasticity of the policy is essential for learning because 1) it enables exploration of the action space and 2) the machinery developed for stochastic policy gradient can then be applied. Since the parameters of the guidance algorithm to learn are three, i.e. K_R, K_V and T_f, the policy is subdivided in three separate parts and parametrized with (θ_{K_R}, θ_{K_V} and θ_{T_f}). The policy can be formally expressed as:

$$K_R = \pi_{\theta_{K_R}} = \mathcal{N}(\mu_{K_R}, \sigma^2) \tag{34}$$

$$K_V = \pi_{\theta_{K_V}} = \mathcal{N}(\mu_{K_V}, \sigma^2) \tag{35}$$

$$T_f = \pi_{\theta_{T_f}} = \mathcal{N}(\mu_{T_f}, \sigma^2) \tag{36}$$

where:

$$\mu_{K_R} = \phi(\mathbf{x})^T \theta_{K_R} \tag{37}$$

$$\mu_{K_V} = \phi(\mathbf{x})^T \theta_{K_V} \tag{38}$$

$$\mu_{T_f} = \phi(\mathbf{x})^T \theta_{T_f} \tag{39}$$

$\phi(\mathbf{x})$ is the vector of feature functions evaluated in state $\mathbf{x}$ and θ_{K_R}, θ_{K_V} and θ_{T_f} are the weight vectors associated with each output. Note that the mean values learned during the training phase are employed in the generalized ZEM/ZEV algorithm. The features are comprising two sets of three dimensional radial basis functions (RBF) with centers distributed evenly across the position and velocity spaces. They are represented by the expression:

$$\phi(\mathbf{r}) = e^{-\beta_R \|\mathbf{r}-\mathbf{c_r}\|^2} \tag{40}$$

$$\phi(\mathbf{v}) = e^{-\beta_V \|\mathbf{v}-\mathbf{c_v}\|^2} \tag{41}$$

with β_R and β_V being constant parameters related to the variance of the radial functions which is set accordingly to the particular case, $\mathbf{r}$ and $\mathbf{v}$ being respectively the position and velocity and $\mathbf{c_r}$ and $\mathbf{c_v}$ the centers of the RBFs. The centers are generated by dividing the state space of the problem in a set of intervals, thus creating a grid of equally spaced points in the position and velocity

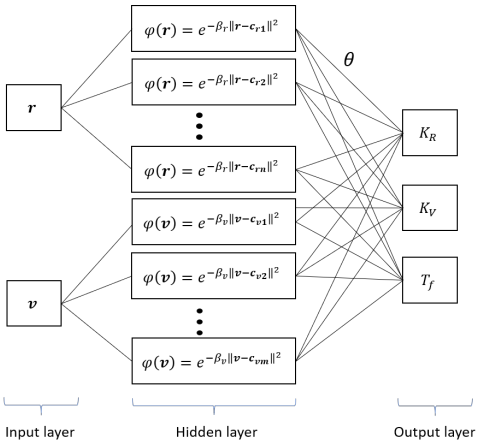


Figure 4: Policy neural network

spaces. The deterministic part of this policy can be seen as a neural network with two three-dimensional inputs (**r**,**v**), a single hidden layer of neurons with radial basis activation functions and a three dimensional output layer (K_R, K_V, T_f). A scheme can be seen in Figure 4. The parameters θ are the weights that multiplied by the features give the output, which is the mean of the stochastic policy as stated above.

4.2. Critic neural network

One key part of the algorithm is the fitting of the neural network that approximates the value function. As explained in 3.2.3, in actor-critic algorithms the expectation in equation 25 is not computed exactly, but it is rather expressed using an approximated value function $Q^w(x,u)$. Here, we employ the advantage function $A^\pi(x,u) = Q^\pi(x,u) - V^\pi(x)$ rather than $Q^\pi(x,u)$. The approximated advantage function can be rewritten, using the definition of Q, as function of V only:

$$Q^w(x,u) = \hat{A}^\pi(u,x) = \hat{Q}^\pi(x,u) - \hat{V}^\pi(x) = r(x,u) + \hat{V}^\pi(x_{t+1}) - \hat{V}^\pi(x) \quad (42)$$

where $\hat{A}^\pi(u,x)$, $\hat{Q}^\pi(u,x)$ and $\hat{V}^\pi(x)$ are the approximated versions of $A^\pi(u,x)$, $Q^\pi(u,x)$ and $V^\pi(x)$. Clearly, in order to compute the approximated advantage

function, only $\hat{V}^{\pi}(x)$ must be obtained. The latter is done by modeling the value function via a single layer forward networks with the following sigmoid activation function

$$\sigma(s_i) = \frac{1}{1+e^{-s_i}} \quad \text{with} \quad s_i = w_i x + b_i \tag{43}$$

The SLFN is used as a function approximator that maps the inputs, in this case the 6D states, into the scalar representing the discounted cost and trained at each step using ELM theories as introduced earlier. The latter is done by generating at each global iteration step, a training set on which the SLFN is trained using the training algorithm described in Section 3.3. There are normally two ways to define this training set referring to two different types of methods:

- Monte Carlo (MC): the value function is approximated at any given state $x_{i,t}$ by the return, which is the discounted cost-to-go $y = \sum_{t'=t}^{T} \gamma^{t'-t} c(x_{i,t'}, u_{i,t'})$. In this case the training set is defined by the couples:

$$\left\{ \left(x_{i,t}, \sum_{t'=t}^{T} \gamma^{t'-t} c(x_{i,t'}, u_{i,t'}) \right) \right\} \tag{44}$$

  This is an unbiased way of expressing the value function but could suffer from high variance.

- Temporal Difference (TD): the value function is approximated by a bootstrapped estimate of the cost-to-go, meaning that the previously fitted value function is used as an estimation of the cost-to-go from time step $t+1$ onward. The training set in this case is given by the couples:

$$\left\{ \left(x_{i,t}, c(x_{i,t}, u_{i,t}) + \hat{V}^{\pi}(x_{t+1}) \right) \right\} \tag{45}$$

  this way of expressing the value function introduces a bias, because the estimation of V is not perfect, but reduces the variance.

In this case the possibility of using the TD errors for value function approximation as in 45 was explored but discarded in favor of the MC version 44. Here, 45

works well only when the bias introduced by the approximation is small. In this case, in two consecutive global iterations the visited states could be very different. Consequently, the neural net approximating the value function and trained on a particular portion of the state space could lead to very big extrapolation errors. For this reason, we decided to employ the MC version of the algorithm to keep the bias contained and find other means of reducing the variance. It should be noted right away that, even if the variance is higher with respect to the bootstrapped version, it is still lower than that of the vanilla REINFORCE algorithm. Indeed, the samples come from all the generated episodes, and therefore the learned value function is an *average* of the expected cost-to-go, which is a better estimate of the value function with respect to the simple sample estimate. Note also that the approximated value function is the discounted cost and not the discounted reward. This is a choice made for this particular case in which the goodness of an action is more clearly represented by a cost instead of a reward.

4.3. Policy update

During the training phase, the policy is optimized using gradient descent instead of gradient ascent. The latter requires gradient estimation to execute the policy update step. Once the value function is approximated by the critic net, it is used to estimate the gradient of the objective function $J(\pi_\theta)$. In stochastic policy gradient, the expectation in equation 25 is not computed directly but is approximated by averaging the gradient over the samples. In this case a batch of trajectories is used to estimate the gradient. The expression of the approximated gradient becomes:

$$\nabla_\theta J(\pi_\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(u_{i,t}|x_{i,t}) \hat{A}^\pi(u_{i,t}, x_{i,t}) \tag{46}$$

where N is the number of sample trajectories in the batch, T is the number of time instants in each trajectory, $\nabla_\theta \log \pi_\theta(u|x)$ is the gradient of the log-probability of the stochastic policy which, for a gaussian policy like 34, is

obtained analytically as:

$$\nabla_\theta \log \pi_\theta = \frac{\pi_\theta - \mu}{\sigma^2}\phi(\mathbf{s}) \tag{47}$$

Here, $\hat{A}^\pi(u_t, x_t)$ is the approximated advantage function described in 4.2 and indicates how much better the action $u_{i,t}$ performs with respect to the average action. Using the advantage function generally reduces the variance (3.2.3) but it relies on an approximation that introduces bias into the process. A way to reduce the effect of bias is to use the advantage function formulated as:

$$\hat{A}^\pi_n(u_{i,t}, x_{i,t}) = \sum_{t'=t}^{T} \gamma^{t'-t} c(x_{i,t'}, u_{i,t'}) - \hat{V}^\pi(x_{i,t}) \tag{48}$$

which is often referred to as the Monte-Carlo formulation of the advantage function, with the discount factor being introduced as $0 < \gamma < 1$. This is unbiased because the real cost to go is used to estimate the action-value function but is low in variance because the average value associated to state $x_{i,t}$ is subtracted.

To implement the gradient descent algorithm, the policy parameters update is simply done by taking a step in the opposite direction of the gradient $\nabla_\theta J(\pi_\theta)$:

$$\theta_{k+1} = \theta_k - \alpha \nabla_\theta J(\pi_\theta) \tag{49}$$

where α is the bounded learning rate. After each update, the algorithm is tested and the cumulative cost is computed

$$C_k = \sum_{t=0}^{T} c(x_t, u_t) \tag{50}$$

where k stands for k-th iteration. The algorithm stops if the average cumulative cost difference among the last 5 iteration is less than a tolerance ϵ or it has reached the maximum number of iterations. A summary of the algorithm in form of pseudo-code is given in Figure 5.

5. Numerical Results

To evaluate the performance of the proposed algorithm, two powered descent examples for Mars pinpoint landing are presented. The spacecraft parameters

for k = 1: nº max iterations
  for i = 1 : nº episodes per batch
    sample policy $\pi \rightarrow T_f$
    for t = 1 : nº time steps per episode - 1
      - Sample policy $\pi \rightarrow K_R, K_V$
      - generate samples $(x_{i,t}, u_{i,t}, c_{i,t}, x_{i,t+1})$
    end **for**
  end **for**
  - fit $\hat{V}^{\pi}(x)$ to sampled cost-to-go $\{(s_{i,t}, \sum_{t'=t}^{T} \gamma^{t'-t} c(x_{i,t'}, u_{i,t}))\}$
  for i = 1 : nº episodes per batch
    for t=1 : nº time steps per episode - 1
      - evaluate $\hat{A}_n^{\pi}(u_{i,t}, x_{i,t}) = \sum_{t'=t}^{T} \gamma^{t'-t} c(x_{i,t'}, u_{i,t'}) - \hat{V}^{\pi}(x_{i,t})$
      - evaluate $\nabla_\theta log \pi_\theta(u_{i,t}|x_{i,t})$
    end **for**
  end **for**
  - evaluate $\nabla_\theta J(\pi_\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta log \pi_\theta(u_{i,t}|x_{i,t}) \hat{A}_n^{\pi}(u_{i,t}, x_{i,t})$
  - update policy $\vartheta_k = \vartheta_{k-1} - \alpha \nabla_\vartheta J(\pi_\vartheta)$
  - test new policy $\pi_k \rightarrow$ obtain cumulative cost $C_k = \sum_{t=0}^{T} c(x_{k,t}, u_{k,t})$
  - calculate average change E in C among last 5 iteration
    if E < ε
      break
end for

Figure 5: Summary of the A-ZEM/ZEV algorithm

for the selected problems are the following:

$$\begin{gathered} \mathbf{g} = [-3.7114\,,\,0\,,\,0]^T m/s^2 \qquad m_{dry} = 1505\,kg \\ m_{wet} = 1905\,kg \qquad I_{sp} = 225\,s \qquad \bar{T} = 3.1\,kN \\ T_{min} = 0.3\bar{T} \qquad T_{min} = 0.8\bar{T} \qquad n = 6 \end{gathered} \tag{51}$$

Here, m_{wet} and m_{dry} are the spacecraft mass both wet and dry, respectively, and $\mathbf{g}$ is the Martian gravity vector, assumed to be constant. Additionally, n is the number of thrusters, each with a full throttle capability of $\bar{T}$, limited to a thrust level between 0.3 and 0.8 at all times. The thrusters are mounted on the spacecraft body with a cant angle ϕ with respect to the net thrust direction $\hat{T}$. If $\bar{\mathbf{T}}$ as the magnitude of instantaneous thrust for each individual thruster, the instantaneous net thrust is:

$$\mathbf{T}_n = n\bar{T}cos(\phi)\hat{\mathbf{T}} \tag{52}$$

The A-ZEM/ZEV algorithm was tested on two cases: a 2D case where the spacecraft is constrained to move on the x-z plane and a full 3D case. The guidance gains and the final time of the adaptive algorithms are learned to ensure safe landing at a selected location of the Martian surface with minimum

fuel. it is assumed that the target point is at the origin of the reference frame fixed with the Martian surface which needs to be achieved with zero velocity. In both cases a glide constraint is introduced:

$$\theta(t) = \arctan(\frac{\sqrt{r_x(t)^2 + r_y(t)^2}}{r_z(t)}) \leq \theta_{lim} \tag{53}$$

with an angle $\theta_{lim} = 4°$ with respect to the horizon. During the descent, the gains adaptation must ensure that this constraint is always satisfied. This is achieved by terminating the episodes whenever the agent violates the constraint, which also leads to an increase in cost. The cost function $c(t)$ that enforces that constraint while searching for fuel optimal solutions is defined as follows:

$$\begin{aligned} C(t) = w_m dm_t + \delta(t - T_f)\left[w_r^f \|r_t - r_f\|^2 + w_v^f \|v_t - v_f\|^2 + b_f\right] \\ + \delta(t - t_i)\left[w_r^i \|r_t - r_f\|^2 + b_i\right] \end{aligned} \tag{54}$$

Where w_m, w_r^f, w_v^f and w_r^i are weights associated with the burned mass, the end position and velocity errors and the impact point position error respectively, t_i and T_f are the time of impact and the final time respectively, and b_f and b_i are biases added at the end of episodes with $b_i > b_f > 0$.

Importantly, $b_i > b_f$ ensures that the collision-less solution has a lower cost than a solution that impacts on the constraint. Conversely, $b_f > 0$ ensures that the value function close to the target does not get too close to 0. The latter may cause problems during training phase because the error introduced by the function approximator might be high relative to the actual value. It is important to note here that the introduction of the positive bias b_i is what ensures that the agent is incentivised to look for a collisionless solution, enforcing the constraint in equation 53.

Setting up the cost is the hardest hustle because the agent can easily fall into a local minimum due to one of the multiple terms in the cost function prevailing over the others. Since the guidance adaptation has to minimize fuel cost without violating the constraints, a careful tuning of the weights values is mandatory. Here, we have decided to add a high bias cost every time an episode ended with an impact. The latter ensures that the minimum cost is always achieved with

Table 1: Initial state distributions

	x (m)	y (m)	z (m)	$\dot{x}$ (m/s)	$\dot{y}$ (m/s)	$\dot{z}$ (m/s)
Nominal 2D	1500	0	1500	100	0	-60
Nominal 3D	-500	-1000	1500	100	-60	-60
Bounds	$\pm$ 500	$\pm$ 500	$\pm$ 0	$\pm$ 5	$\pm$ 5	$\pm$ 5

Table 2: Hyperparameters

w_m	w_r^f	w_v^f	w_r^i	b_i	b_f
0.5	1e-1	1e-1	5e-4	100	10

a collision-less solution. In this fashion, we have observed that the algorithm always tries first to avoid the constraint, then to lower the fuel consumption. The introduction of this constant biases leads to discontinuous jumps in the cost profile as training progresses. The reason is that a trajectory without collisions has a much lower cost than one that collides with the constraint as $b_i > b_f$.

For both 2D and 3D guidance simulations, the starting position for each episode was sampled from a uniform distribution around the nominal starting state. Table 1 shows the initial state distributions for both cases. Table 2 instead shows the values of the hyperparameters used in the definition of the cost function in 54.

Additionally, in the 3D case, at each iteration 25 test trajectories were generated after the policy update step. To evaluate the performances of the current version of the policy, the initial state is selected from the same distribution used for the training samples. The latter slows down the learning process but allows to learn a policy that works for any starting state within the above mentioned distribution. In both 2D and 3D guided descents, the solutions obtained with A-ZEM/ZEV is compared with the classical ZEM/ZEV solution and a fuel optimal solution obtained with GPOPS [36] (***G****eneral* ***P****seudospectral* ***OP****timal Control* ***S****oftware*). The 2D guided descent scenario results are reported in Figure 6.

Clearly, the algorithm manages to find a solution that comply with the glide slope constraint whereas the classical-ZEM/ZEV solution violates it. Here, the ELM-based critic is trained using 80% of the training set defined in 44 and the rest is used for testing. Note that this step is repeated at each time-step with a different training set. The number of neurons of the ELM is set as 1/10 of the number of samples. The latter is demonstrated to work well in all situations, the most challenging ones being iterations where only some of the sample trajectories would hit the constraints and the other would arrive at the target. This condition created a stiff value function with neighbouring states having very different values given by a different future development (one hitting the constraint, the other reaching the target). Overall, A-ZEM/ZEV has shown good performance in terms of constraint avoidance. Fuel consumption is considered in the cost but there is still space for improvement as the GPOPS fuel optimal solution is not reached.

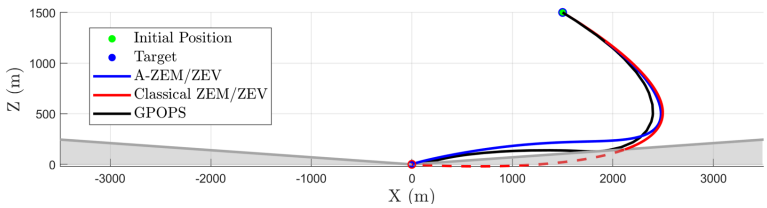
Some details on the training process are shown in Table 3. One can appreciate that the ELM has a very short learning time if compared to the overall iteration time. This ensures that most of the computational time is spent generating the sample trajectories and updating the policy.

Table 3: RL performance

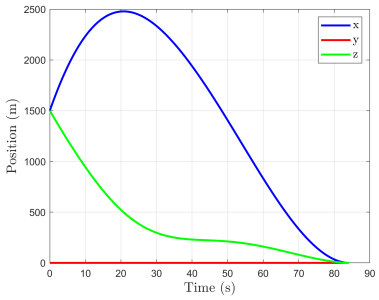
Case	N^o **iterations**	**Total training time (hours)**	**Average iteration time (s)**	**Average critic training time (s)**	**Average critic NRMSE**[9]
2D	503	1.78	12.72	7.849e-2	2.860e-1
3D	804	20.68	92.58	2.727e-1	1.742e-1

The resuls in terms of fuel consumption are shown in Table 4. A-ZEM/ZEV performs less optimally with respect to the optimal GPOPS solution as expected but slightly more optimally than Classical-ZEM/ZEV, while managing

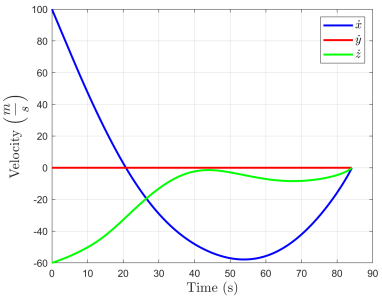
[9]Normalized Root Mean Squared Error



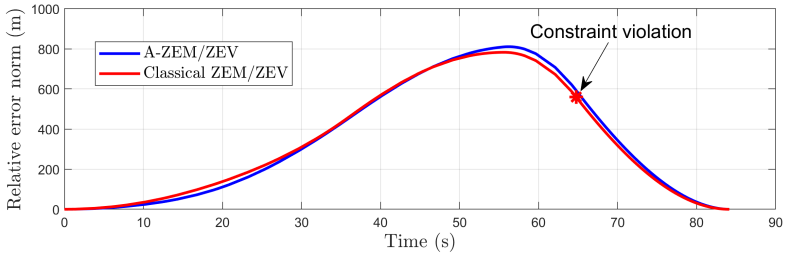
(a) Trajectory



(b) Position

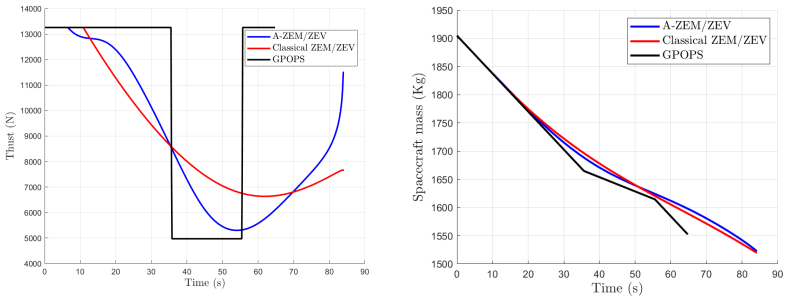


(c) Velocity



(d) Norm of the position error with respect to the GPOPS solution.

Figure 6: $r_0 = [1500, 0, 1500]^T m$, $\dot{r_0} = [100, 0, -60]^T m/s$. Note that although achieving the target state, the classical ZEM/ZEV violate the slope constraints. Conversely, as seen in (a) the Adaptive ZEM/ZEV guidance law does not violate the slope constraints.



(a) Thrust (b) Spacecraft mass

Figure 7: Thrust, mass and guidance gains for 2D case

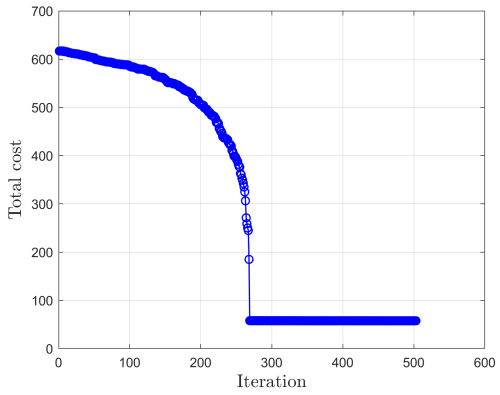
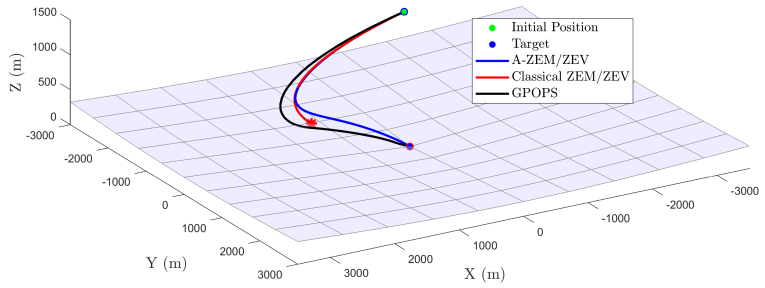
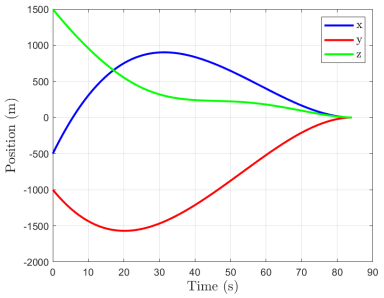


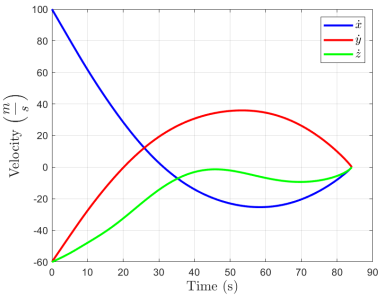
Figure 8: Cost during training



(a) Trajectory - $*$ is the constraint violation location

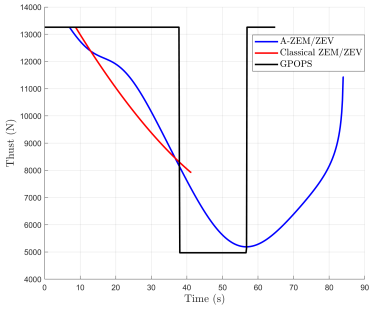


(b) Position

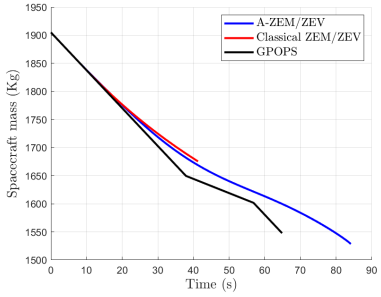


(c) Velocity

Figure 9: $r_0 = [-500, -1000, 1500]^T m$, $\dot{r_0} = [100, -60, -60]^T m/s$



(a) Thrust



(b) Spacecraft mass

Figure 10: Thrust, mass and guidance gains for 3D case

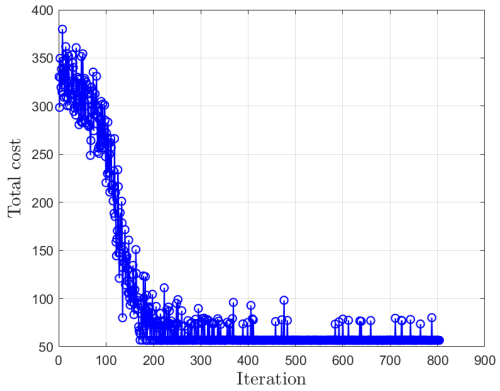
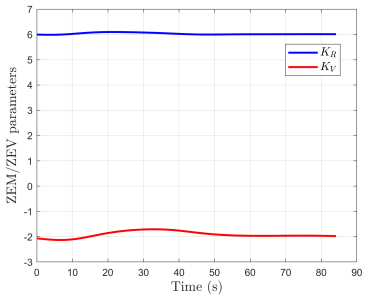
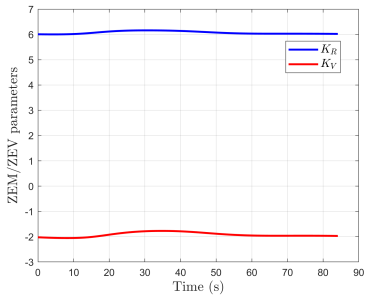


Figure 11: Cost during training



(a) Guidance gains - 2D



(b) Guidance gains - 3D

Table 4: Performance comparison

Test case	**Algorithm**	**Mass depleted (kg)**	**TOF (s)**
2D	A-ZEM/ZEV	382.75	84.1
	Classical-ZEM/ZEV	385.51	84.1
	GPOPS	352.59	64.7
3D	A-ZEM/ZEV	376.54	84.1
	Classical-ZEM/ZEV	378.81	84.1
	GPOPS	357.25	64.8

to avoid collision with the constraint. Classical ZEM/ZEV instead reaches the target but violates the constraint. This shows the major shortcoming of classical ZEM/ZEV: the impossibility to enforce path constraints and why out algorithm overcomes this by making the gains state dependent.

One of the major strengths of the algorithm is the ability to provide a closed-loop guidance control that is both close to optimal and compliant with the constraints. An interesting remark emerges from Figures 12a and 12b. Here the evolution of the control gains K_r and K_v is shown. It possible to see that the learning algorithm adjusts the values of the gains according to the constraint scenario in a way that allows the lander to avoid collisions and get to the target safely. The power of the method resides also in the fact that the underlying structure of the ZEM-ZEV guidance allows the algorithm to achieve pinpoint landing accuracy as shown in the following section. The TOF (T_f) is optimized by the learning algorithm as a function of the initial state, as explained in section 4.1.1, and is not modified during the trajectory. The optimal value can be seen in Table 4. It should be noted that the TOF for the Classical-ZEM-ZEV is the one obtained after the training process so it has the same value as the one for A-ZEM/ZEV. The TOF of the optimal solution is instead optimized with GPOPS itself.

5.1. Monte-Carlo analysis

A Monte-Carlo analysis was carried out on the 3D case. The objective is to prove that the trained agent is able to perform pinpoint landing with a high degree of accuracy both in terms of final position and velocity. In this case, following the procedure described above, the neural network was trained by selecting the initial state of each sample trajectory from a quite large uniform distribution around the nominal start state. In particular the x and y are taken from a distribution with bounds $\pm 500\ m$, the z coordinate is kept at $1500\ m$. The velocity instead has bounds $\pm 5\ m/s$. Figure 12b shows the distribution of the final position on the ground across 1000 trials after training. Figure 12c shows the distribution of final velocity magnitude. The trained policy clearly manages to drive the spacecraft to the target without ever violating the constraint and with a high degree of accuracy in terms of position, as well as keeping the final velocity below a safe 5 cm/s.

6. Stability Analysis

A guidance algorithm should in general be stable so that it can be safely used in practice. In the case of ideal, unperturbed dynamics, it has already been demonstrated that the classical ZEM/ZEV described in Section 3.1 is stable [26]. In this section we study the stability of the A-ZEM/ZEV algorithm.

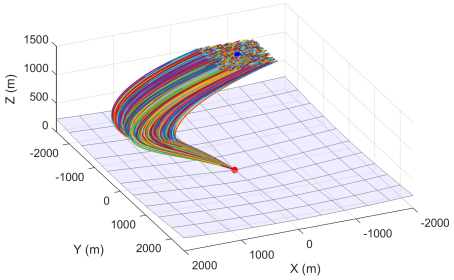
6.1. Closed-loop Dynamics

The formulation of the guidance acceleration as function of ZEM and ZEV results is a linear, non-autonomous, feedback dynamical system. Consequently, classical linear system method of analysis can be employed. The acceleration command for the generalized ZEM/ZEV, as expressed in Section 3.1, is

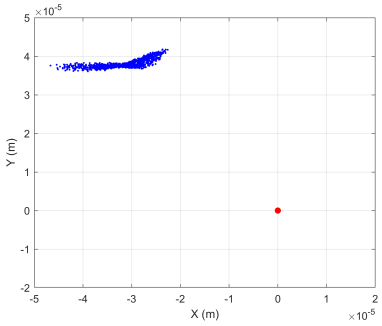
$$\mathbf{a_c} = \frac{K_R}{t_{go}^2}\mathbf{ZEM} + \frac{K_V}{t_{go}}\mathbf{ZEV} \tag{55}$$

considering then that

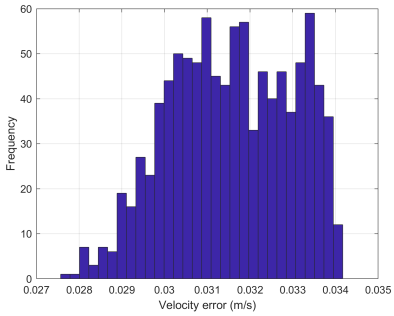
$$\begin{aligned} \dot{\mathbf{ZEM}} &= -\mathbf{a_c} t_{go} \\ \dot{\mathbf{ZEV}} &= -\mathbf{a_c} \end{aligned} \tag{56}$$



(a) Trials



(b) Final position



(c) Final velocity

Figure 12: Monte-Carlo analysis

the guidance system can be expressed as follows:

$$\begin{bmatrix} \mathbf{ZE\dot{M}} \\ \mathbf{Z\dot{E}V} \end{bmatrix} = \begin{bmatrix} -\frac{K_R}{t_{go}} & -K_V \\ -\frac{K_R}{t_{go}^2} & -\frac{K_V}{t_{go}} \end{bmatrix} \begin{bmatrix} \mathbf{ZEM} \\ \mathbf{ZEV} \end{bmatrix} \tag{57}$$

In order to study the stability of the Linear Time-Varying (LTV) system one can utilize known properties of linear systems in order to reduce the system to an equivalent Linear Time-Invariant (LTI) system that is much more convenient for stability analysis.

6.2. Transformation of LTV systems into LTI systems

Following the work by Wu [33], let us consider a linear time-varying system

$$\dot{\mathbf{x}} = A(t)\mathbf{x} \tag{58}$$

Definition 1. A linear time-varying system of the form of Eq.(58) is said to be *invariable* if it can be transformed into a linear time-invariant system of the form $\dot{\mathbf{z}} = F\mathbf{z}$ by some valid transformations (such as the algegraic transformation and the $t \longleftrightarrow \tau$ transformation defined below), where F is a constant matrix and $\mathbf{z}$ may or may not be an explicit function of t.

Definition 2. An *algebraic transformation* is a transformation of states defined by $\mathbf{x}(t) = T(t)\bar{\mathbf{x}}(t)$ where $T(t)$ is a non-singular matrix for all t and $\dot{T}(t)$ exists.

Definition 3. A $t \longleftrightarrow \tau$ transformation is a transformation of time scale from t to τ and is defined by a function of the form $\tau = g(t)$

The invariable systems can be of two different kinds:

1. Algebraically invariable systems: the LTV systems that can be transformed into LTI by means of an algebraic transformation alone.
2. τ-algebraically invariable systems: the LTV systems that can be transformed into LTI systems using an algebraic transformation plus the $t \longleftrightarrow \tau$ transformation.

It can be demonstrated that an LTV of the form Eq.(58) is invariable [33]. It is also algebraically invariable if the state transition matrix (STM) of the system can be found. Unfortunately, in this case, the definition of such STM is extremely cumbersome. Consequently, a $t \longleftrightarrow \tau$ transformation must be employed. The following theorem is valid for τ-algebraically invariable systems.

Theorem:. The linear time-varying system

$$\dot{\mathbf{x}} = A(t)\mathbf{x} \tag{59}$$

is τ-algebraically invariable if the STM of the system in Eq. (59) *is of the form*

$$\Phi(t, t_0) = T(t, t_0) \exp[Rg(t, t_0)], \quad T(t_0, t_0) = \boldsymbol{I} \tag{60}$$

where $\dot{g}(t)$ exists and t_0 is chosen so that $g(t_0, t_0) = 0$.

In particular, the algebraic transformation

$$\mathbf{x}(t) = T(t, t_0)\bar{\mathbf{x}}(t) \tag{61}$$

together with the $t \longleftrightarrow \tau$ transformation

$$\tau = g(t, t_0) \tag{62}$$

will transform the system of Eq. (59) into the time-invariant system

$$\dot{\mathbf{z}}(\tau) = R\mathbf{z}(\tau) \tag{63}$$

where $\mathbf{z}(\tau) = \bar{\mathbf{x}}(t)$ and $\dot{\mathbf{z}}(\tau) = d\mathbf{z}(\tau)/d\tau$

Note that by using the definition of $\Phi(t, t_0)$ and the fact that $\exp[Rg(t, t_0)]$ is non-singular, from 60 we have:

$$A(t)T(t, t_0) - \dot{T}(t, t_0) = T(t, t_0)R\dot{g}(t, t_0) \tag{64}$$

which means

$$R\dot{g}(t, t_0) = T^{-1}\left(A(t)T - \dot{T}\right) \tag{65}$$

which will be important in the following section. Once the LTV system has been transformed into a LTI one, the problem of stability is addressed. There are two paths that can be taken in order to prove stability:

- The eigenvalues of the LTI system matrix R are computed; if the real part of all the eigenvalues is negative or at most 0, the system is stable.
- The State Transition Matrix (STM) of the original LTV system is retrieved. If it is possible to demonstrate that it is bounded at all time, then the system is stable.

6.3. Stability of the A-ZEM/ZEV algorithm

For the A-ZEM/ZEV guidance, the matrix A is the following:

$$A = \begin{bmatrix} -\frac{K_R}{t_{go}} & -K_V \\ -\frac{K_R}{t_{go}^2} & -\frac{K_V}{t_{go}} \end{bmatrix} \tag{66}$$

using Eq.(65) with

$$T = \begin{bmatrix} 1 & 0 \\ 0 & \frac{t_f}{t_{go}} \end{bmatrix} \tag{67}$$

the system in Eq.(66) can be transformed in the algebraically equivalent system, as follows:

$$R\dot{g}(t,t_0) = \begin{bmatrix} -K_R & -K_V t_f \\ -\frac{K_R}{t_f} & -(K_V+1) \end{bmatrix} \frac{1}{t_{go}} \tag{68}$$

with

$$R = \begin{bmatrix} -K_R & -K_V t_f \\ -\frac{K_R}{t_f} & -(K_V+1) \end{bmatrix} \tag{69}$$

and

$$\dot{g}(t,t_0) = \frac{1}{t_{go}} \tag{70}$$

The resulting system is simpler but still dependent on time. In order to make it time-invariant, the $t \longleftrightarrow \tau$ transformation must be applied. The time basis transformation is

$$\tau = g(t,t_0) = \int_{t_0}^{t} \frac{1}{t_{go}} d\tau = -\log \frac{t_{go}}{t_f} \tag{71}$$

where t_0 has been chosen so that $g(t_0,t_0) = 0$. With this transformation, the system is now a LTI system

$$\dot{\mathbf{z}}(\tau) = R\mathbf{z}(\tau) \tag{72}$$

with system matrix R. The stability of the system can be proven by finding the eigenvalues of such matrix and prove they have negative real part at all times. The R matrix in Eq.(69) has eigenvalues

$$\lambda_{1,2} = \frac{-K \pm \sqrt{K^2 - 4K_R}}{2}, \quad K = K_R + K_V + 1 \tag{73}$$

The stability conditions can be found in two cases: $\Delta \geq 0$ and $\Delta < 0$, where $\Delta = K^2 - 4K_R$.

Case 1: When $\boldsymbol{\Delta \geq 0}$.

. The condition $\Delta \geq 0$ translates into

$$K_V^2 + K_R^2 + 2K_R K_V + 2K_V - 2K_R + 1 \geq 0 \tag{74}$$

and means that the eigenvalues are purely real. 74 must be verified in order for the following stability condition to hold:

$$-K \pm \sqrt{\Delta} < 0 \tag{75}$$

or

$$\begin{aligned} -K + \sqrt{\Delta} < 0 \quad &\rightarrow \quad K > \sqrt{\Delta} \\ -K - \sqrt{\Delta} < 0 \quad &\rightarrow \quad K > -\sqrt{\Delta} \end{aligned} \tag{76}$$

which means that, since $\sqrt{\Delta} > 0$, the condition for stability is

$$K > \sqrt{\Delta} = \sqrt{K^2 - 4K_R} \tag{77}$$

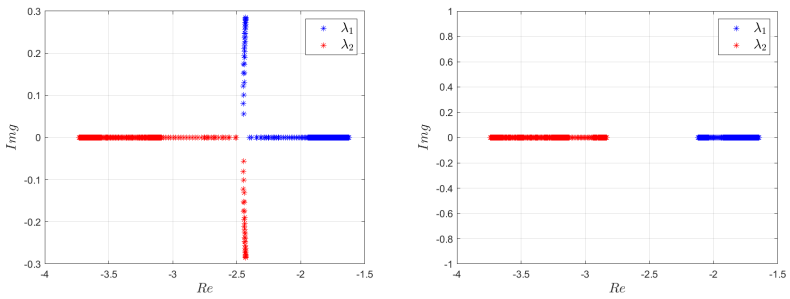
Case 2: When $\boldsymbol{\Delta < 0}$.

. If $\Delta < 0$, so

$$K_V^2 + K_R^2 + 2K_R K_V + 2K_V - 2K_R + 1 < 0 \tag{78}$$

the eigenvalues have a real and an imaginary part. In order for the system to be stable, the real part must be negative. So in this case the stability condition is simply

$$-K < 0 \quad \rightarrow \quad K > 0 \quad \rightarrow \quad K_R + K_V + 1 > 0 \tag{79}$$



(a) Case 1: 2D (b) Case 2: 3D

Figure 13: Eigenvalues during A-ZEM/ZEV-guided powered descent

These two conditions offer a quick way to check for the instantaneous stability of the guidance algorithm. This can be added as a checking step inside the control loop with a relatively low computational cost and action can be taken in case the algorithm goes in unstable regions. To prove that the algorithm remains stable in our test cases, the eigenvalues were computed along the descent trajectories in both cases (both 2D and 3D) and the results are reported in Figure 13. It is clear that the real part of the eigenvalues all remains strictly negative which ensures stability.

As stated in Section 6.2, another way of addressing the stability of the closed-loop dynamics is to employ the State Transition Matrix (STM). In particular, the STM of the LTV system must be bounded at all times in order for it to be stable. The calculation of the state transition matrix of the LTV system is derived from the STM of the LTI system. Letting Φ^* be the STM of the LTI system, then the STM of the original LTV system is $\Phi(t, t_0) = T\Phi^*$. According to [35], the STM of a linear system can be found from the knowledge of eigenvalues and eigenvectors as

$$\Phi(t, t_0) = \mathbf{M} e^{\mathbf{\Lambda} t} \mathbf{M}^{-1} \tag{80}$$

Where

$$\mathbf{M} = \begin{bmatrix} \mathbf{m}_1 & | & \mathbf{m}_2 & | & \dots & | & \mathbf{m}_n \end{bmatrix} \tag{81}$$

is the matrix whose columns are the eigenvectors and

$$\mathbf{\Lambda} = \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_n \end{bmatrix} \tag{82}$$

is the diagonal matrix of the eigenvalues. In this case, applying such definitions to the algebraically equivalent system in 68 and then the T transformation, the STM of the original system turns out to be:

$$\Phi = \begin{bmatrix} \Phi_{11} & \Phi_{12} \\ \Phi_{21} & \Phi_{22} \end{bmatrix} \tag{83}$$

with

$$\Phi_{11} = \frac{\lambda_1+k_r}{\lambda_1-\lambda_2}\left(\frac{t_{go}}{t_f}\right)^{-\lambda_2} - \frac{\lambda_2+k_r}{\lambda_1-\lambda_2}\left(\frac{t_{go}}{t_f}\right)^{-\lambda_1} \tag{84}$$

$$\Phi_{12} = -\frac{k_v t_f}{\lambda_1-\lambda_2}\left(\frac{t_{go}}{t_f}\right)^{-\lambda_1} + \frac{k_v t_f}{\lambda_1-\lambda_2}\left(\frac{t_{go}}{t_f}\right)^{-\lambda_2} \tag{85}$$

$$\Phi_{21} = \frac{\lambda_1+k_r}{k_v t_f}\frac{\lambda_2+k_r}{\lambda_1-\lambda_2}\left(\frac{t_{go}}{t_f}\right)^{-\lambda_1-1} - \frac{\lambda_2+k_r}{k_v t_f}\frac{\lambda_1+k_r}{\lambda_1-\lambda_2}\left(\frac{t_{go}}{t_f}\right)^{-\lambda_2-1} \tag{86}$$

$$\Phi_{22} = \frac{\lambda_1+k_r}{\lambda_1-\lambda_2}\left(\frac{t_{go}}{t_f}\right)^{-\lambda_1-1} - \frac{\lambda_2+k_r}{\lambda_1-\lambda_2}\left(\frac{t_{go}}{t_f}\right)^{-\lambda_2-1} \tag{87}$$

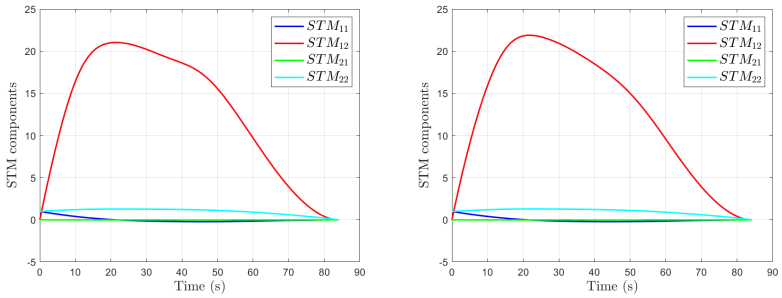
where

$$\lambda_{1,2} = \frac{-K \pm \sqrt{K^2 - 4K_R}}{2}, \quad K = K_R + K_V + 1 \tag{88}$$

According to the theory on non-autonomous linear systems, the LTV in equation 58 is stable if and only if the STM in 83 is bounded at all time $0 \leq t \leq t_f$. The STM in equation 83 was computed along the entire guided descent trajectories for both 2D and 3D cases. Figure 14 shows the STM for two sampled guided trajectories. It is clear that the components of the STM are bounded at all times so we can conclude that, at least in these cases, the algorithm is stable.

7. Conclusions and future efforts

The work has shown a novel closed-loop spacecraft guidance algorithm for soft landing. Using machine learning, in particular an actor-critic algorithm



(a) Case 1: 2D (b) Case 2: 3D

Figure 14: State transition matrix components

based on policy gradient, with advantage function estimation, it has been possible to expand the capabilities of classical ZEM/ZEV feedback guidance. The resulting *adaptive* algorithm (A-ZEM/ZEV) improves its performance in terms of fuel consumption and allows path constraints to be implemented directly into the guidance law. The actor-critic algorithm outputs a trained guidance neural network that can be implemented directly on-board. Provided that the spacecraft is equipped with the appropriate set of sensor for state determination, the guidance is recovered as function of the state in real time and can be followed by the control system, allowing for a completely autonomous soft landing maneuver in presence of path constraints. Moreover the stability of the method has been addressed: by transforming the LTV system in a LTI one, it has been possible to easily check for instability condition, either during post processing or, more importantly, online in the control loop. This ensures that countermeasures can be applied if the unstable regions are visited.

From a machine learning prospective, ELM have shown to work well as critic in an actor-critic algorithm. This shows that when the task is a simple regression problem like in this case, deep learning is not needed. A shallow network is enough and the one-step learning process of ELM makes them perfect for the task. They scale well with input dimension achieving good accuracy with a negligible computational time if compared with the total iteration time.

The work presented in this paper shows the advantages of using machine learning to learn state dependent parameters in an existing parametrized policy. In this case, using ZEM/ZEV guidance as a baseline allows for extremely precise terminal guidance with the increased flexibility given by the usage of reinforcement learning. This can be expanded to other guidance problem that rely on a parametrized law. As long as the state space can be discretized effectively, this ELM-based actor-critic algorithm can be applied. Convergence of the method is guaranteed by the fact that the advantage function estimation is unbiased and the cost function is set up correctly but convergence is still slow. Work can be done to improve the learning performance. For example, meta-learning could be used to learn different sequential task in order to speed up the process, especially if the environmental condition change (i.e. actuator or sensor failure). Meta-learning could also be used to create an algorithm that is more robust mis-modelled dynamics. By learning over a distributions of MDPs corresponding to different randomized instances of the environment, the agent should in fact be able to adapt to a wider set of environmental parameters, ultimately improving the performance in a more realistic set up.

References

[1] Shotwell, Robert, "Phoenix—the first Mars Scout mission" *Acta Astronautica*, Vol. 57, No. 2-8, pp. 121-134, 2005.

[2] Grotzinger, John P and Crisp, Joy and Vasavada, Ashwin R and Anderson, Robert C and Baker, Charles J and Barry, Robert and Blake, David F and Conrad, Pamela and Edgett, Kenneth S and Ferdowski, Bobak , "Mars Science Laboratory mission and science investigation" *Space science reviews*, Vol. 170, No. 1-4, pp. 5-56, 2012.

[3] Burns, Jack O and Mellinkoff, Benjamin and Spydell, Matthew and Fong, Terrence and Kring, David A and Pratt, William D and Cichan, Timothy and Edwards, Christine M , "Science on the lunar surface facilitated by

low latency telerobotics from a Lunar Orbital Platform-Gateway" *Acta Astronautica*, 2018.

[4] Steltzner, Adam D and Burkhart, P Dan and Chen, Allen and Comeaux, Keith A and Guernsey, Carl S and Kipp, Devin M and Lorenzoni, Leila V and Mendeck, Gavin F and Powell, Richard W and Rivellini, Tommaso P and others , "Mars science laboratory entry, descent, and landing system overview" *Pasadena, CA: Jet Propulsion Laboratory, National Aeronautics and Space Administration*, 2010.

[5] Klumpp, Allan R , "Apollo lunar descent guidance" *Automatica*, Vol. 10, No. 2, pp. 133-146, 1974.

[6] Singh, Gurkirpal and SanMartin, Alejandro M and Wong, Edward C , "Guidance and control design for powered descent and landing on Mars" *Aerospace Conference, 2007 IEEE*, pp. 1-8, 2007.

[7] Acikmese, B. and Ploen, S.R., "Convex programming approach to powered descent guidance for mars landing" *Journal of Guidance, Control, and Dynamics*, Vol. 30, No. 5, pp. 1353-1366, 2007.

[8] Blackmore, Lars and Acikmese, Behcet and Scharf, Daniel P , "Minimum-landing-error powered-descent guidance for Mars landing using convex optimization" *Journal of guidance, control, and dynamics*, Vol. 33, No. 4, pp. 1161-1171, 2010.

[9] Açıkmeşe, Behçet and Carson, John M and Blackmore, Lars , "Lossless convexification of nonconvex control bound and pointing constraints of the soft landing optimal control problem" *IEEE Transactions on Control Systems Technology*, Vol. 21, No. 6, pp. 2104-2113, 2013.

[10] Trawny, Nikolas and Benito, Joel and Tweddle, Brent E and Bergh, Charles F and Khanoyan, Garen and Vaughan, Geoffrey and Zheng, Jason and Villalpando, Carlos and Cheng, Yang and Scharf, Daniel P and others , "Flight testing of terrain-relative navigation and large-divert guidance

on a VTVL rocket" *AIAA SPACE 2015 Conference and Exposition*, pp. 4418, 2015.

[11] Liu, Xinfu and Lu, Ping and Pan, Binfeng , "Survey of convex optimization for aerospace applications" *Astrodynamics*, Vol. 1, No. 1, pp. 23-40, 2017.

[12] Lu, P., "Propellant-Optimal Powered Descent Guidance" *Journal of Guidance, Control, and Dynamics*, Vol. 41, No. 4, pp. 813-826, 2017.

[13] Lu, Ping and Sostaric, Ronald R and Mendeck, Gavin F , "Adaptive Powered Descent Initiation and Fuel-Optimal Guidance for Mars Applications" *2018 AIAA Guidance, Navigation, and Control Conference*, pp. 0616, 2018.

[14] Lu, Ping and Pan, Binfeng , "Highly constrained optimal launch ascent guidance" *Journal of Guidance, Control, and Dynamics*, Vol. 33, No. 2, pp. 404-414, 2010.

[15] Lu, Ping and Griffin, Brian J and Dukeman, Gregory A and Chavez, Frank R , "Rapid optimal multiburn ascent planning and guidance" *Journal of Guidance, Control, and Dynamics*, Vol. 31, No. 6, pp. 1656-1664, 2008.

[16] Lu, Ping and Forbes, Stephen and Baldwin, Morgan , "A versatile powered guidance algorithm" *AIAA Guidance, Navigation, and Control Conference*, pp. 4843, 2012.

[17] Guo, Yanning and Hawkins, Matt and Wie, Bong , "Waypoint-optimized zero-effort-miss/zero-effort-velocity feedback guidance for mars landing" *Journal of Guidance, Control, and Dynamics*, Vol. 36, No. 3, pp. 799-809, 2013.

[18] Pinson, R. and Lu, P. , "Trajectory design employing convex optimization for landing on irregularly shaped asteroids", AIAA/AAS Astrodynamics Specialist Conference, 2016.

[19] Zhang, C. and Topputo, F. and Bernelli-Zazzera, F. and Zhao, Y., "Low-thrust minimum-fuel optimization in the circular restricted three-body problem" *Journal of Guidance, Control, and Dynamics*, Vol. 38, No. 8, pp. 1501–1510, 2018.

[20] Lu, P. and Liu, X., "Autonomous trajectory planning for rendezvous and proximity operations by conic optimization" *Journal of Guidance, Control, and Dynamics*, Vol. 36, No. 2, pp. 375-389, 2013.

[21] Rao, A.V, "A survey of numerical methods for optimal control" *Advances in the Astronautical Sciences*, Vol. 135, No. 1, pp. 497-528, 2009.

[22] Betts, J.T, "Practical methods for optimal control and estimation using nonlinear programming" *Siam*, Vol. 19, 2010.

[23] Guo, Y. and Hawkins, M. and Wie, B. , "Applications of generalized zero-effort-miss/zero-effort-velocity feedback guidance algorithm" *Journal of Guidance, Control, and Dynamics*, Vol. 36, No. 3, pp. 810-820, 2013.

[24] Guo, Y. and Hawkins, M. and Wie, B. , "Optimal feedback guidance algorithms for planetary landing and asteroid intercept" *AAS/AIAA astrodynamics specialist conference*, Vol. 36, No. 3, pp. 588, 2011.

[25] Furfaro, R. and Linares, R. , "Waypoint-Based Generalized ZEM/ZEV Feedback Guidance for Planetary Landing via a Reinforcement Learning Approach" *3rd IAA Conference on Dynamics and Control of Space Systems, Moscow, Russia*, 2017.

[26] Furfaro, R. and Wibben, R. D. , "Robustification of a class of guidance algorithms for planetary landing: Theory and applications" *26th AAS/AIAA Space Flight Mechanics Meeting, 2016. Univelt Inc.*, 2016.

[27] Huang, G. B. , "What are extreme learning machines? Filling the gap between Frank Rosenblatt's dream and John von Neumann's puzzle" *Cognitive Computation 7.3*, 2015.

[28] Huang, G. B. and Wang, D. H. and Lan, Y. , "Extreme learning machines: a survey" *International journal of machine learning and cybernetics*, Vol. 2, No. 2, pp. 107-122, 2011.

[29] Silver, D. and Lever, G. and Heess, N. and Degris, T. and Wierstra, D. and Riedmiller, M. , "Deterministic Policy Gradient Algorithms" *ICML*, 2014.

[30] Sutton, R. S. and Barto, A. G. , "Reinforcement learning: An introduction" *Cambridge: MIT press*, Vol. 1, No. 1, 1998.

[31] Sutton, R. S. and McAllester, D. and Singh, S. and Mansour, Y. , "Policy gradient methods for reinforcement learning with function approximation" *Advances in neural information processing systems*, 2000.

[32] Williams, R. J. , "Reinforcement learning" *Springer*, 1992.

[33] Wu, M. , "Transformation of a linear time-varying system into a linear time-invariant system" *International Journal of Control*, Vol. 24, No. 4, pp. 589-602, 1978.

[34] Hagan, M. T. and Menhaj, M. B. , "Training feedforward networks with the Marquardt algorithm" *IEEE transactions on Neural Networks*, Vol. 5, No. 6, pp. 989-993, 1994.

[35] Rowell, D. , "Time-domain solution of LTI state equations" *Class Handout in Analysis and Design of Feedback Control System*, 2002.

[36] Patterson, Michael A., and Anil V. Rao. "GPOPS-II: A MATLAB software for solving multiple-phase optimal control problems using hp-adaptive Gaussian quadrature collocation methods and sparse nonlinear programming" ACM Transactions on Mathematical Software (TOMS) 41.1 (2014): 1-37.