Journal of Guidance, Control, and Dynamics
Vol. 41, No. 5, May 2018

Real-Time Optimal Control via Deep Neural Networks: Study on Landing Problems

Carlos Sánchez-Sánchez* and Dario Izzo†
ESA, 2201 AZ Noordwijk, The Netherlands

DOI: 10.2514/1.G002357

Recent research has shown the benefits of deep learning, a set of machine learning techniques able to learn deep architectures, for modelling robotic perception and action. In terms of a spacecraft navigation and control system, this suggests that deep architectures may be considered now to drive all or part of the onboard decision-making system. In this paper, this claim is investigated in more detail, training deep artificial neural networks to represent the optimal control action during a pinpoint landing and assuming perfect state information. It is found possible to train deep networks for this purpose, and the resulting landings, driven by the trained networks, are close to simulated optimal ones. These results allow for the design of an onboard real-time optimal control system able to cope with large sets of possible initial states while still producing an optimal response.

Nomenclature

$\mathcal{A}$	=	initialization area for the spacecraft states
g	=	activation function of a neural network
$\boldsymbol{g}$	=	gravity force vector
g	=	planetary gravity, m/s^2
g_0	=	Earth's gravity, m/s^2
$\mathcal{H}$	=	Hamiltonian
I_{sp}	=	specific impulse, s
J	=	cost function
m	=	mass, kg
$\mathcal{N}$	=	artificial neural network
$\boldsymbol{r}$	=	position vector
$\boldsymbol{u}$	=	control variables
$\boldsymbol{v}$	=	velocity vector
v_x	=	horizontal velocity, m/s
v_y	=	vertical velocity, m/s
$\boldsymbol{w}, \boldsymbol{b}$	=	weights and biases of a neural network
$\boldsymbol{x}$	=	state
x	=	horizontal position, m
y	=	vertical position, m
γ	=	weights of the cost function terms
θ	=	pitch, rad
λ	=	costate vector

Subscripts

f	=	final conditions
MOC	=	mass optimal control
QC	=	quadratic control
TOC	=	time optimal control
t	=	target conditions
0	=	initial conditions

I. Introduction

THANKS to the decreasing cost of computational resources and to theoretical advances in research to train neural networks with many hidden layers [1,2], there is a renewed interest in artificial neural networks (ANNs) and, in particular, in deep neural networks (DNNs). DNNs are artificial neural networks with several hidden layers that, layer after layer, form a hierarchical representation of an input–output map able, when correctly learned, to produce striking results. Examples of successful applications of DNNs include artificial intelligence for games [3], language processing [4], and image understanding [5], to just name some recent successes.

Although deep networks' representation capabilities are particularly appropriate for perception-related tasks such as image and speech recognition, it has been more recently pointed out how control problems may also benefit from these models [6,7]. In these works, interesting results were obtained in cases where incomplete state information was available. Work in this direction has thus mostly taken the standpoint of dynamic programming, mapping the problem to that of learning some approximation of state–action pairs coming from a mixture of reinforcement learning techniques and dynamic programming algorithms. Instead, the use of deep artificial neural networks to approximate the state–action pairs computed from the solution to the optimal control problem of deterministic continuous nonlinear systems, where the full state is directly observed, has been largely neglected.

Some past attempts to explore possible uses of (shallow) ANNs in connection to the optimal control theory of deterministic continuous-time nonlinear systems are noteworthy, although they are limited to simple domains (e.g., linear systems often appearing in case studies) or to unbounded control [8–10]. Contributions to the solution of both the Hamilton–Jacobi–Bellmann (HJB) equations and the two-point boundary value problem resulting from Pontryagin's optimal control theory showed possible uses of ANNs in the domain of deterministic, continuous optimal control [11]. On the one hand, several methods were proposed for the approximation of the value function $v(t, \boldsymbol{x})$ by means of ANNs' architectures [10,12,13]. On the other hand, ANNs have been proposed and studied to provide a trial solution to the states, to the costates, and to the controls so that their weights can be trained to make sure the assembled trial Hamiltonian respects Pontryagin's conditions [8]. In this last case, the networks have to be retrained for each initial condition. Recursive networks have also been used to learn near-optimal controllers for motion tasks [14] where the velocities (kinematics), and not the actual control, are to be predicted. A deep neural network trained with supervised signals, in the form of a stacked autoencoder, was recently shown [15] to be able to learn an accurate temporal profile of the optimal control and state in a point-to-point reach, nonlinear limb model, but their architecture enforced the notable restriction of being based on a fixed-time problem. Recent work on onboard optimal guidance achieved good results by means of the theory of lossless convexification, with an onboard demand to the computational units on the order of seconds [16,17], which is attractive for some applications.

Received 22 July 2016; revision received 27 October 2017; accepted for publication 24 November 2017; published online 26 February 2018.

*Scientist, European Space Research and Technology Center, Advanced Concepts Team.

†Scientific Coordinator, European Space Research and Technology Center, Advanced Concepts Team.

In this paper, DNNs are successfully trained to represent the solution to the Hamilton–Jacobi–Bellmann policy equation in four different cases of pinpoint landing: a quadcopter model, a mass varying spacecraft with bounded thrust, a mass varying spacecraft equipped with a reaction wheel for attitude control, and a mass varying rocket with thrust vector control. In all cases, the landing scenario is studied by assuming perfect information on the spacecraft state. Approaches like the guided policy search or dynamic programming hybrids [6] are thus not necessary, and a simpler training architecture can be assembled. Due to the assumptions considered in the models, feedforward DNN architectures can be trained directly in a supervised manner on the optimal state–action pairs obtained via an indirect method (based on single shooting). The trained networks are suitable for the onboard generation of descent guidance profiles because their computation requires a modest CPU effort. Training, on the other hand, can be done offline, and is thus not of concern to a real-time optimal control architecture.

The resulting DNNs thus enable real-time optimal control capabilities, without relying on optimal control methods (direct or indirect) on board, which could lead to an excessive use of the CPU and is undesirable due to numeric instabilities often connected to such solvers. In this sense, our work is related to previous attempts to obtain pinpoint landing guidance profiles computable on board [18], and it offers a novel, valid alternative. Remarkably, the learned policies have a validity that extends outside the area where training data are computed, contributing to their robustness and use possibilities.

This paper builds on, and completes, previous work [19] where, notably, the state–action pairs were computed via direct methods, and thus subject to chattering noise that prevented the study of more complex models such as pinpoint landing and thrust vectoring. The paper is structured as follows: In Sec. II we introduce the generic mathematical form of the optimal control problems (OCPs) considered, and we give the formal definition of the optimal control policy to be learned by the DNNs. In the following Sec. III, four instances of OCPs are introduced, which are all related to pinpoint landing scenarios of relevance to aerospace systems; in each case, the two-point boundary value problem (TPBVP) is derived from the application of Pontryagin's maximum principle [20]. In the following Sec. IV, we describe how the TPBVPs are solved by means of single shooting and continuation (homotopy) techniques to generate training data (optimal state–action pairs), uniformly covering a large region of interest. In Sec. V, we describe the network architectures and training procedures used to approximate the optimal solutions. Section VI defines how the results of the DNNs are compared to the optimal trajectories and, in Sec. VII, the performance of the networks is studied, including a comparison between different architectures and the study of the network behavior for cases not considered in the training data.

II. Optimal Control

Let us consider deterministic systems defined by the time-independent dynamics $\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t))$, where $\boldsymbol{x}(t){:}\mathbb{R} \to \mathbb{R}^{n_x}$ and $\boldsymbol{u}(t){:}\mathbb{R} \to \mathcal{U} \subset \mathbb{R}^{n_u}$. Consider the fundamental problem of finding an admissible control policy $\boldsymbol{u}(t)$ able to steer the system from any $\boldsymbol{x}_0 \in \mathbb{R}^{n_x}$ to some target $\mathcal{S} \subset \mathbb{R}^{n_x}$ in a (free) time t_f while minimizing the cost function:

$$J(\boldsymbol{x}(t), \boldsymbol{u}(t)) = \int_0^{t_f} \mathcal{L}(\boldsymbol{x}(t), \boldsymbol{u}(t))\, \mathrm{d}t + h(\boldsymbol{x}(t_f))$$

The value function, defined as

$$v(\boldsymbol{x}_0) = \min_u J(\boldsymbol{x}(t), \boldsymbol{u}(t)) \tag{1}$$

represents the minimal cost to reach the goal, starting from $\boldsymbol{x}_0$. Note how the value function is, in this case, not depending on time because t_f is left free to be optimized. A finite-horizon control problem is thus considered that has, mathematically, characteristics similar to an infinite horizon problem. Equivalently, the value function can be introduced as the solution to the partial differential equation [11]:

$$\min_u\{\mathcal{L}(\boldsymbol{x}, \boldsymbol{u}) + \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) \cdot \nabla_{\boldsymbol{x}} v(\boldsymbol{x})\} = 0 \tag{2}$$

subject to the boundary conditions $v(\boldsymbol{x}_t) = h(\boldsymbol{x}(t_f)),\ \forall \boldsymbol{x}_t \in \mathcal{S}$. The optimal control policy is then

$$\boldsymbol{u}^*(\boldsymbol{x}) = \operatorname*{argmin}_u\{\mathcal{L}(\boldsymbol{x}, \boldsymbol{u}) + \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) \cdot \nabla_{\boldsymbol{x}} v(\boldsymbol{x})\} \tag{3}$$

Equations (2) and (3) are the Hamilton–Jacobi–Bellman equations for the free-time deterministic, optimal control problem here considered. They are a set of extremely challenging partial differential equations for which the solution, pursued in the "viscosity" sense, is the solution to the original optimal control problem [21]. The HJB equations are important here because they imply the existence and uniqueness of an optimal state-feedback $\boldsymbol{u}^*(\boldsymbol{x})$ that, in turn, allows us to consider universal function approximators such as DNNs to represent it. Numerical approaches to solving HJB equations often rely on parametric approximations of the value function (e.g., using the Galerkin method [22]), and have thus considered ANNs for the same purpose in the past [13]. Here, deep neural networks are proposed to learn directly the optimal state-feedback $\boldsymbol{u}^*(\boldsymbol{x})$, thus also obtaining, indirectly, a representation of the value function $v(\boldsymbol{x}) = J(\boldsymbol{x}^*, \boldsymbol{u}^*)$ that can be computed by the numerical integration of the network outputs. Although the alternative of directly learning the value function has also been considered, in this case, obtaining the optimal state feedback from $v(x)$ would require us to compute the network gradients, which are likely to result in less accurate controls. Eventually, the trained DNN directly represents the optimal state feedback and can thus be used, for example, in a nonlinear model predictive control architecture [23] to achieve real-time optimal control capabilities.

III. Optimal Landing Control Problems

The OCPs that are considered here correspond to different landing scenarios, all under a uniform gravity field, where the control is, in each case, defined by two variables u_1 and u_2 ($n_u = 2$). Consider two different objectives: time optimal control (TOC) and quadratic control (QC) for the quadcopter model, and mass optimal control (MOC) and quadratic control for the spacecraft models. The resulting set of test cases represents different classes of control profiles, as illustrated in Fig. 1, including continuous control, discontinuous control, bang-off-bang control, and saturated control. A summary of the models' characteristics is shown in Table 1.

In the following subsections, the details of each of the models considered are described and the Pontryagin maximum principle is used to derive the corresponding two-point boundary value problem. If values for the initial values of the costates and for the final time t_f are found so that the dynamics and boundary conditions are satisfied, as well as the additional condition $\mathcal{H}(t_f) = 0$ (a free time problem is considered), the corresponding control along the trajectory is assumed to be optimal and is used to create a number of optimal state–action pairs used for training the DNNs.

A. Quadcopter (QUAD)

Consider the following set of ordinary differential equations (ODEs):

$$\begin{aligned}\dot{\boldsymbol{r}} &= \boldsymbol{v}\\ \dot{\boldsymbol{v}} &= c_1 \frac{u_1}{m} \hat{\boldsymbol{i}}_\theta + \boldsymbol{g}\\ \dot{\theta} &= c_2 u_2\end{aligned} \tag{4}$$

modeling the dynamics of a quadcopter moving in a two-dimensional space [24]. The state is determined by the position $\boldsymbol{r} = (x, z)$, the velocity $\boldsymbol{v} = (v_x, v_z)$, and the orientation θ of the quadcopter. The mass of the quadcopter is $m = 1$ kg; and the acceleration due to the Earth's gravity is $\boldsymbol{g} = (0, -g)$, where $g = 9.81\ \mathrm{m/s^2}$. The control $u_1 \in [0.05, 1]$ models a thrust action applied along the direction $\boldsymbol{i}_\theta = [\sin\theta, \cos\theta]$ bounded by a maximum magnitude $c_1 = 20$ N and a minimum magnitude of $0.05 c_1 = 1$ N. The control $u_2 \in [-1, 1]$

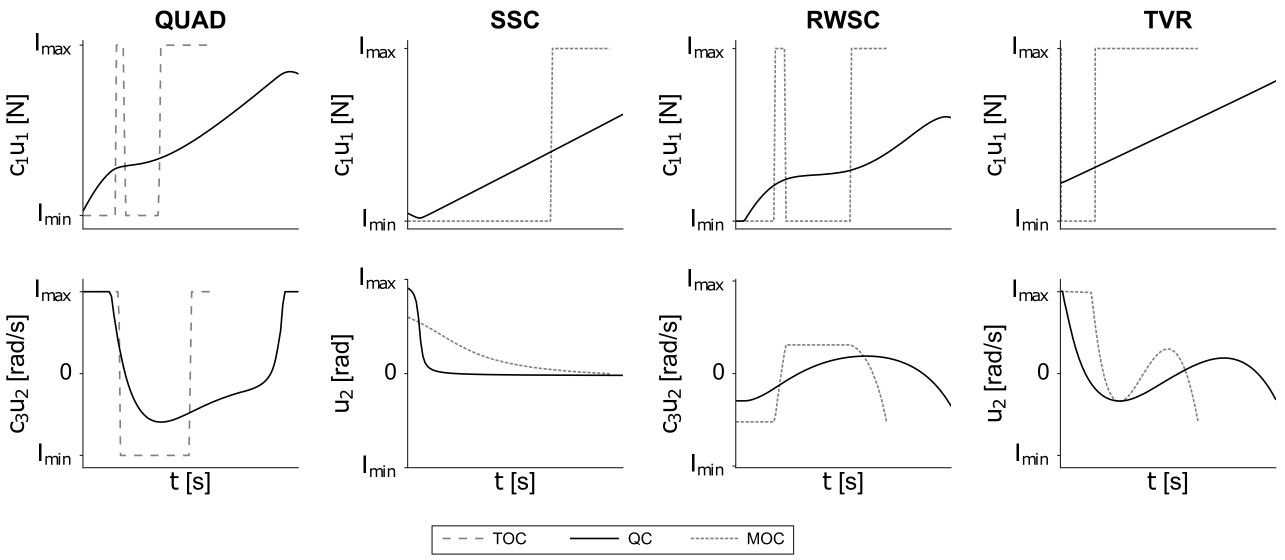

Fig. 1 Optimal control profiles of the models and objective functions here considered.

models the quadcopter pitch rate bounded by $c_2 = 2$ rad/s. Consider, as the target state, $\boldsymbol{r}_t = (0, 0)$, $\boldsymbol{v}_t = (0, 0)$, and $\theta = 0$. Consider the minimization of the cost function

$$J = (1-\alpha)\int_0^{t_f}(\gamma_1 c_1^2 u_1^2 + \gamma_2 c_2^2 u_2^2)\,\mathrm{d}t + \alpha\int_0^{t_f}\mathrm{d}t$$

where $\gamma_1 = 1\ 1/\mathrm{N}^2$ and $\gamma_2 = 1\ \mathrm{s}^2/\mathrm{rad}^2$ are weights defining the balance between the cost of using u_1 or u_2 to control the quadcopter, and α is a continuation parameter. The parameter $\alpha \in [0, 1]$ defines a continuation between a quadratic optimal control problem $\alpha = 0$ and a time optimal control problem (TOC) $\alpha = 1$. Following Pontryagin [20], consider the following Hamiltonian:

$$\mathcal{H} = \boldsymbol{\lambda}_r \cdot \boldsymbol{v} + \boldsymbol{\lambda}_v \cdot \left(c_1\frac{u_1}{m}\hat{\boldsymbol{i}}_\theta + \boldsymbol{g}\right) + \lambda_\theta c_2 u_2 + (1-\alpha)\Big(\gamma_1 c_1^2 u_1^2 + \gamma_2 c_2^2 u_2^2\Big) + \alpha$$

where the costate functions $\boldsymbol{\lambda}_r(t)$, $\boldsymbol{\lambda}_v(t)$, and $\lambda_\theta(t)$ are introduced. Because $u_1 \in [0.05, 1]$ and $u_2 \in [-1, 1]$ both appear as a quadratic term, from the maximum principle, it follows that their optimal values must be (if $\alpha \neq 1$)

$$u_1^* = \min\left(\max\left(-\frac{\boldsymbol{\lambda}_v\cdot\hat{\boldsymbol{i}}_\theta}{2\gamma_1(1-\alpha)c_1}, 0.05\right), 1\right)$$
$$u_2^* = \min\left(\max\left(-\frac{\lambda_\theta}{2\gamma_2(1-\alpha)c_2}, -1\right), 1\right) \tag{5}$$

and, if $\alpha = 1$ (TOC case),

$$u_1^* = \begin{cases}1 & S_1 < 0\\ 0.05 & S_1 > 0\end{cases} \qquad u_2^* = \begin{cases}1 & S_2 < 0\\ -1 & S_2 > 0\end{cases} \tag{6}$$

where $S_1 = \boldsymbol{\lambda}_v \cdot \hat{\boldsymbol{i}}_\theta$ and $S_2 = \lambda_\theta$ are called switching functions because they determine the control switch between extreme values of the domain where it is defined. The differential equations defining the costates $(\dot{\boldsymbol{\lambda}}_q = -(\partial\mathcal{H}/\partial q))$ are

$$\dot{\boldsymbol{\lambda}}_r = 0$$
$$\dot{\boldsymbol{\lambda}}_v = -\boldsymbol{\lambda}_r$$
$$\dot{\lambda}_\theta = -c_1 u_1 \boldsymbol{\lambda}_v \cdot \hat{\boldsymbol{i}}_\tau \tag{7}$$

which can be simplified considerably because the first four differential equations are trivial. Overall, the following holds:

$$\lambda_x = \lambda_{x0}$$
$$\lambda_z = \lambda_{z0}$$
$$\lambda_{vx} = \lambda_{vx0} - \lambda_{x0}t$$
$$\lambda_{vz} = \lambda_{vz0} - \lambda_{z0}t$$
$$\dot{\lambda}_\theta = -c_1 u_1[(\lambda_{vx0} - \lambda_{x0}t)\cos\theta - (\lambda_{vz0} - \lambda_{z0}t)\sin\theta)] \tag{8}$$

Eventually, the following two-point boundary value problem is obtained:

$$\dot{\boldsymbol{r}} = \boldsymbol{v}$$
$$\dot{\boldsymbol{v}} = \frac{c_1 u_1^*}{m}\hat{\boldsymbol{i}}_\theta + \boldsymbol{g}$$
$$\dot{\theta} = c_2 u_2^*$$
$$\dot{\lambda}_\theta = -c_1 u_1^*[(\lambda_{vx0} - \lambda_{x0}t)\cos\theta - (\lambda_{vz0} - \lambda_{z0}t)\sin\theta)] \tag{9}$$

with boundary conditions $\boldsymbol{r}_0$, $\boldsymbol{v}_0$, and θ_0 at $t = 0$ and $\boldsymbol{r}_t = (0, 0)$, $\boldsymbol{v}_t = (0, 0)$, and $\theta_t = 0$ at $t = t_f$.

Table 1 Four considered models at a glance

Model	n_x[a]	u_1	u_2	Variable mass	$\boldsymbol{g}$	Optimization problems
Quadcopter	5	N	rad/s	No	Earth	TOC, QC
Simple spacecraft	4	N	rad	Yes	Moon	MOC, QC
Reaction wheel spacecraft	5	N	rad/s	Yes	Moon	MOC, QC
Thrust vectoring rocket	6	N	rad	Yes	Moon	MOC, QC

[a]n_x: length of the state vector $\boldsymbol{x}$.

B. Simple Spacecraft (SSC)

Consider the following set of ODEs:

$$\begin{aligned}\dot{\boldsymbol{r}} &= \boldsymbol{v}\\ \dot{\boldsymbol{v}} &= c_1\frac{u_1}{m}\hat{\boldsymbol{i}}_\theta + \boldsymbol{g}\\ \dot{m} &= -\frac{c_1}{c_2}u_1\end{aligned} \tag{10}$$

modeling the dynamics of a SSC in a two dimensional space (mass-varying point mass). The state is determined by its position $\boldsymbol{r} = (x, z)$, its velocity $\boldsymbol{v} = (v_x, v_z)$ and its mass m. The acceleration $\boldsymbol{g} = (0, -g)$ considered is due to the moon's gravity, where $g = 1.6229\ \mathrm{m/s^2}$. The constant $c_2 = I_{\mathrm{sp}}g_0$ represents the rocket engine efficiency in terms of its specific impulse $I_{\mathrm{sp}} = 311$ s and $g_0 = 9.81\ \mathrm{m/s^2}$. The control $u_1 \in [0, 1]$ models a thrust action applied along the direction $\boldsymbol{i}_\theta = [\sin\theta, \cos\theta]$ bounded by a maximum magnitude $c_1 = 44{,}000$ N. Because the model does not include any rotational inertia, we assume to be able to freely steer the spacecraft pitch so that θ is to be considered as a second control input u_2. Consider as a target the states $\boldsymbol{r}_t = (0, 0)$, $\boldsymbol{v}_t = (0, 0)$, and any m. Consider the minimization of the cost function

$$J = \frac{1}{c_2}\left[(1-\alpha)\int_0^{t_f}\gamma_1 c_1^2 u_1^2\,\mathrm{d}t + \alpha\int_0^{t_f} c_1 u_1\,\mathrm{d}t\right]$$

where $\gamma_1 = 1\ 1/\mathrm{N}$ is a weight defining a tradeoff between the two contributions. The parameter $\alpha \in [0, 1]$ defines a continuation between a quadratic optimal control problem $\alpha = 0$ and a mass optimal control problem (MOC) $\alpha = 1$. Following Pontryagin [20], consider the following Hamiltonian:

$$\begin{aligned}\mathcal{H} &= \boldsymbol{\lambda}_r\cdot\boldsymbol{v} + \boldsymbol{\lambda}_v\cdot\left(c_1\frac{u_1}{m}\hat{\boldsymbol{i}}_\theta + \boldsymbol{g}\right) - \lambda_m\frac{c_1}{c_2}u_1\\ &\quad + \frac{1}{c_2}\left[(1-\alpha)\gamma_1 c_1^2 u_1^2 + \alpha c_1 u_1\right]\end{aligned}$$

where the costate functions $\boldsymbol{\lambda}_r(t)$, $\boldsymbol{\lambda}_v(t)$ and $\lambda_m(t)$ are introduced. From the maximum principle, it immediately follows that, necessarily, the optimal value for u_2 (and hence θ) must be

$$\hat{\boldsymbol{i}}_\theta^* = -\frac{\boldsymbol{\lambda}_v}{\lambda_v}$$

whereas for u_1, because it appears quadratically, we may conclude that, if $\alpha \neq 1$,

$$u_1^* = \min\left(\max\left(\frac{(\lambda_v c_2/m) + \lambda_m - \alpha}{2\gamma_1 c_1(1-\alpha)}, 0\right), 1\right)$$

and, if $\alpha = 1$ (MOC case),

$$u_1^* = \begin{cases}1 & S_1 < 0\\ 0 & S_1 > 0\end{cases}$$

where

$$S_1 = \alpha - \frac{\lambda_v c_2}{m} - \lambda_m$$

is the switching function for this problem. The differential equations defining the costates ($\dot{\boldsymbol{\lambda}}_q = -(\partial\mathcal{H}/\partial q)$) are

$$\begin{aligned}\dot{\boldsymbol{\lambda}}_r &= 0\\ \dot{\boldsymbol{\lambda}}_v &= -\boldsymbol{\lambda}_r\\ \dot{\lambda}_m &= \frac{c_1}{m^2}\boldsymbol{\lambda}_v\cdot\hat{\boldsymbol{i}}_\theta u\end{aligned} \tag{11}$$

Eventually, the following two-point boundary value problem is obtained:

$$\begin{aligned}\dot{\boldsymbol{r}} &= \boldsymbol{v}\\ \dot{\boldsymbol{v}} &= c_1\frac{u_1^*}{m}\hat{\boldsymbol{i}}_\theta^* + \boldsymbol{g}\\ \dot{m} &= -\frac{c_1}{c_2}u_1^*\\ \dot{\lambda}_m &= \frac{c_1}{m^2}u_1^*[(\lambda_{vx0} - \lambda_{x0}t)\sin\theta + (\lambda_{vz0} - \lambda_{z0}t)\cos\theta)]\end{aligned} \tag{12}$$

with boundary conditions $\boldsymbol{r}_0$, $\boldsymbol{v}_0$ and m_0 at $t = 0$ and $\boldsymbol{r}_t = (0, 0)$, $\boldsymbol{v}_t = (0, 0)$, and $\lambda_{mt} = 0$ at $t = t_f$.

C. Reaction Wheel Spacecraft (RWSC)

In this model, the spacecraft attitude is controlled by a reaction wheel able to induce a bounded angular velocity on the spacecraft body (we neglect the fact that, as the spacecraft becomes lighter, the maximum angular velocity also increases, as well as the fact that the wheel may get saturated). The state is thus the position $\boldsymbol{r} = (x, z)$, the velocity $\boldsymbol{v} = (v_x, v_z)$, and the mass m. The system dynamics is described by the following set of ODEs:

$$\begin{aligned}\dot{\boldsymbol{r}} &= \boldsymbol{v}\\ \dot{\boldsymbol{v}} &= c_1\frac{u_1}{m}\hat{\boldsymbol{i}}_\theta + \boldsymbol{g}\\ \dot{\theta} &= c_3 u_2\\ \dot{m} &= -\frac{c_1}{c_2}u_1\end{aligned} \tag{13}$$

Similarly to what was considered for the SSC model, the acceleration $\boldsymbol{g} = (0, -g)$ is due to the moon's gravity where $g = 1.6229\ \mathrm{m/s^2}$; whereas the constant $c_2 = I_{\mathrm{sp}}g_0$ represents the rocket engine efficiency in terms of its specific impulse $I_{\mathrm{sp}} = 311$ s and $g_0 = 9.81\ \mathrm{m/s^2}$. Although the control u_1 has the same meaning as in the previous SSC model, $u_2 \in [-1, 1]$ now corresponds to the pitch rate control actuated by a reaction wheel and is bounded by a maximum magnitude of $c_3 = 0.0698$ rad/s. This difference also results in rather different optimal landing trajectories, as illustrated in the example in Fig. 2. Consider as a target state for this system

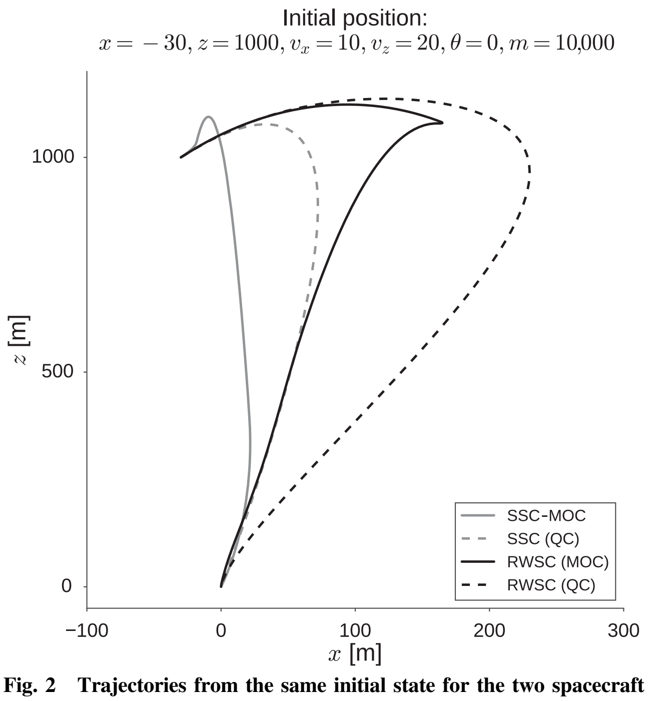

Fig. 2 Trajectories from the same initial state for the two spacecraft models and the two objective functions.

$\boldsymbol{r}_t = (0,0)$, $\boldsymbol{v}_t = (0,0)$, $\theta_t = 0$, and $m =$ any; note that, unlike in the SSC case, now, a terminal vertical descent is forced, thanks to the final condition on the attitude $\theta_t = 0$. Consider the minimization of the cost function:

$$J = (1-\alpha)\int_0^{t_f}\left[\frac{\gamma_1 c_1^2}{c_2}u_1^2 + c_3^2 u_2^2\right]\mathrm{d}t + \alpha\int_0^{t_f}\left[\frac{\gamma_2 c_1}{c_2}u_1 + c_3^2 u_2^2\right]\mathrm{d}t$$

where $\gamma_1 = 1.5E^{-6}$ $(\mathrm{rad}^2\cdot\mathrm{s})/(\mathrm{kg}^2\cdot\mathrm{m})$ and $\gamma_2 = 1.5E^{-2}$ $\mathrm{rad}^2/(\mathrm{s}\cdot\mathrm{kg})$ are defining the cost tradeoff between the use of u_1 and u_2 to control the spacecraft trajectory and are chosen as to give priority to optimize the use of u_1 (i.e., the thruster). The parameter $\alpha \in [0,1]$ defines a continuation between a quadratic optimal control problem ($\alpha = 0$) and a mass optimal control problem (MOC, $\alpha = 1$) for u_1, whereas for u_2, the power spent by the reaction wheel is always considered. Following Pontryagin [20], consider the following Hamiltonian:

$$\begin{aligned}\mathcal{H} &= \boldsymbol{\lambda}_r\cdot\boldsymbol{v} + \boldsymbol{\lambda}_v\cdot\left(c_1\frac{u_1}{m}\hat{\boldsymbol{i}}_\theta + \boldsymbol{g}\right) - \lambda_m\frac{c_1}{c_2}u_1 + \lambda_\theta c_3 u_2\\ &\quad + (1-\alpha)\frac{\gamma_1 c_1^2}{c_2}u_1^2 + \alpha\frac{\gamma_2 c_1}{c_2}u_1 + c_3^2 u_2^2\end{aligned}$$

where the costate functions $\boldsymbol{\lambda}_r(t)$, $\boldsymbol{\lambda}_v(t)$, $\lambda_m(t)$, and λ_θ are introduced. From the maximum principle, it immediately follows that, necessarily, the optimal value for the controls must be, if $\alpha \neq 1$,

$$\begin{aligned}u_1^* &= \min\left(\max\left(-\frac{\boldsymbol{\lambda}_v\cdot\hat{\boldsymbol{i}}_\theta(c_2/m) - \lambda_m + \alpha\gamma_2}{2(1-\alpha)\gamma_1 c_1}, 0\right), 1\right)\\ u_2^* &= \min\left(\max\left(-\frac{\lambda_\theta}{2c_3}, -1\right), 1\right)\end{aligned} \tag{14}$$

and, if $\alpha = 1$ (MOC case),

$$u_1^* = \begin{cases}1 & S_1 < 0\\ 0 & S_1 > 0\end{cases}$$

where

$$S_1 = \boldsymbol{\lambda}_v\cdot\hat{\boldsymbol{i}}_\theta\frac{c_2}{m} - \lambda_m + \alpha\gamma_2$$

is the switching function for this problem. The differential equations defining the costates ($\dot{\boldsymbol{\lambda}}_q = -(\partial\mathcal{H}/\partial q)$) are

$$\begin{aligned}\dot{\boldsymbol{\lambda}}_r &= 0\\ \dot{\boldsymbol{\lambda}}_v &= -\boldsymbol{\lambda}_r\\ \dot{\lambda}_\theta &= -\frac{c_1}{m}\boldsymbol{\lambda}_v\cdot\hat{\boldsymbol{i}}_\tau u_1\\ \dot{\lambda}_m &= \frac{c_1}{m^2}\boldsymbol{\lambda}_v\cdot\hat{\boldsymbol{i}}_\theta u_1\end{aligned} \tag{15}$$

where we have introduced the unit vector $\hat{\boldsymbol{i}}_\tau = [\cos\theta, -\sin\theta]$. Eventually, the following two-point boundary value problem is obtained:

$$\begin{aligned}\dot{\boldsymbol{r}} &= \boldsymbol{v}\\ \dot{\boldsymbol{v}} &= c_1\frac{u_1}{m}\hat{\boldsymbol{i}}_\theta + \boldsymbol{g}\\ \dot{\theta} &= c_3 u_2\\ \dot{m} &= -\frac{c_1}{c_2}u_1\\ \dot{\lambda}_\theta &= -\frac{c_1}{m}u_1^*[(\lambda_{vx0} - \lambda_{x0}t)\cos\theta - (\lambda_{vz0} - \lambda_{z0}t)\sin\theta)]\\ \dot{\lambda}_m &= \frac{c_1}{m^2}u_1^*[(\lambda_{vx0} - \lambda_{x0}t)\sin\theta + (\lambda_{vz0} - \lambda_{z0}t)\cos\theta)]\end{aligned} \tag{16}$$

with boundary conditions $\boldsymbol{r}_0$, $\boldsymbol{v}_0$, m_0, and θ_0 at $t = 0$ and $\boldsymbol{r}_t = (0,0)$, $\boldsymbol{v}_t = (0,0)$, and $\lambda_{mt} = 0$ at $t = t_f$.

D. Thrust Vectoring Rocket (TVR)

Consider the following set of ODEs:

$$\begin{aligned}\dot{\boldsymbol{r}} &= \boldsymbol{v}\\ \dot{\boldsymbol{v}} &= c_1\frac{u_1}{m}\hat{\boldsymbol{t}} + \boldsymbol{g}\\ \dot{\theta} &= \omega\\ \dot{\omega} &= -c_1\frac{u_1}{Rm}\hat{\boldsymbol{t}}\cdot\hat{\boldsymbol{i}}_\tau\\ \dot{m} &= -\frac{c_1}{c_2}u_1\end{aligned} \tag{17}$$

modeling the dynamics of a rocket moving in a two-dimensional space and controlled by thrust vectoring as illustrated in Fig. 3. The state is determined by the position $\boldsymbol{r} = (x, z)$, velocity $\boldsymbol{v} = (v_x, v_z)$, orientation θ, angular velocity ω, and mass m of the rocket. The acceleration due to the moon's gravity is $\boldsymbol{g} = (0, -g)$, where $g = 1.6229\ \mathrm{m/s^2}$. The constant $c_2 = I_{\mathrm{sp}}g_0$ represents the rocket engine efficiency in terms of its specific impulse $I_{\mathrm{sp}} = 311$ s and $g_0 = 9.81\ \mathrm{m/s^2}$. The control $u_1 \in [0,1]$ N models a thrust action applied along the direction $\hat{\boldsymbol{t}}$ and bounded by a maximum magnitude $c_1 = 20$ N. The control $u_2 \in [-\phi, \phi]$ models the thrust vector tilt with respect to the symmetry axis, so that

$$\hat{\boldsymbol{t}} = \cos(\theta + u_2)\hat{\boldsymbol{i}}_x + \sin(\theta + u_2)\hat{\boldsymbol{i}}_y$$

We will consider $\phi = 10$ deg here. Consider, as a target state, $\boldsymbol{r}_t = (0,0)$, $\boldsymbol{v}_t = (0,0)$, $\theta_t = 0$, and $\omega_t = 0$. Consider the minimization of the cost function (note that the cost is the same as that defined for the SSC case in Sec. III.A):

$$J = \frac{1}{c_2}\left[(1-\alpha)\int_0^{t_f}\gamma_1 c_1^2 u_1^2\,\mathrm{d}t + \alpha\int_0^{t_f}c_1 u_1\,\mathrm{d}t\right]$$

where $\gamma_1 = 1$ 1/N is a weight defining a tradeoff between the two contributions. The parameter $\alpha \in [0,1]$ defines a continuation between a quadratic optimal control problem $\alpha = 0$ and a mass optimal control problem (MOC) $\alpha = 1$. Following Pontryagin [20], consider the following Hamiltonian:

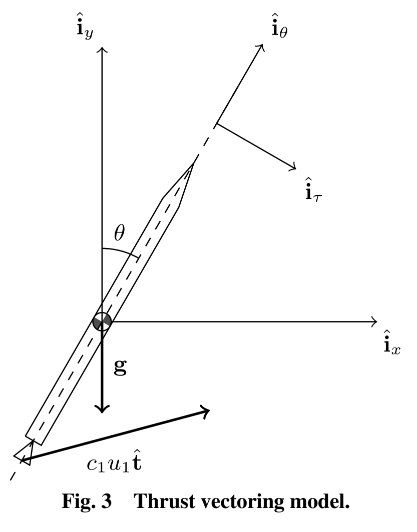

Fig. 3 Thrust vectoring model.

$$\mathcal{H} = \boldsymbol{\lambda}_r \cdot \boldsymbol{v} + \boldsymbol{\lambda}_v \cdot \left(c_1 \frac{u_1}{m} \hat{\boldsymbol{t}} + \boldsymbol{g} \right) + \lambda_\theta \omega - \lambda_\omega c_1 \frac{u_1}{Rm} \hat{\boldsymbol{t}} \cdot \hat{\boldsymbol{i}}_\tau - \lambda_m \frac{c_1}{c_2} u_1 + \frac{1-\alpha}{c_2} \gamma_1 c_1^2 u_1^2 + \frac{c_1}{c_2} \alpha u_1 \tag{18}$$

From the maximum principle, it immediately follows that, necessarily, the optimal value for $\hat{\boldsymbol{t}}$ (and hence u_2) must be

$$\hat{\boldsymbol{t}}^* = -\frac{\boldsymbol{\lambda}_v - (\lambda_\omega / R)\hat{\boldsymbol{i}}_\tau}{|\boldsymbol{\lambda}_v - (\lambda_\omega / R)\hat{\boldsymbol{i}}_\tau|} = -\frac{\boldsymbol{\lambda}_{\text{aux}}}{\lambda_{\text{aux}}}$$

where we have introduced the auxiliary costate $\boldsymbol{\lambda}_{\text{aux}} = \boldsymbol{\lambda}_v - (\lambda_\omega / R)\hat{\boldsymbol{i}}_\tau$. The Hamiltonian along an optimal trajectory may be then rewritten as follows:

$$\mathcal{H} = \boldsymbol{\lambda}_r \cdot \boldsymbol{v} + \boldsymbol{\lambda}_v \cdot \boldsymbol{g} + \lambda_\theta \omega - \lambda_m \frac{c_1}{c_2} u_1 - \lambda_{\text{aux}} \frac{c_1}{m} u_1 + \frac{1-\alpha}{c_2} \gamma_1 c_1^2 u_1^2 + \frac{c_1}{c_2} \alpha u_1 \tag{19}$$

and, because u_1 appears as a quadratic term, its optimal value must be, if $\alpha \neq 1$,

$$u_1^* = \min\left(\max\left(\frac{\lambda_m + (c_2/m)\lambda_{\text{aux}} - \alpha}{2\gamma_1 (1-\alpha) c_1}, 0 \right), 1 \right)$$

and, if $\alpha = 1$ (MOC case),

$$u_1^* = \begin{cases} 1 & S_1 < 0 \\ 0 & S_1 > 0 \end{cases}$$

where

$$S_1 = \alpha - \lambda_m - \frac{c_2}{m} \lambda_{\text{aux}}$$

is the switching function for this problem. The differential equations for the costates ($\dot{\lambda}_q = -(\partial \mathcal{H} / \partial q)$) are

$$\begin{aligned} \dot{\boldsymbol{\lambda}}_r &= 0 \\ \dot{\boldsymbol{\lambda}}_v &= -\boldsymbol{\lambda}_r \\ \dot{\lambda}_\theta &= -\frac{\lambda_\omega}{R} c_1 \frac{u_1}{m} \hat{\boldsymbol{t}} \cdot \hat{\boldsymbol{i}}_\theta \\ \dot{\lambda}_\omega &= -\lambda_\theta \\ \dot{\lambda}_m &= \frac{c_1 u_1}{m^2} \left(\boldsymbol{\lambda}_v - \frac{\lambda_\omega}{R} \hat{\boldsymbol{i}}_\tau \right) \cdot \hat{\boldsymbol{t}} \end{aligned} \tag{20}$$

Eventually, the following two-point boundary value problem is obtained:

$$\begin{aligned} \dot{\boldsymbol{r}} &= \boldsymbol{v} \\ \dot{\boldsymbol{v}} &= c_1 \frac{u_1^*}{m} \hat{\boldsymbol{t}}^* + \boldsymbol{g} \\ \dot{\theta} &= \omega \\ \dot{\omega} &= -c_1 \frac{u_1^*}{Rm} \hat{\boldsymbol{t}}^* \cdot \hat{\boldsymbol{i}}_\tau \\ \dot{m} &= -\frac{c_1}{c_2} u_1^* \\ \dot{\lambda}_\theta &= -\frac{\lambda_\omega}{R} c_1 \frac{u_1^*}{m} \hat{\boldsymbol{t}}^* \cdot \hat{\boldsymbol{i}}_\theta \\ \dot{\lambda}_\omega &= -\lambda_\theta \\ \dot{\lambda}_m &= \frac{c_1 u_1^*}{m^2} \left(\boldsymbol{\lambda}_v - \frac{\lambda_\omega}{R} \hat{\boldsymbol{i}}_\tau \right) \cdot \hat{\boldsymbol{t}}^* \end{aligned} \tag{21}$$

where

$$\boldsymbol{\lambda}_v = [\lambda_{vx0} + \lambda_{x0} t, \lambda_{vz0} + \lambda_{z0} t]$$

with boundary conditions $\boldsymbol{r}_0, \boldsymbol{v}_0, \theta_0$, and ω_0 at $t = 0$ and $\boldsymbol{r}_t = (0, 0)$, $\boldsymbol{v}_t = (0, 0)$, $\theta_t = 0$, $\omega_t = 0$, and $\lambda_{mt} = 0$ at $t = t_f$.

IV. Generating the Training and Validation Data

A dataset containing optimal trajectories is generated for each of the problems described in the previous section. Each optimal trajectory consists of a list of pairs ($\boldsymbol{x}^*, \boldsymbol{u}^*$), where $\boldsymbol{x}^*$ is the state and $\boldsymbol{u}^*$ is the corresponding optimal action. For each one of the problems, initial conditions are drawn fro an initialization area $\mathcal{A}$ so that, formally, $x_0 \in \mathcal{A}$. The definition of $\mathcal{A}$ for each model can be found in Table 2. One-hundred and thirty-five thousand optimal trajectories are generated for each problem; from each optimal trajectory, 100 state–control pairs are uniformly selected along the trajectory and inserted in the training data (thus containing 13,500,000 optimal state–action pairs).

The direct method used in previous work to compute the optimal trajectories [19] resulted in chattering problems due to numeric instabilities, whereas the profiles obtained via the indirect methods used in this paper are clean and accurately represent the optimal control without the need for extra regularization, as illustrated by Fig. 4. The direct method, in this case, produces a control with some chattering having a minor overall effect on the predicted optimal trajectory but posing a major problem if the state–action pairs have to be used in a training set.

For the single shooting method to converge, the initial guess for the costates $\boldsymbol{\lambda}_0$ is required to be close to the optimal solution. The quadratic control problems, given the smoothness of their solutions, can be solved from a random initial guess inside some relatively broad bounds. However, for the time and mass optimal control cases (both corresponding to $\alpha = 1$), a more precise guess is needed. Continuation methods are used to reduce the required computation time in the former and to be able to find the solutions in the latter, thus using as the initial guess for $\boldsymbol{\lambda}_0$ the solution to an optimal problem that is expected to be similar.

To find the TOC or MOC solution, the corresponding quadratic control problem is solved first; then, a homotopy path is followed by continuously increasing α from zero to one, resulting in smooth changes as shown by Fig. 5. Once a solution (QC, TOC, or MOC) with an associated costate vector $\boldsymbol{\lambda}_0$ is found for an initial state $\boldsymbol{x}_0$,

Table 2 Initialization areas $\mathcal{A}$ for the four landing problems

	x, m	z, m	v_x, m/s	v_z, m/s	θ, rad	ω, rad/s	m, kg
QUAD	[−5, 5]	[2.5, 20]	[−1, 1]	[−1, 1]	$[-(\pi/10), \pi/10]$	——	——
SSC	[−200, 200]	[500, 2,000]	[−10, 10]	[−30, 10]	——	——	[8,000, 12,000]
RWSC	[−200, 200]	[500, 2,000]	[−10, 10]	[−30, 10]	$[-(\pi/20), \pi/20]$	——	[8,000, 12,000]
TVR	[−10, 10]	[500, 2,000]	[−0.5, 0.5]	[−40, 0]	$[-10^{-3}, 10^{-3}]$	$[-10^{-4}, 10^{-4}]$	[8,000, 12,000]

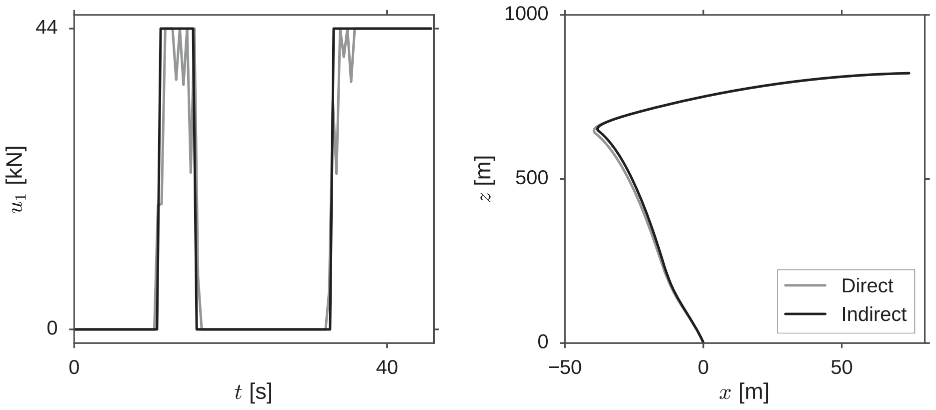

Fig. 4 RWSC-MOC problem solved using direct and indirect methods: thrust control (left), and trajectory (right).

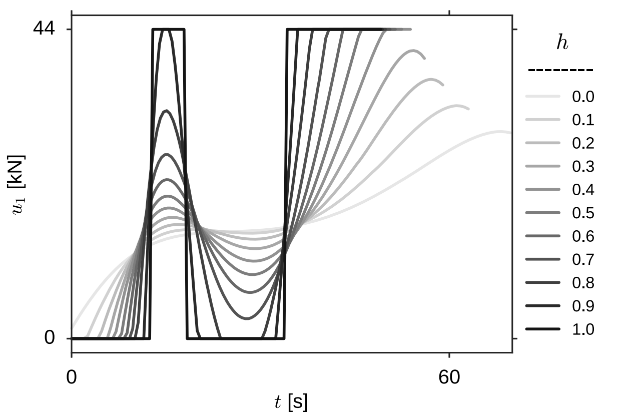

Fig. 5 Continuation from quadratic control ($h = 0$) to mass optimal control ($h = 1$) for the RWSC model.

a random walk in the n_x-dimensional state space is initiated. The costate vector $\boldsymbol{\lambda}_0$ will be used as the initial guess to find the optimal trajectory, starting from a point $\boldsymbol{x}_0'$ generated by perturbing each variable $x_i \in \boldsymbol{x}_0$ so that $x_i' = x_i + \delta$, with δ being a step size drawn from the uniform distribution $\delta \sim U(0, \eta r_i)$, where r_i corresponds to the range of the initialization area for the variable x_i and $\eta = 0.02$ determines the maximum step size. The random walk continues until reaching the bounds of $\mathcal{A}$ or 300 trajectories have been generated to then start a new random walk. Eventually, enough optimal trajectories are found to form the evaluation set. Separate and independent random walks are then started to create the validation set, up to when 15,000 optimal trajectories and 1,500,000 optimal state–action pairs are found. For the quadrotor, the simple, and the RWSC models, the initial state initializing each random walk is uniformly drawn at random in $\mathcal{A}$, resulting in a quite uniform coverage of the space $\mathcal{A}$. Figure 6 shows some of the random walks as well as the distribution of the initial states for the RWSC-MOC problem, showing that this method achieves a uniform coverage of $\mathcal{A}$.

Finding the optimal control for the TVR from arbitrary initial conditions in $\mathcal{A}$ (see Table 2) was revealed, instead, to be too challenging for the optimal control solver used here; thus, the random walk was initialized around a perfect vertical landing scenario of $x = 0$, $v_x = 0$, $\theta = 0$, and $\omega = 0$, with the remaining variables randomly initialized in $\mathcal{A}$. The random walk would then take care of continuing this trivial case into diverse initial conditions, but it would not be able to fill $\mathcal{A}$ uniformly. An example of the optimal trajectories thus computed and the histograms of their initial states are shown in Fig. 7. We can see how, due to the repetition of the initial state, the distribution of some variables (particularly, x, v_x, θ, and ω) approximates a Gaussian distribution around the nominal descent. In the same figure, the joint distribution of x and v_x is included, showing an inverse relation between these variables that corresponds to trajectories roughly pointing to the landing position in the horizontal axis. Trajectories with a high initial v_x pointing away from x are thus not in the training dataset, which is not an issue because those cases are not expected in real landing scenarios.

V. Learning the Optimal Control

Deep neural networks with a simple feedforward architecture are trained to learn the optimal state–action relation from the generated data. Equation (3) shows, mathematically, how the quantity to be learned is a function of the state alone (not its history), suggesting a feedforward deep architecture is indeed an appropriate choice. As each model has several control variables, we train separate networks for each of them. The following section describes in more detail the feedforward architectures considered and the training procedure.

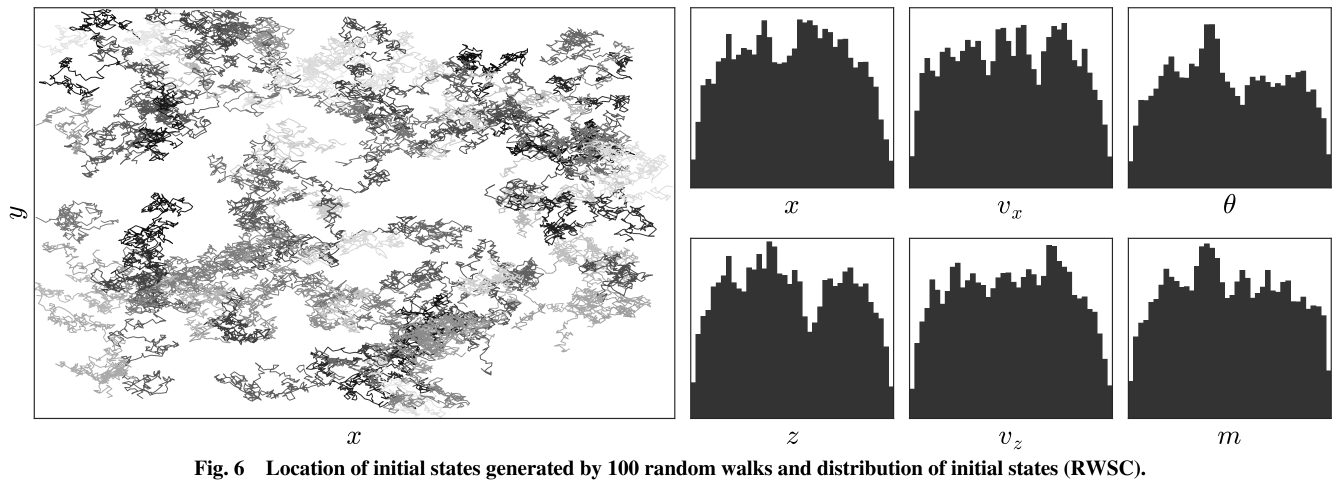

Fig. 6 Location of initial states generated by 100 random walks and distribution of initial states (RWSC).

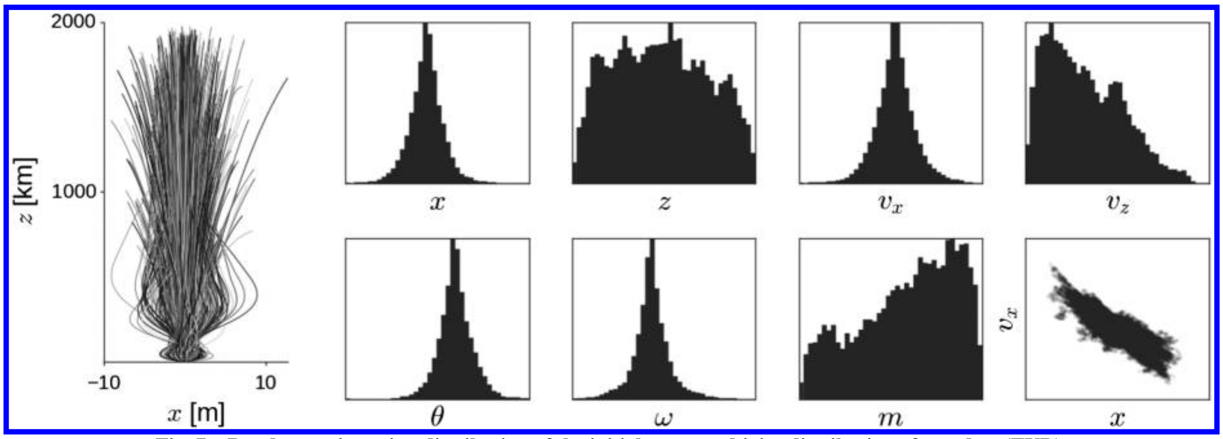

Fig. 7 Random trajectories, distribution of the initial states and joint distribution of x and v_x (TVR).

A. DNN Architecture

Architectures with different numbers of layers and units (neurons) per layer are considered. For comparison purposes, both shallow networks (one hidden and one output layer) and deep networks (several hidden layers) will be studied. The output o_{ij} of the unit i of layer j can be expressed as follows:

$$o_{ij} = g(\boldsymbol{w}_{ij}\boldsymbol{o}_{i-1} + b_{ij}) \tag{22}$$

where $\boldsymbol{w}_{ij}$ is the vector of weights, b_{ij} is the bias corresponding to that unit, $\boldsymbol{o}_{i-1}$ is the full output of the previous layer, and g is a nonlinear function.

The selection of the nonlinearities g (neuron types) has been identified as one of the most important factors of DNN architectures [25], and thus different functions are considered for the hidden and output layers. For the hidden layers, classical sigmoid units are compared to rectified linear units (ReLUs), which correspond to the activation function $\max(0, x)$. It has been pointed out that ReLUs have two main benefits when compared to sigmoid functions: they do not saturate, which avoids the units to stop learning in deep networks (the vanishing gradient problem); and the output of the units is frequently zero, which forces a sparse representation that is often addressed as a way of regularization that improves the generalization capabilities of the model [26]. The sigmoid function used for the comparison is the hyperbolic tangent, selected based on its better convergence as compared to the more common logistic function [27].

For the output layer, the following functions g are considered: the hyperbolic tangent function, the linear output $g(x) = x$, and a bounded linear output $g(x) = \max(m, \min(M, x))$, where m, M are the bounds of the control variable. The bounded linear output function here proposed (which is unusual in the machine learning community), and it tries to leverage the fact that the optimal control is saturated in some of the problems (bang–bang structure). All the inputs and outputs of the network are normalized by subtracting the mean and dividing by the standard deviation (of the training data). Additionally, when using the hyperbolic tangent in the last layer, the normalized outputs are scaled to the range of the function $[-1, 1]$. Figure 8 shows all the functions considered for the hidden and output layers.

B. Training

All networks are trained until convergence with stochastic gradient descent and a batch size of $b = 8$. At each iteration, the training process seeks to minimize the squared loss function

$$C = \sum_{i=0}^{b} \frac{1}{b} (\mathcal{N}(\boldsymbol{x}_i) - y(\boldsymbol{x}_i))^2$$

for the neural network output $\mathcal{N}(\boldsymbol{x}_i)$ and the optimal action $y(\boldsymbol{x}_i)$. Each weight w is then updated with a learning rate of $\eta = 0.001$ and a momentum with $\mu = 0.9$ [28]:

$$v_i \to v_i' = \mu v_i - \eta \frac{\partial C}{\partial w_i}$$
$$w_i \to w_i' = w_i + v_i'$$

After every epoch, the loss error is computed for a small portion of training data (5%) that is not used during training, and an early stopping criteria based on the idea of patience [29] is used to determine when to stop. In essence, all networks are trained for p more epochs after the last epoch that showed an improvement. The patience increment p is here set to five.

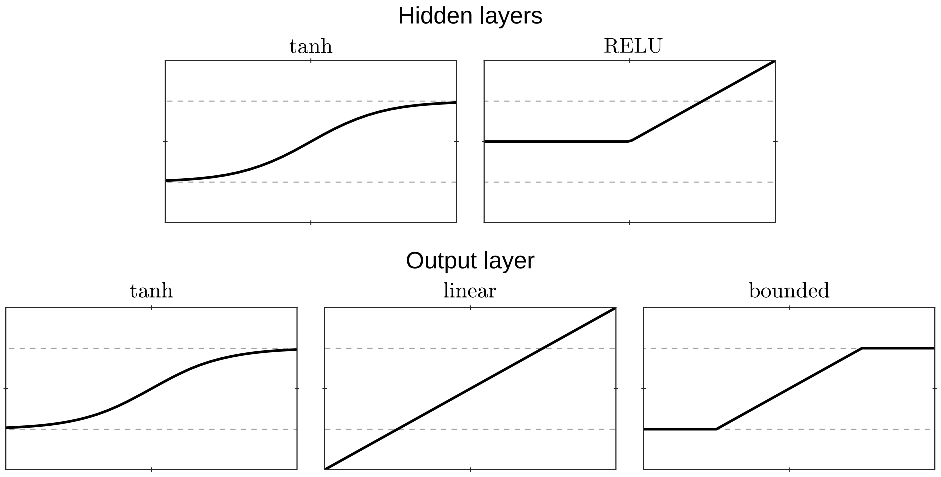

Fig. 8 Functions g considered for the hidden and output layers.

Table 3 **Range of the variables and value of the crash tolerance τ**

	Quadcopter			Simple spacecraft		Reaction wheel spacecraft			Thrust vectoring rocket			
	r, m	v, m/s	θ, deg	r, m	v, m/s	r, m	v, m/s	θ, deg	r, m	v, m/s	θ, deg	v_θ, deg/s
Range	20.62	17.34	100.84	2018.0	135.4	2022.3	141.6	80.2	1991.4	73.88	4.47	0.89
τ	0.1	0.1	1	10	0.7	10	0.7	1	10	0.7	0.1	0.01

Xavier's initialization method [30] is used to randomly set the initial weights. Although it was designed to improve the learning process for logistic units, it has been shown that this idea can also be beneficial for networks with ReLUs [31]. Each weight w_i is drawn from a uniform distribution $U[-a, a]$, with

$$a = \sqrt{\frac{\beta}{\text{fan}_{\text{in}} + \text{fan}_{\text{out}}}}$$

where $\beta = 12$ for the ReLUs and $\beta = 6$ for the linear and tanh units; and fan_{in} and fan_{out} are the number of units of the previous and following layers.

VI. Evaluation

Given any state of an optimal landing trajectory, the mean absolute error (MAE) is used to evaluate the difference between the optimal actions and the network predictions. This measure allows the comparison of the performance of different DNNs but does not provide an accurate measurement of how well the landing task is accomplished. Small errors could be propagated through the trajectory, resulting in suboptimal or failed landings. Errors could even potentially be corrected without impacting on the landing success. We thus need an additional evaluation scheme to judge how well a DNN has learnt the optimal control of the given scenarios. For this purpose, we introduce the DNN-driven trajectories: simulations of the landing dynamics as controlled by the DNN.

The DNN-driven trajectories are computed by numerical integration of the system dynamics $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) = \boldsymbol{f}(\boldsymbol{x}, \mathcal{N}(\boldsymbol{x}))$:

$$\boldsymbol{x}(t) = \int_0^t \boldsymbol{f}(\boldsymbol{x}(\tau), \mathcal{N}(\boldsymbol{x}(\tau)))\, \mathrm{d}\tau$$

The integration is carried out with a Livermore solver with maximum error of 10^{-12}, and a state is stored every 0.01 s.

The DNN-driven trajectory will reach the target point $\boldsymbol{x}_t$ with some error; thus, a tolerance region is defined around x_t and a trajectory is considered successful when it reaches this region. The tolerance τ for each problem is defined for the final value of each state variable (position τ_r, velocity τ_v, angle τ_θ, and angular velocity $\tau_{v\theta}$) and can be found in Table 3. These tolerance values have been set so that they do not exceed a 2% of the range of each variable, and they would not result in catastrophic landings for this scenarios.

The final state $\boldsymbol{x}_f$ of a DNN-driven trajectory is defined as the closest state to the target:

$$\boldsymbol{x}_f = \arg\min_{x_i} |\boldsymbol{x}_t - \boldsymbol{x}_i|$$

Given that the state includes variables with heterogeneous units, when computing $|\boldsymbol{x}_t - \boldsymbol{x}_i|$, we use τ_i as units (i.e., a final distance of τ_r m is deemed as equivalent to a final velocity of τ_v m/s).

To evaluate the performance of the DNNs in terms of optimality, it would seem obvious to compare the cost function $J(\boldsymbol{x}_0)$ evaluated at $\boldsymbol{x}_f$ along a DNN-driven trajectory to the optimal cost $J^*(\boldsymbol{x}_0)$. However, due to the introduced tolerances, the said cost function can result in being slightly better than the optimal cost. To get a fairer comparison, the optimal cost is also computed, stopping the optimal trajectory at the same distance to the target as $\boldsymbol{x}_f$.

The final evaluation of a DNN-driven trajectory is thus fully described by several quantities: the success rate (i.e., the likelihood to actually get to the target point within the set tolerances); the distance of $\boldsymbol{x}_f$ to the target value in terms of $\boldsymbol{r}$, $\boldsymbol{v}$, (according to the model) θ, and ω; and, in case that the DNN trajectory is deemed as successful, the optimality defined as the relative error of the cost function J with respect to the value of optimal control solution.

Figure 9 shows an example landing trajectory where the success bounds indicate the tolerance around x_t and the intersections between the end-state radius and the two trajectories show the points used to compare the DNN-driven trajectory to the optimal solution. Note that, in the figure, only the distance to the goal has been considered; but, in the computations, all variables are taken into account.

VII. Results

The mean absolute error computed for the states of the optimal trajectories in the validation set is used in order to compare the architectures of various artificial neural networks with different number of layers and different nonlinearities. To provide an analysis of the different architectures, four control variables representing the different control profiles (as illustrated in Fig. 1) are selected: 1) SSC-MOC u_2, which is smooth and continuous; 2) QUAD-QC u_2, which is smooth and continuous with saturated regions;

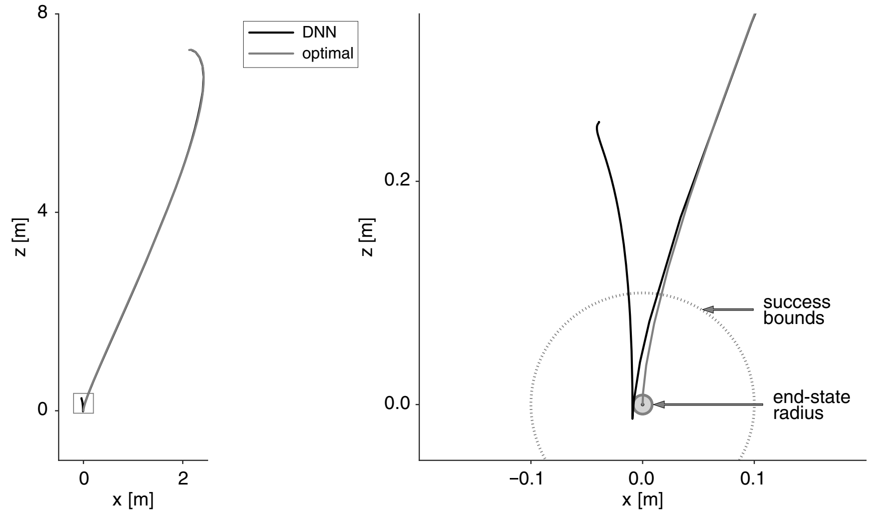

Fig. 9 Last part (right) of a QUAD-QC landing trajectory (left) depicting the success bounds and end-state radius.

3) RWSC-MOC u_1, which is bang–bang (always saturated); and 4) RWSC-MOC u_2, which is continuous with abundant plateaus and sharp transitions.

Then, in order to study the performance of the trajectories produced by the DNNs for each problem, trajectories from 1000 different initial states are simulated and evaluated.

A. Neuron Type: Nonlinearity Selection

Table 4 shows the results of the evaluation for the four selected control variables. The mean absolute error is computed for deep (five layers) networks with 32 units per layer and different neuron types for the hidden and output layers on the test set.

In the case of the two control variables with saturated regions (u_2 in QUAD-QC and u_1 in RWSC-MOC), the best results are provided by networks with a bounded linear output, closely followed by those obtained by networks with a tanh nonlinearity on the output layer. These two networks can easily reproduce the saturated regions: the bounded output by producing values higher or lower than the saturation level and the tanh by producing values as high or low as possible. Networks with linear outputs, however, correspond to the lowest performance on these cases, as expected, given that these network need to output the exact saturation value.

Regarding the neurons in the hidden layers, the DNNs with ReLUs outperform those with tanh in most cases. Better results are consistently obtained for ReLU in the first three models (u_2 in SSC-MOC, QUAD-QC, and u_1 in RWSC-MOC) when compared to networks with the same output but tanh hidden units. This difference is particularly large in the two models with bounded profiles, where the performance of the ReLUs is up two times better. An exception to this is the variable u_2 of RWSC-MOC, where networks with tanh units consistently perform better, although the difference, in this case, is small. Modeling the abundant plateaus present in this profile could be challenging for networks with ReLUs, whereas the flat areas of the tanh function could be an advantage for this problem. In any case, it is not possible to conclude that a nonlinearity is better across all domains, and the difference between them should be addressed for new models. The rest of the evaluation in this section is done using ReLUs for the hidden layers, and tanh or bounded linear functions for the output layers depending on the type of profile, with the later used only in cases where the landing profile includes saturation.

B. Depth of the Network

The mean absolute error is computed for models ranging from two to five hidden layers and different numbers of units per layer. The results are included in Table 5; note that networks with more layers always outperform shallower networks with a similar number of parameters. DNNs with five layers and just 16 units per layer are consistently better than shallow networks with 515 units in the hidden layer, even when the latter has almost four times more parameters. DNNs with five layers and 32 units will be used in the following sections.

C. DNN-Driven Trajectories

The DNN-driven trajectories are evaluated as described in Sec. VI. A summary of the results is included in Table 6. High success rates are achieved across all domains while obtaining a low relative error with respect to the value of the optimal trajectories.

Table 4 Effect of the neuron type: MAE of DNNs with three layers and 32 units/layer[a]

	$t-t$	$t-l$	$t-b$	$R-t$	$R-l$	$R-b$
SSC-MOC u_2	0.0257	0.0339	0.0382	0.0227[b]	0.0260	0.0259
QUAD-QC u_2	0.0697	0.0537	0.0668	0.0345	0.0371	0.0321[b]
RWSC-MOC u_1	651.5	887.9	646.6	458.3	671.4	304.8[b]
RWSC-MOC u_2	0.000934[b]	0.000981	0.000952	0.00114	0.00105	0.00108

[a]Hidden-output neuron type: t (tanh), R (ReLU), l (linear), b (bounded).
[b]Indicates the best result for each control variable.

Table 5 Mean absolute error of networks with different numbers of layers and units per layer

Layers/units	No. of weights[a]	QUAD-QC u_2	SSC-MOC u_2	RWSC-MOC u_1	RWSC-MOC u_2
2/256	1,793	0.0580	0.0357	752.4	0.00188
2/512	3,585	0.0577	0.0297	618.1	0.00150
3/16	385	0.0625	0.0341	677.4	0.00201
3/32	1,281	0.0524	0.0330	551.1	0.00138
3/64	4,609	0.0436	0.0257	497.2	0.00123
4/16	657	0.0503	0.0271	568.8	0.00161
4/32	2,337	0.0480	0.0250	592.0	0.00121
5/16	929	0.0475	0.0208[b]	474.8	0.00148
5/32	3,393	0.0321[b]	0.0227	304.8[b]	0.00114[b]

[a]Number of weights for a network with five inputs (QUAD and SSC); for RWSC (six inputs), the number of weights is increased by the number of units per layer.
[b]Indicates the best result for each control variable.

Table 6 Evaluation of the DNN-driven trajectories

		Distance to target				
	Success rate, %	r, m	v, m/s	θ, deg	ω, deg/s	Optimality, %
QUAD-QC	100.0	0.014	0.027	0.36	——	1.82
QUAD-TOC	100.0	0.016	0.028	0.48	——	1.12
SSC-QC	100.0	0.40	0.052	——	——	0.24
SSC-MOC	100.0	2.47	0.12	——	——	0.45
RWSC-QC	100.0	0.29	0.044	0.20	——	0.40
RWSC-MOC	98.3	2.90	0.073	0.31	——	0.72
TVR-QC	97.4	1.10	0.038	0.060	0.0075	0.38
TVR-MOC	94.7	1.95	0.094	0.012	0.0053	0.32

The quadcopter model and the SSC achieve a 100% success rate for the two objective functions in each case. In all cases, the average absolute distance to the target $D(\boldsymbol{x}_f)$ is way below the success bounds, showing that the state $\boldsymbol{x}_f$ reached by the network is close to the target state $\boldsymbol{x}_t$. The relative error of J, which is lower than 2% for the quadrotor problems and lower than 0.5% for the simple model, indicates that the profile followed by the network accurately represents the optimal control.

Similar results are obtained for the case of the RWSC model, with 100 and 98.3% success rates for quadratic and time optimal controls. A low distance to the target is obtained, and the relative error of J is below 1% for both objective functions. Figure 10 shows an example of the DNN predictions and the trajectory driven by the DNN. It is interesting to note that the predictions are accurate, even for the case of u_2 in the RWSC-MOC, where the numerous plateaus were expected to be difficult to approximate with the neural networks. Although this figure is only included as an example of the DNN predictions and DNN-driven trajectories, similar results are obtained for the other problems here considered.

Slightly lower success rates are achieved for the thrust vectoring (TVR) models (98.7 and 95.0% for the QC and MOC objectives), but the DNN is still able to reproduce the controls as illustrated by Fig. 11, where all the state and control variables of a TVR-MOC landing are included. Similarly to the previous cases, the relative error of J is below 0.5% for both objective functions.

Note that, although success rates of 100% are not achieved in all cases, this does not imply that the rest of the DNN-driven trajectories result in catastrophic failures, which do not happen in any of these cases. As the pinpoint landing criteria here defined are rather strict, trajectories with small deviations are considered as unsuccessful trajectories, although they would be valid from an engineering point of view. For example, in the case of the RWSC-MOC problem, a DNN-driven trajectory reaching the ground at a point 10 m from the defined pinpoint position or with an angle larger than 1 deg (an error admissible in any mission) is here considered a failure.

D. Behavior After Reaching $\boldsymbol{x}_f$

The ideal behavior of a DNN-driven trajectory after it reaches the final point x_f would be to start a hovering phase at the exact target position x_t. However, in all the considered cases, the structure of the optimal control makes it impossible to learn such a behavior for the

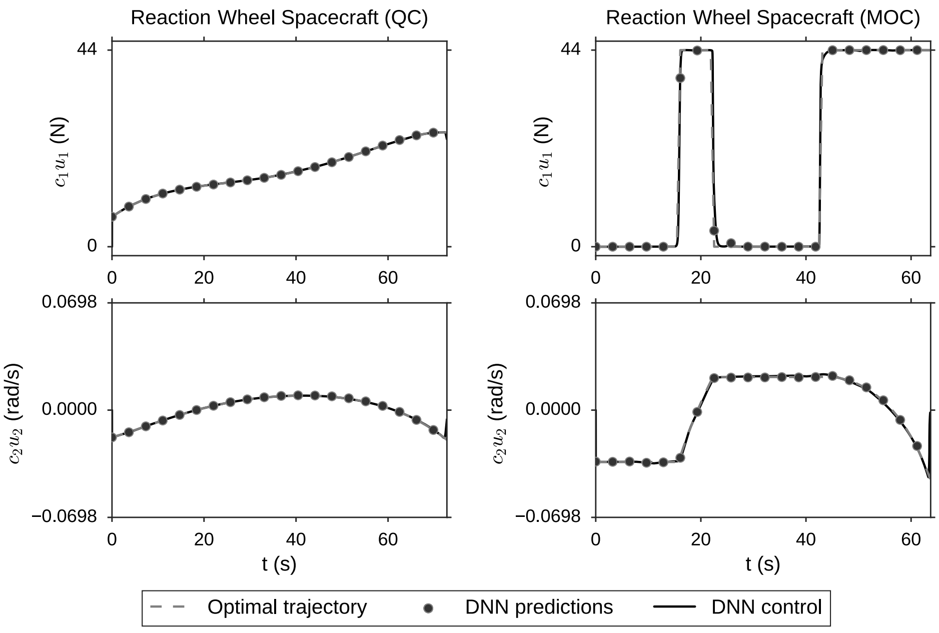

Fig. 10 DNN predictions during the optimal trajectory and during a DNN-driven trajectory (RWSC problems).

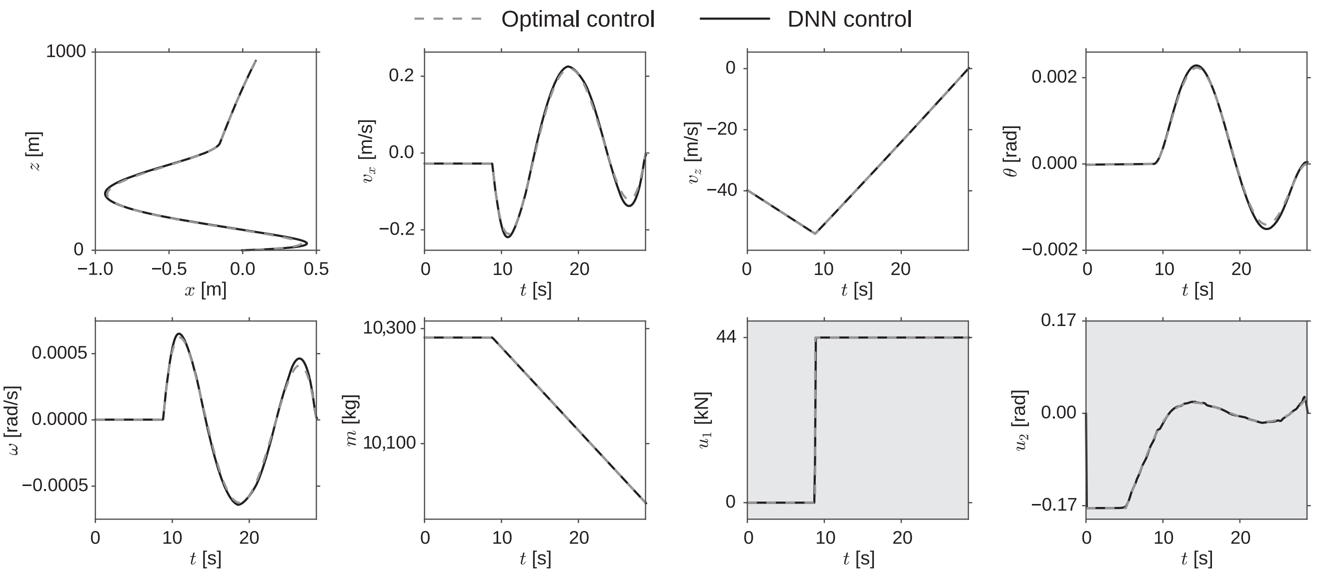

Fig. 11 State and control variables during a DNN-driven landing (TVR-MOC problem).

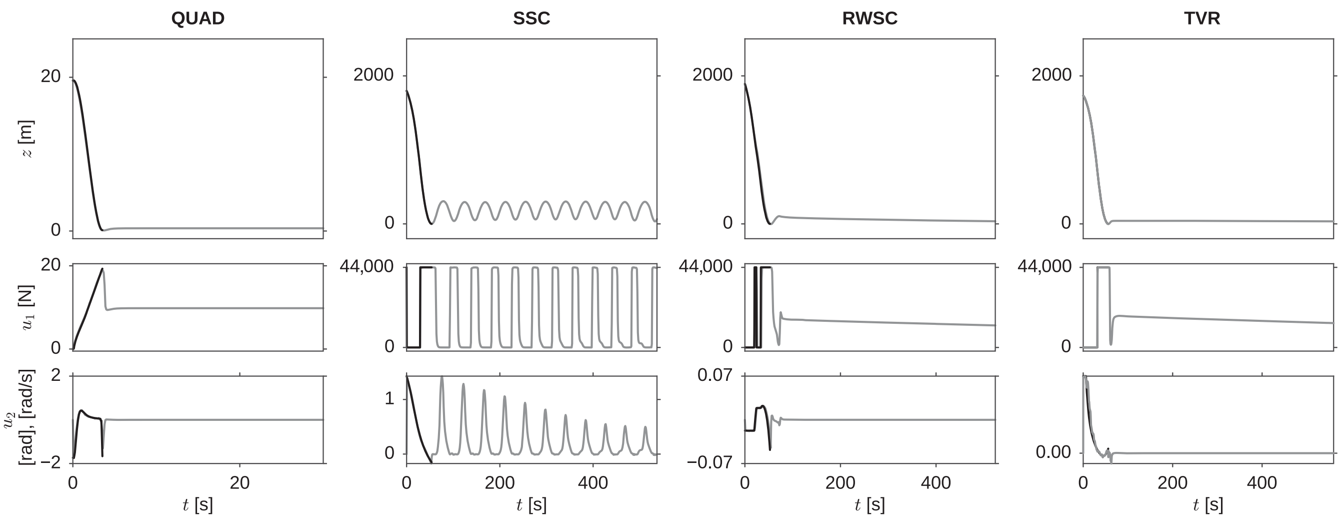

Fig. 12 Altitude and control variables of four models until x_f is reached (black) and afterward (gray).

DNNs. This is clear, for example, in the cases of time and mass optimal controls, where the target position is always reached with either maximum or minimum thrust, but a thrust exactly equal to mg is required for hovering at that position, which is a value that will never be present in the dataset, and is thus difficult to learn. Similarly, in the other models, the target state x_t is reached with different control values u that do not necessarily correspond to the value required for hovering, thus making it impossible for the network to learn how to reach and hover the exact x_t position. Bearing in mind the impossibility for a DNN to learn how to hover at x_t, we analyze the behavior of a DNN-driven trajectory after it reaches its final target point x_f.

Figure 12 shows DNN-driven trajectory behavior after reaching x_f for the quadcopter (quadratic control) and the other models (mass optimal control). Remarkably, in the case of the QUAD, RWSC, and TVR dynamics, a hovering behavior is observed at a position close to x_t. For the QUAD case, the numerical value of the thrust $c_1 u_1$ at this point acquires an approximate value of $9.81\ \mathrm{m/s^2}$, even for the case of time optimal control when only saturated values are present in the training data. For the TVR and RWSC cases, the numerical value of $c_1 u_1$ is, instead constantly decreasing in time because the mass of the rocket is also decreasing.

The behavior is different for the model where the pitch angle is directly controlled (SSC). In this case, there is no hovering phase but the spacecraft continuously oscillates above the target position. This is not unexpected because the optimal value of $u_2 = \theta$ at the final target point is not unique and the DNN can only learn its average value, which will likely be different from zero. As a consequence, the spacecraft thrust is not vertical around the target position, excluding the possibility to hover.

E. Generalization

It is of interest to study the behavior of DNN-driven trajectories when the initial conditions are outside of the initialization area $\mathcal{A}$. If the networks have learned an approximation to the solution of the HJB equations, we would expect them to still be able to represent the action for states that are not included in the training data. The results for 1000 DNN-driven trajectories starting from initial conditions drawn from two different extensions of $\mathcal{A}$ are shown, with each one excluding the previous area. The quadcopter and the simple and reaction wheel spacecraft (MOC) are considered. A summary of the results is included in Table 7.

In the case of the quadrotor, the trajectories are selected from an extension $\mathcal{A}_1$ of 5 m both in x and z and an extension A_2 of 10 m. Success rates of 84.4 and 75.0% are obtained for these extensions, although the average distance to the target is still close to zero, being 0.29 m, 0.21 m/s, and 1.63 deg for the furthest extension A_2. The optimality error in the extensions is 3.53 and 5.59%. Figure 13 shows some examples of these trajectories. Remarkably, the ability of the network to achieve the final target also extends for initial conditions lower than the landing position, as illustrated in the same figure, which requires thrusting to move upward, which is a condition not encountered during training.

For the spacecraft models, the x and y coordinates of the possible initial conditions are extended by 100 and 1000 m for $\mathcal{A}_1$ and 200 and 2000 m for $\mathcal{A}_2$. The SSC achieves success rates of 88.8 and 57.8%, but very high distances to the target are obtained because some trajectories result in catastrophic trajectories ending up far from x_t. The trajectories obtained for the reaction wheel model, although achieving lower success rates, have a lower average distance to the target because points close to the target are always reached. Figure 14 shows some examples of trajectories for these models.

A further, intriguing property of the DNN-driven trajectories is revealed when observing the RWSC and TVR model behaviors for a long time after they reach the target point. As previously noted, a hovering behavior is observed and persists in time long after the acquisition of the target state. The reduction of the mass due to the propellant loss is compensated by a commanded reduction of the thrust

Table 7 Evaluation of the DNN-driven trajectories outside of $\mathcal{A}$

	Success rate		Distance to target			Optimality error, %
			r, m	v, m/s	θ, deg	
QUAD-QC	$\mathcal{A}$	100%	0.014	0.027	0.36	1.82
	$\mathcal{A}_1$	84.4%	0.036	0.077	0.34	3.53
	$\mathcal{A}_2$	75.0%	0.29	0.21	1.63	5.59
SSC-MOC	$\mathcal{A}$	100%	0.40	0.052	——	0.24
	$\mathcal{A}_1$	88.8%	27.71	0.29	——	0.64
	$\mathcal{A}_2$	57.8%	521.44	1.741	——	1.31
RWSC-MOC	$\mathcal{A}$	98.3%	2.90	0.073	0.31	0.72
	$\mathcal{A}_1$	57.9%	9.34	0.28	0.35	1.13
	$\mathcal{A}_2$	34.9%	9.48	1.72	0.58	0.86

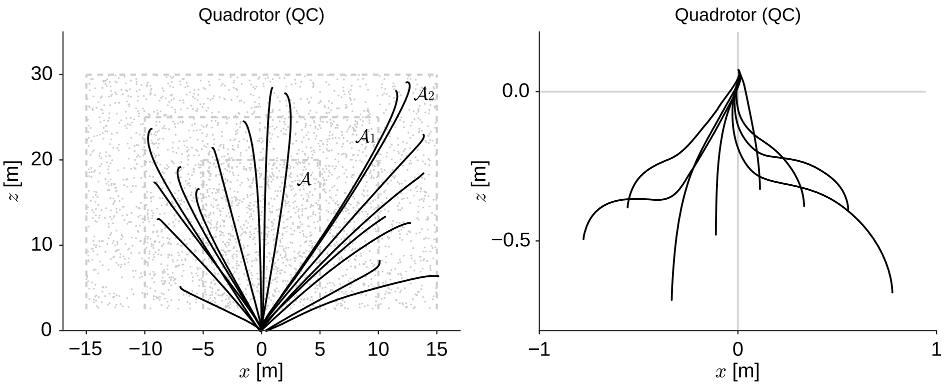

Fig. 13 Generalization of the DNN (QUAD-QC) to extensions of $\mathcal{A}$ and initial states with $z < 0$.

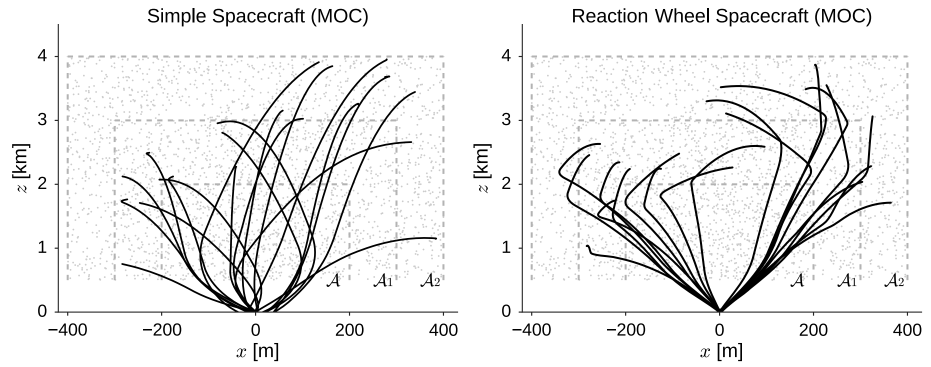

Fig. 14 Generalization of the DNN (SSC-MOC and RWSC-MOC) to extensions of $\mathcal{A}$.

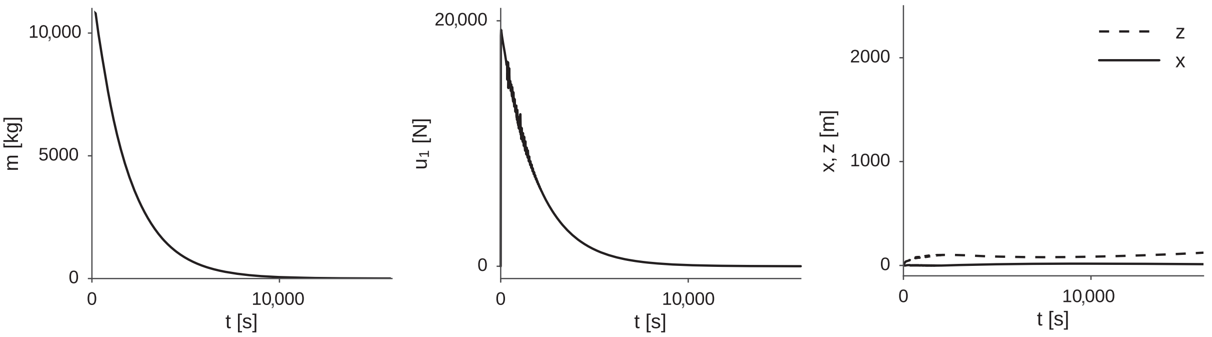

Fig. 15 RWSC-MOC landing maintains a stable after the final state t_f has been reached.

c_1u_1, as illustrated in Fig. 15. The spacecraft hovers close to the target state long after reaching x_f, even when the mass of the spacecraft is reduced to a fraction of the values found on the training data. It is thus clear that the network has learned in some way the problem dynamics and exploits it to maintain the spacecraft close to the target position.

F. Real-Time Implementation

In addition to the accurate control and the generalization capabilities shown for the DNNs, the main advantage of this method as compared to alternatives such as the direct and indirect methods mentioned in this paper is their low computational cost. Although training can be an expensive process, once the networks are trained, the control can be computed at very high rates. For instance, the output of a network with four hidden layers of 32 units is computed in 0.092 ms for an implementation using the Theano Python library and running on an Intel® Xeon® E5-2687 W on a 3.10 GHz CPU (the average of 100,000 iterations is reported). In this case, the control could be updated with a frequency of 10.86 MHz, which is much higher than needed for any of the problems we study here.

Additionally, the DNNs do not suffer of the convergence problems that appear in the direct and indirect methods to solve optimal control tasks. As a deterministic solution that does not require an initial guess, a control is always produced, regardless of the input values, making it a viable and safe solution for the landing cases here considered.

VIII. Conclusions

In this paper, it has been shown how deep neural networks can be trained to learn the optimal state feedback in a number of continuous-time deterministic, nonlinear systems of interest in the aerospace domain. The trained networks are not limited to predict the optimal state feedback from points within the subset of the state space used during training but are able to generalize to points well outside the training data, suggesting that the solution to Hamilton–Jacobi–Bellman equations is the underlying model being learned. The depth of the networks has a great influence on the obtained results, and it is remarkable that shallow networks, while trying to approximate the optimal state feedback, are unable to learn its complex structure satisfactorily. This work opens up the possibility to design real-time optimal control architectures for planetary landing using a DNN to directly drive the state–action selection. In this respect, the error introduced by the use of the trained DNN not only does not have a significant impact on the final cost function achieved but is also safe in terms of avoiding catastrophic failures for conditions that are far from nominal.

References

[1] LeCun, Y., Bengio, Y., and Hinton, G., "Deep Learning," *Nature*, Vol. 521, No. 7553, 2015, pp. 436–444. doi:10.1038/nature14539

[2] Schmidhuber, J., "Deep Learning in Neural Networks: An Overview," *Neural Networks*, Vol. 61, Jan. 2015, pp. 85–117. doi:10.1016/j.neunet.2014.09.003

[3] Silver, D., et al., "Mastering the Game of Go with Deep Neural Networks and Tree Search," *Nature*, Vol. 529, No. 7587, 2016, pp. 484–489. doi:10.1038/nature16961

[4] Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P., "Natural Language Processing (Almost) from Scratch," *Journal of Machine Learning Research*, Vol. 12, Aug. 2011, pp. 2493–2537.

[5] Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhutdinov, R., Zemel, R., and Bengio, Y., "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention," *Proceedings of the 32nd International Conference on Machine Learning*, Vol. 37, edited by F. Bach, and D. Blei, Proceedings of Machine Learning Research, PMLR, Lille, France, July 2015, pp. 2048–2057.

[6] Levine, S., "Exploring Deep and Recurrent Architectures for Optimal Control," *CoRR*, Vol. abs/1311.1761, 2013, http://arxiv.org/abs/1311.1761.

[7] Zhang, T., Kahn, G., Levine, S., and Abbeel, P., "Learning Deep Control Policies for Autonomous Aerial Vehicles with MPC-Guided Policy Search," *CoRR*, Vol. abs/1509.06791, 2015, http://arxiv.org/abs/1509.06791.

[8] Effati, S., and Pakdaman, M., "Optimal Control Problem via Neural Networks," *Neural Computing and Applications*, Vol. 23, Nos. 7–8, 2013, pp. 2093–2100. doi:10.1007/s00521-012-1156-2

[9] Xiong, Y., Derong, L., Ding, W., and Hongwen, M., "Constrained Online Optimal Control for Continuous-Time Nonlinear Systems Using Neuro-Dynamic Programming," *2014 33rd Chinese Control Conference (CCC)*, IEEE Publ., Piscataway, NJ, 2014, pp. 8717–8722. doi:10.1109/ChiCC.2014.6896465

[10] Medagam, P. V., and Pourboghrat, F., "Optimal Control of Nonlinear Systems Using RBF Neural Network and Adaptive Extended Kalman Filter," *Proceedings of the 2009 Conference on American Control Conference, ACC'09*, IEEE Publ., Piscataway, NJ, 2009, pp. 355–360. doi:10.1109/ACC.2009.5160105

[11] Todorov, E., "Optimality Principles in Sensorimotor Control," *Nature Neuroscience*, Vol. 7, No. 9, 2004, pp. 907–915. doi:10.1038/nn1309

[12] Lewis, F. L., and Abu-Khalaf, M., "A Hamilton-Jacobi Setup for Constrained Neural Network Control," *2003 IEEE International Symposium on Intelligent Control*, IEEE Publ., Piscataway, NJ, 2003, pp. 1–15. doi:10.1109/ISIC.2003.1253906

[13] Tassa, Y., and Erez, T., "Least Squares Solutions of the HJB Equation with Neural Network Value-Function Approximators," *IEEE Transactions on Neural Networks*, Vol. 18, No. 4, 2007, pp. 1031–1041. doi:10.1109/TNN.2007.899249

[14] Mordatch, I., Lowrey, K., Andrew, G., Popovic, Z., and Todorov, E. V., "Interactive Control of Diverse Complex Characters with Neural Networks," *Advances in Neural Information Processing Systems*, Curran Associates, Inc.,, Red Hook, NY, 2015, pp. 3114–3122.

[15] Berniker, M., and Kording, K. P., "Deep Networks for Motor Control Functions," *Frontiers in Computational Neuroscience*, Vol. 9, 2015, p. 32. doi:10.3389/fncom.2015.00032

[16] 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. doi:10.2514/1.G001480

[17] Scharf, D. P., Açıkmeşe, B., Dueri, D., Benito, J., and Casoliva, J., "Implementation and Experimental Demonstration of Onboard Powered Descent Guidance," *Journal of Guidance, Control, and Dynamics*, Vol. 40, No. 2, 2017, pp. 213–229. doi:10.2514/1.G000399

[18] 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, 2007, pp. 1353–1366. doi:10.2514/1.27553

[19] Sanchez-Sanchez, C., Izzo, D., and Hennes, D., "Optimal Real-Time Landing Using Deep Networks," *Proceedings of the 6th International Conference on Astrodynamics Tools and Techniques, ICATT*, Vol. 12, European Space Agency, The Netherlands, Aug. 2016, pp. 2493–2537.

[20] Pontryagin, L. S., *Mathematical Theory of Optimal Processes*, CRC Press, Boca Raton, FL, 1987, pp. 10–72, Chaps. 1, 2.

[21] Bardi, M., and Capuzzo-Dolcetta, I., *Optimal Control and Viscosity Solutions of Hamilton–Jacobi–Bellman Equations*, Springer Science and Business Media, New York, 2008, pp. 25–96. doi:10.1007/978-0-8176-4755-1

[22] Beard, R. W., Saridis, G. N., and Wen, J. T., "Galerkin Approximations of the Generalized Hamilton–Jacobi–Bellman Equation," *Automatica*, Vol. 33, No. 12, 1997, pp. 2159–2177. doi:10.1016/S0005-1098(97)00128-3

[23] Izzo, D., and de Croon, G., "Nonlinear Model Predictive Control Applied to Vision-Based Spacecraft Landing," *Proceedings of the EuroGNC 2013, 2nd CEAS Specialist Conference on Guidance, Navigation & Control*, Delft Univ. of Technology, Delft, The Netherlands, 2013, pp. 91–107.

[24] Hehn, M., Ritz, R., and D'Andrea, R., "Performance Benchmarking of Quadrotor Systems Using Time-Optimal Control," *Autonomous Robots*, Vol. 33, Nos. 1–2, 2012, pp. 69–88. doi:10.1007/s10514-012-9282-3

[25] Jarrett, K., Kavukcuoglu, K., Ranzato, M., and LeCun, Y., "What is the Best Multi-Stage Architecture for Object Recognition?" *2009 IEEE 12th International Conference on Computer Vision*, IEEE Publ., Piscataway, NJ, 2009, pp. 2146–2153. doi:10.1109/ICCV.2009.5459469

[26] Glorot, X., Bordes, A., and Bengio, Y., "Deep Sparse Rectifier Neural Networks," *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, edited by G. Gordon, D. Dunson, and M. Dudík, Vol. 15, Proceedings of Machine Learning Research, PMLR, Fort Lauderdale, FL, 2011, pp. 315–323, http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf.

[27] LeCun, Y. A., Bottou, L., Orr, G. B., and Müller, K.-R., "Efficient BackProp," *Neural Networks: Tricks of the Trade*, 2nd ed., Springer, Berlin, 2012, pp. 9–48. doi:10.1007/978-3-642-35289-8_3

[28] Sutskever, I., Martens, J., Dahl, G., and Hinton, G., "On the Importance of Initialization and Momentum in Deep Learning," *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, edited by S. Dasgupta, and D. McAllester, Vol. 28, No. 3, Proceedings of Machine Learning Research, PMLR, Atlanta, GA, 2013, pp. 1139–1147, http://proceedings.mlr.press/v28/sutskever13.html.

[29] Bengio, Y., "Practical Recommendations for Gradient-Based Training of Deep Architectures," *Neural Networks: Tricks of the Trade*, Springer, New York, 2012, pp. 437–478, Chap. 26. doi:10.1007/978-3-642-35289-8

[30] Glorot, X., and Bengio, Y., "Understanding the Difficulty of Training Deep Feedforward Neural Networks," *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, edited by Y. W. Teh, and M. Titterington, Vol. 9, Proceedings of Machine Learning Research, PMLR, Sardinia, Italy, May 2010, pp. 249–256, http://proceedings.mlr.press/v9/glorot10a.html.

[31] He, K., Zhang, X., Ren, S., and Sun, J., "Delving Deep into Rectifiers: Surpassing Human-Level Performance on Imagenet Classification," *Proceedings of the IEEE International Conference on Computer Vision*, IEEE Publ., Piscataway, NJ, 2015, pp. 1026–1034. doi:10.1109/ICCV.2015.123

This article has been cited by:

1. Yuheng Guo, Xiang Li, Houjun Zhang, Ming Cai, Feng He. Data-Driven Method for Impact Time Control Based on Proportional Navigation Guidance. *Journal of Guidance, Control, and Dynamics*, ahead of print1-12. [Abstract] [Full Text] [PDF] [PDF Plus]
2. James D. Biggs, Hugo Fournier. 2020. Neural-Network-Based Optimal Attitude Control Using Four Impulsive Thrusters. *Journal of Guidance, Control, and Dynamics* **43**:2, 299-309. [Abstract] [Full Text] [PDF] [PDF Plus]
3. Andrea Scorsoglio, Roberto Furfaro, Richard Linares, Brian Gaudet. Image-based Deep Reinforcement Learning for Autonomous Lunar Landing . [Abstract] [PDF] [PDF Plus]
4. Sixiong You, Changhuang Wan, Ran Dai, Ping Lu, Jeremy R. Rea. Learning-based Optimal Control for Planetary Entry, Powered Descent and Landing Guidance . [Abstract] [PDF] [PDF Plus]
5. Kshitij Mall, Ehsan Taheri. Unified Trigonometrization Method for Solving Optimal Control Problems in Atmospheric Flight Mechanics . [Abstract] [PDF] [PDF Plus]
6. Yang Shi, Zhenbo Wang. A Deep Learning-Based Approach to Real-Time Trajectory Optimization for Hypersonic Vehicles . [Abstract] [PDF] [PDF Plus]
7. Kirk Hovell, Steve Ulrich. On Deep Reinforcement Learning for Spacecraft Guidance . [Abstract] [PDF] [PDF Plus]
8. Sungyung Lim, Matthew Stoeckle, Brett J. Streetman, Matthew Neave. Markov Neural Network For Guidance, Navigation and Control . [Abstract] [PDF] [PDF Plus]
9. Yue-he Zhu, Ya-Zhong Luo. 2019. Fast Evaluation of Low-Thrust Transfers via Multilayer Perceptions. *Journal of Guidance, Control, and Dynamics* **42**:12, 2627-2637. [Abstract] [Full Text] [PDF] [PDF Plus]