OPTIMAL REAL-TIME LANDING USING DEEP NETWORKS

Carlos Sánchez-Sánchez[1], *Dario Izzo*[1], *Daniel Hennes*[2]

[1] Advanced Concepts Team - European Space Agency,
[2] Robotics Innovation Center - German Research Center for Artificial Intelligence

ABSTRACT

Optimal trajectories for spacecraft guidance, be it during orbital transfers or landing sequences are often pre-computed on ground and used as nominal desired solutions later tracked by a secondary control system. Linearization of the dynamics around such nominal profiles allows to cancel the error during the actual navigation phase when the trajectory is executed.

In this study, instead, we assess the possibility of having the optimal guidance profile be represented on-board by a deep artificial neural network trained, using supervised learning, to represent the optimal control structure. We show how the deep network is able to learn the structure of the optimal state-feedback outside of the training data and with great precision. We apply our method to different interesting optimal control problems, a multicopter time and power optimal pinpoint landing control problem and two different mass optimal spacecraft landing problems. In all cases, the deep network is able to safely learn the optimal state-feedback, also outside of the training data, making it a viable candidate for the implementation of a reactive real-time optimal control architecture.

Index Terms— real-time landing, optimal control, neural networks, deep neural networks

1. INTRODUCTION

Optimal control for spacecraft landing typically requires expensive computations which make it infeasible for on-board real-time applications. For this reason, although an optimal reactive controller is desirable, current solutions often consider a precomputed optimal profile with an additional control system that corrects for deviations during descent [1].

In the past, artificial neural networks (ANN) have been frequently proposed as an efficient way to compute optimal control in different domains. Due to the decreasing cost of computational resources and the latest advances in research to train neural networks with many hidden layers [2] we can see a renewed interest in the field, and in particular in deep neural networks (DNN), in the context of optimal control.

While the representation capabilities of DNNs makes them particularly appropriate for perception related tasks such as image and speech recognition, it has been more recently pointed out that control problems also benefit from these models [3, 4]. However, work in this direction is mostly limited to discrete-time problems and investigated the implications on dynamic programming tasks. Instead, the use of artificial neural networks for the optimal control of continuous time, non-linear systems received less attention. Successful applications have so far been limited to simple domains (e.g., linear systems often appearing in case studies) or to unbounded control [5, 6, 7]. As a consequence, their possible use in robotics and engineering is rather restricted. Contributions to the solution of both the Hamilton-Jacobi-Belmann (HJB) equations and the two point boundary value problem resulting from Pontryagin's optimal control theory also showed possible uses of ANNs in this context [8]. On the one hand, several methods were proposed for the approximation of the value function $v(t, \mathbf{x})$ by means of ANN architectures [9, 10, 7]. On the other hand, ANNs have been proposed and studied to provide a trial solution to the states, the co-states and to the controls so that their weights can be trained to make sure the assembled trial Hamiltonian respects Pontryagin's conditions [5]. Clearly, in this last case, the networks have to be retrained for each initial condition. More recently, a deep neural network, in the form of a stacked auto-encoder was shown [11] to be able to learn an accurate temporal profile of the optimal control and state in a point-to-point reach of a continuous time, non-linear limb model.

In this paper we successfully use DNNs to represent the solution to the Hamilton-Jacobi-Belmann policy equation for sevral deterministic, non-linear, continuous time systems. Our work shows that trained DNNs are suitable to achieve real-time optimal control capabilities in complex tasks in the domain of optimal landing, providing the spacecraft with a reactive control system able to safely achieve its goals.

2. BACKGROUND

We consider deterministic systems defined by the dynamics $\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t))$, where $\mathbf{x}(t) : \mathbb{R} \to \mathbb{R}^{n_x}$ and $\mathbf{u}(t) : \mathbb{R} \to \mathcal{U} \subset \mathbb{R}^{n_u}$. We address the fundamental problem of finding an admissible control policy $\mathbf{u}(t)$ able to steer the system from any $\mathbf{x}_0 = \mathbf{x}(0)$ to some target goal in a subset $\mathcal{S} \subset \mathbb{R}^{n_x}$. At t_f the system will be in its final state $\mathbf{x}_f = \mathbf{x}(t_f) \in \mathcal{S}$ having minimized the following cost function:

$$J(\mathbf{x}(\cdot), \mathbf{u}(\cdot)) = \int_0^{t_f} \mathcal{L}(\mathbf{x}(t), \mathbf{u}(t))dt + h(\mathbf{x}(t_f))$$

The value function, defined as:

$$v(\mathbf{x}) = \min_{\mathbf{u}} J(\mathbf{x}(\cdot), \mathbf{u}(\cdot)) \qquad (1)$$

represents the minimal cost to reach the goal, starting from $\mathbf{x}$. Equivalently, the value function can be introduced as the solution to the partial differential equation:

$$\min_{\mathbf{u}} \{\mathcal{L}(\mathbf{x}, \mathbf{u}) + f(\mathbf{x}, \mathbf{u}) \cdot \nabla v(\mathbf{x})\} = 0 \qquad (2)$$

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

$$\mathbf{u}^*(\mathbf{x}) = \text{argmin}_{\mathbf{u}} \{\mathcal{L}(\mathbf{x}, \mathbf{u}) + f(\mathbf{x}, \mathbf{u}) \cdot \nabla v(\mathbf{x})\} \qquad (3)$$

Equations 2 and 3 are the Hamilton-Jacobi-Bellman (HJB) equations for the optimal control problem here considered. They are a set of extremely challenging partial differential equations (PDEs) whose solution, pursued in the "viscosity" sense, is the solution to the original optimal control problem [12]. The HJB equations show the existence of an optimal state-feedback $\mathbf{u}^*(\mathbf{x})$ and provide a way to compute it once the value function is known. Numerical approaches to solving HJB equation, thus, often rely on parametric approximations of the value function, e.g. using the Galerkin method [13], which have also included ANNs in the past [10]. Unlike in previous work, we use deep neural networks (DNNs) to learn directly the optimal state-feedback $\mathbf{u}^*(\mathbf{x})$ thus obtaining, indirectly, also a representation of the value function $v(\mathbf{x}) = J(\mathbf{x}^*, \mathbf{u}^*)$, while avoiding to make use of the network gradients when converting from value function to the optimal policy. The DNN weights are trained using supervised learning on precomputed values of the optimal state-feedback for a set of initial states. Eventually, the trained DNN represents directly the optimal state-feedback and can be thus used, for example, in a non-linear model predictive control architecture [14] to achieve real-time optimal control capabilities.

3. OPTIMAL CONTROL PROBLEMS

Three different domains of increasing complexity and dimension are considered: the pinpoint landing of a multicopter and the landing for two different models of spacecraft. Additionally, we consider different cost functions, resulting in different types of control profiles, i.e. smooth, saturated and bang-bang. In each problem we introduce an initialization area $\mathcal{A} \subset \mathbb{R}^{n_x}$: only initial conditions in $\mathcal{A}$ will be considered for the purpose of the training data generation.

3.1. Pinpoint landing (multicopter model)

The first domain we consider is the multicopter pinpoint landing ($n_x = 5$, $n_u = 2$). The state is described by $\mathbf{x} = [x, z, \theta, v_x, v_z]$ and the system dynamics by the following ODEs [15].

$$\begin{array}{ll} \dot{x} = v_x & \dot{v}_x = u_1 \sin(\theta) \\ \dot{z} = v_z & \dot{v}_z = u_1 \cos(\theta) - g \\ & \dot{\theta} = u_2 \end{array} \qquad (4)$$

We use a multicopter mass of $m = 1$ [kg] and $g = 9.81$ [m/s^2]. The two control actions are the thrust $u_1 \in [0, 20]$ [N], and the pitch rate $u_2 \in [-2, 2]$ [rad/s]. The goal is to reach the landing position $x_f = 0$, $z_f = 0.1$, $v_{xf} = 0$, $v_{zf} = -0.1$, $\theta_f = 0$. The initialization area $\mathcal{A}$ is $x_0 \in [-5, 5]$ [m], $z_0 \in [5, 20]$ [m], $v_{x0} \in [-1, 1]$ [m/s], $v_{z0} \in [-1, 1]$ [m/s], $\theta_0 \in \left[-\frac{\pi}{10}, \frac{\pi}{10}\right]$ [rad].

Two different cost functions are studied: quadratic control, where we seek to minimize the power consumption $J_p = \int_0^{t_f} (\alpha_1 u_1^2 + \alpha_2 u_2^2) dt$ with $\alpha_1 = \alpha_2 = 1$, and time $J_t = t_f$.

3.2. Landing (spacecraft models)

The other two domains correspond to spacecraft landing under a uniform gravity field. Two mass-varying systems are considered, the first one controlled by a thruster and a momentum exchange wheel (spacecraft model 1) with ($n_x = 6$, $n_u = 2$) and the second one by one main thruster and two lateral engines for pitch control (spacecraft model 2) with ($n_x = 7, n_u = 3$) [14].

The state in the case of the spacecraft model 1 is $\mathbf{x} = [x, v_x, z, v_z, \theta, m]$ and the system is described by the following set of ODEs.

$$\begin{array}{ll} \dot{x} = v_x & \dot{v}_x = u_1 \sin(\theta)/m \\ \dot{z} = v_z & \dot{v}_z = u_1 \cos(\theta)/m - gm \\ \dot{\theta} = u_2 & \dot{m} = -u_1/(Isp \cdot g_0) \end{array} \qquad (5)$$

We consider the Moon gravity $g = 1.62$ [m/s^2] and a main engine with specific impulse $I_{sp} = 311$ [s]. The controls are the thrust $u_1 \in [0, 45760]$ [N] and the pitch rate $u_2 \in [-0.0698, 0.0698]$ [rad/s].

For model 2, the state is $\mathbf{x} = [x, v_x, z, v_z, \theta, v_\theta, m]$ and, similarly to the previous one, the system is described by the following ODEs.

$$\begin{array}{ll} \dot{x} = v_x & \dot{v}_x = (u_1 + u_2 + u_3) \sin(\theta)/m \\ \dot{z} = v_z & \dot{v}_z = (u_1 + u_2 + u_3) \cos(\theta)/m - g \\ \dot{\theta} = v_\theta & \dot{v}_\theta = (u_3 - u_2)/m/R \\ & \dot{m} = -(u_1 + u_2 + u_3)/(Isp \cdot g_0) \end{array} \qquad (6)$$

In this case, instead of the pitch range we control two lateral thrusters $u_2, u_3 \in [0, 880]$ [N] separated by a distance $R = 3$ [m].

In both cases the target is: $z_f = 10$ [m], $v_{xf} = 0$ [m/s], $v_{zf} = -0.1$ [m/s]. No target is defined for x as we consider a free landing scenario, hence the control action does not depend on x. The initialization area $\mathcal{A}$ is: $z_0 \in [500, 2000]$ [m], $m_0 \in [8000, 12000]$ [kg], $v_{x0} \in [-100, 100]$ [m/s], $v_{z0} \in [-30, 10]$ [m/s] and $\theta_0 \in \left[-\frac{\pi}{20}, \frac{\pi}{20}\right]$ [rad].

We study the minimum mass problem, the cost function being $J_m = \int_{t_0}^{t_f} u_1/(Isp \cdot g) = m_f - m_0$. In the case of model 1, considering that the optimal action u_2 is not well defined when $u_1 = 0$ (multiple equivalent actions exist), we add the term αu_2^2 thus minimizing the cost function $J'_m = J_m + \int_{t_0}^{t_f} \alpha(u_2^2)dt$ with $\alpha = 10$.

The optimal control solutions of these three domains represent different classes of control profiles. The quadratic control problems result in continuous functions saturated at their bounds. The time-optimal and mass-optimal cost functions, instead, result in discontinuous bang-off-bang control structures for the thrust. In the case of model 1, an added complexity comes from the change in behaviour of the pitch control during the thrust on and off phases: constant until the switching point (the term $\int_{t_0}^{t_f} \alpha(u_2^2)dt$ is active) and smooth and continuous afterwards. In the case of model 2 the three thrusters (main and laterals) have a bang-off-bang structure in theory. In practice we will discuss how in our data some noise was introduced by the use of a direct method, which accounts for some performance loss in this most complex case.

3.3. Training data generation

For each of the problems described above we generate a dataset containing pairs $(\mathbf{x}^*, \mathbf{u}^*)$, where $\mathbf{x}^*$ is a state and $\mathbf{u}^*$ is the corresponding optimal action. The dataset is generated solving the non-linear programming problem (NLP) resulting from using the Hermite-Simpson transcription method [16], a direct method to tackle optimal control problems. The solution is provided by a sequential quadratic programming NLP solver, namely SNOPT [17].

For each of the considered problems we generate $150,000$ different trajectories starting from a point randomly sampled from the initialization area $\mathcal{A}$. Of each trajectory we store 60 (multicopter) or 100 (spacecraft) state-control pairs, resulting in $9,000,000$-$15,000,000$ training samples. We use 90% of the trajectories to train the model while the rest is used for validation.

In the landing cases, due to known problems caused by the use of the direct method in connection with saturated controls (i.e. bang-bang), we observe small chattering effects that have a negative impact on the learning process. We address this problem by adding a regularization term to the objective function when computing the NLP solution. For the time optimal quadrotor and the spacecraft model 1, the added term corresponds to βJ_p, where J_p is the quadratic power goal function. If β is chosen sufficiently small, the effect of this term on the final value of $J(\mathbf{x}(\cdot))$ is negligible, but helps the sequential quadratic programming solver to converge towards a solution without chattering. We use $\alpha_1 = \alpha_2 = 1, \beta = 0.001$ for the multicopter time-optimal problem and $\alpha_1 = 1$, $\alpha_2 = 0$, $\beta = 10^{-10}$ for the spacecraft mass-optimal problem.

Given the complexity of the spacecraft model 2, regularization as described is not effective and instead we use the more aggressive term $\beta \sum_i \sum_t \alpha_i(u_i(t+1) - u_i(t))$ with $[\beta = 10^{-5}, \alpha = [0.02, 1, 1]$, where the different α_i values are used because of the different ranges of the main thrust and the lateral ones. This removes most of the chattering effects, but some of them still appear, and thus we remove those trajectories that switch from one extreme value to the other one more than five times. Note that generating the trajectories using an indirect method, instead of a direct one, would remove the need of these regularization terms that may interfere with the main objectives, what is currently being researched.

4. STATE-FEEDBACK APPROXIMATION

The data generated as described above is used to train the feed-forward neural networks. Given that each model has several control variables, we train separate networks for each one of them (i.e. thrust or pitch) .

4.1. DNN architecture

We consider both models with only one (shallow) and several hidden layers (deep). The selection of the non-linearities used in the hidden units has been identified as one of the most important factors of DNN architectures [18]. We compare the classical sigmoid units to rectified linear units (ReLUs), which correspond to the activation function $max(0, x)$. It has been pointed out that ReLu units have two main benefits when compared to sigmoid functions: they do not saturate, which avoids the units to stop learning after reaching a point (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 [19]. The sigmoid function used for the comparison is the hyperbolic tangent, selected based on their convergence behaviour compared to the standard logistic function [20]. We consistently obtain higher performance with ReLus (see Table 1) and thus they are used in the hidden units of our DNNs.

The output units are $tanh$ units, which always provided a better result when we compare them to linear units. All the inputs and outputs are normalized by subtracting the mean and dividing by the standard deviation. The normalized outputs are then further scaled to the range [-1,1] to make sure they can all be represented by the output of the $tanh$ function.

	Architecture	Train (u_1)	Val. (u_1)
u_1	tanh - tanh	955.53	963.61
	ReLu - tanh	918.90	936.21
	ReLu - linear	973.20	978.35
u_2	tanh - tanh	0.00283	0.00284
	ReLu - tanh	0.00250	0.00252
	ReLu - linear	0.00257	0.00259

Table 1. Performance comparison for different non-linearities (hidden - output) on the spacecraft model 1 trained for mass optimal landing.

4.2. Training

All networks are trained until convergence with stochastic gradient descent (SGD) and a batch size of 8. After every epoch the loss error is computed for the evaluation set and the learning process is stopped if there are more than 3 epochs with no increments affecting, at least, the third significant digit. We use Xavier's initialization method [21] 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 ReLu units [22]. In our case, each weight w_i is drawn from a uniform distribution $U[-a, a]$, with $a = \sqrt{\frac{12}{\text{fan}_{\text{in}}+\text{fan}_{\text{out}}}}$, being fan$_{\text{in}}$, fan$_{\text{out}}$ the number of units of the previous and following layers.

The training process seeks to minimize the squared loss function $C = \sum_{i=0}^{8} \frac{1}{8}(\mathcal{N}(\mathbf{x_i}) - y(\mathbf{x_i}))^2$ for the neural network output $\mathcal{N}(\mathbf{x_i})$ and the optimal action $y(\mathbf{x_i})$. The weights w_i are updated with a learning rate $\eta = 0.001$ and momentum with $\mu = 0.9$ [23]:

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

$$w_i \to w_i' = w_i + v_i'$$

4.3. DNN driven trajectories

Once trained, the DNNs can be used to predict the control given any state $\mathbf{x}$. This allows us to also compute the full trajectory by numerical integration of the system dynamics $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}^*) = \mathbf{f}(\mathbf{x}, \mathcal{N}(\mathbf{x}))$:

$$\mathbf{x}(t) = \int_0^t \mathbf{f}(\mathbf{x}, \mathcal{N}(\mathbf{x}))ds$$

The integration is stopped when the goal state is reached within some tolerance or a failure is detected (i.e., too much time has elapsed or a crash $z < 0$ is detected). For the multicopter the tolerance to the goal position is set to 0.1 [m] and for the final velocity to 0.1 [m/s]. For the spacecraft landing case, a velocity tolerance of 1 [m/s] has to be achieved after passing the 10 [m] goal without crashing.

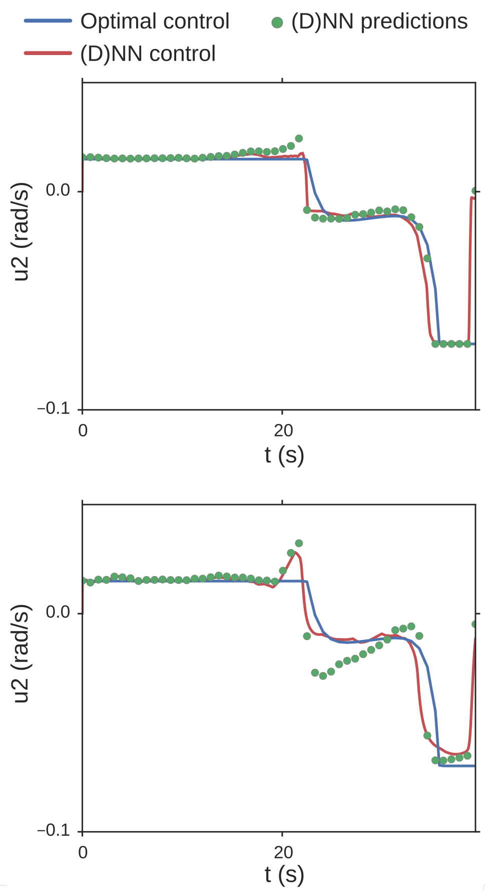

Fig. 1. u_2 control during a challenging mass optimal spacecraft (model 1) landing. The optimal control, the (D)NN predictions for each state along the optimal trajectory and the full trajectory driven by the (D)NN are shown. Top: deep network (4 hidden layers, 64 units, 12992 parameters). Bottom: shallow network (1 hidden layer, 2048 units, 16384 parameters). The deep network achieves safe landing conditions, while the shallow network results in a catastrophic failure.

5. RESULTS

We are interested both in determining whether the DNNs are able to reach the goal position and in computing the cost function along the DNN trajectory.

The mean absolute error (MAE) is used to evaluate the difference between the optimal actions and the network predictions. Although this measure is useful for comparison purposes, it is not a way to evaluate how well the required task is accomplished. Small errors are propagated through the whole trajectory and may result in sub-optimal trajectories as well as in catastrophic failures. The full DNN-driven trajectories are thus compared to the optimal ones. Figure 1 shows a comparison between a deep network and a shallow one in terms of both the predicted controls and (D)NN-driven trajectories.

To study the multicopter and spacecraft domains, 100 ran-

Multicopter - Power

Training area	Outside (5m)	Outside (10m)	Time
0.47% (100%)	0.73% (100%)	2.28% / (64%)	9.75%

Multicopter - Time

Training area	Outside (5m)	Outside (10m)	Power
0.41% (100%)	2.12% (70%)	-	14.92%

Spacecraft model 1 - Mass

Training area	Outside (1km)	Outside (2km)	Time
1.50% (100%)	1.07% (100%)	1.83% (53%)	4.46%

Spacecraft model 2 - Mass

Training area	Outside (1km)	Outside (2km)
1.44% (70%/25%)	1.33% (54%/24%)	- (25%/1%)%

Table 2. Error with respect to the optimal value function and success rate. The values are shown for initial conditions from inside and outside of the training area and the results corresponding to a trajectory optimized for a different objective. For spacecraft model 2, the two values in are the success rate and the percentage of trajectories (classified as unsuccessful) that did not crash but kept ascending and descending above the goal not reaching it.

dom initial positions are generated in $\mathcal{A}$ and the DNN driven trajectory is simulated. Table 2 summarizes the results. The multicopter and the spacecraft with the momentum exchange wheel achieve a 100% success rate with high accuracy and precision when they are initialized within $\mathcal{A}$. In the same table we include the error for DNNs trained on a different objective, showing that the errors are, as expected, significantly higher. Examples of the optimal control profile vs. the control along a DNN-driven trajectory are shown in Figure 2.

In the case of the spacecraft model 2, the accuracy is considerably lower although we can see how the structure of the control has been learned (Figure 3). It is also possible to notice how the optimal profiles do not correspond to the expected *bang-bang* control due to the aggressive regularization that had to be used in this case. Even in this case, the neural network is still replicating the behaviour of the training data, but small deviations, in particular in the lateral thrusters may result in catastrophic failures. Improving the data used for training will likely lead to a quality similar to the other models. All the state and control variables duing a full trajectory are shown in Figure 4.

5.1. Generalization

In most cases, high success rates from outside of the training data are obtained without a major impact on performances as shown in table 2. In the case of the power optimal multicopter, if the trajectories are initialized from an extension of $\mathcal{A}$ of 5 [m] both in x and z, the DNN still achieves 100% of safe landings with minor deviations from the optimal cost (0.74%). The success rate remains high (84%) in the case of an extension of 10 [m]. For time optimal trajectories, the success rate outside of $\mathcal{A}$ is lower (70%) mainly due to the terminal velocity v_z violating the set tolerance. In the spacecraft (model 1) case, safe landings are achieved consistently from initial positions up to 1 [km] higher than those considered in $\mathcal{A}$ and more than 50% of safe landings are achieved for an extension of 2 [km].

If the networks have learned an approximation to the solution of the HJB equations, we would expect them to work in conditions that are not considered in the training data. Other strong indications suggest that this is indeed the case.

In Figure 2 we see that, if the simulation is not stopped after reaching the goal position, the multicopter or spacecraft starts hovering close to the goal with zero velocity and the necessary thrust to compensate the gravitational force. For the case of the multicopter it is remarkable that the thrust predicted by the network is ~ 9.81 [N] also in those cases where such a value is not present in the training data. The spacecraft model 1, after reaching its goal, does not hover with precision, but it still shows a similar behaviour and it is possible to observe that the thrust (u_2) constantly decreases, which can be interpreted as an attempt to compensate for the mass reduction due to fuel consumption.

In addition to Table 2, Figure 5 shows how both the multicopter and the spacecraft reach the goal position from initial states well outside of the bounds of the training area. This generalization happens not only for meaningful initial positions but also for points lower than the landing position, which requires to move upwards, a behaviour not presented during training.

5.2. Importance of depth

Table 3 shows how shallow networks with only two layers (one hidden and one output layer) are not able to approximate the state-feedback control as accurately as deep networks. Deep networks always outperform shallow networks with the same number of parameters. In the table we include an example for each one of the domains and we offer a more detailed comparison over u_2 for the spacecraft model 1, given that it has the most representative control profile. It is possible to see how, after some point, doubling the number of units in a shallow network does not produce significant improvements. In this case the performance is worse than deeper networks with a much lower number of parameters, which suggests that, given the data and the training methods used, it is not possible to learn a precise representation of the function with shallow networks.

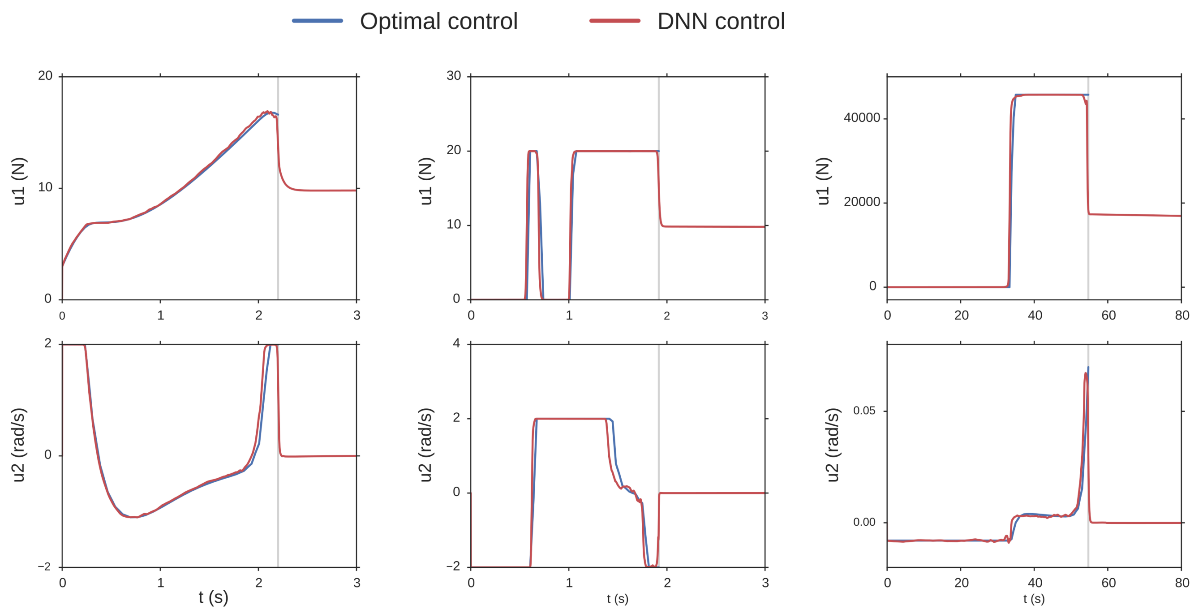

Fig. 2. Control profiles along trajectories driven by a DNN for the multicopter power (left) and time (center) optimal problems and for the spacecraft mass problem (right). After reaching the final goal, the trajectory computed by the network starts a hovering phase. For the multicopter cases this requires a constant thrust of 9.8 [N], for the spacecraft model the thrust decreases over time as the total mass decreases.

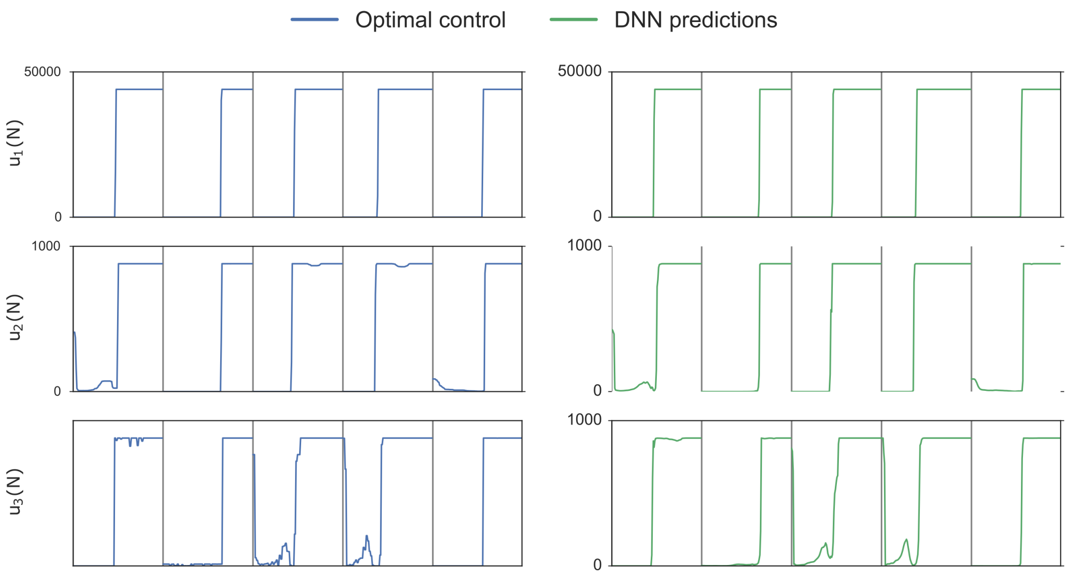

Fig. 3. Optimal control and DNN predictions for the states of several optimal trajectories, not included in the training set, for the spacecraft model 2.

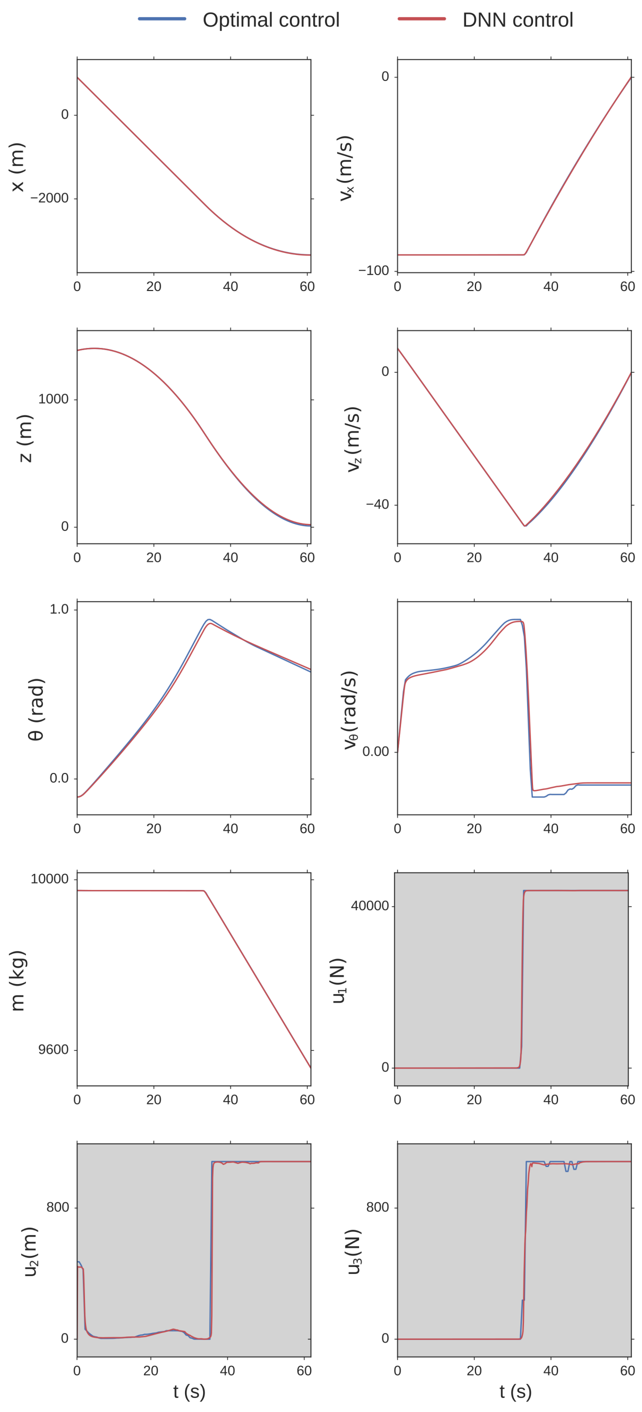

Fig. 4. All state and control (in gray) variables of the spacecraft model 2. The DNN-drive trajectory replicates the optimal trajectory with a high accuracy.

Multicopter - Power

	Layers	Units	#params	Train	Val.
u_1	2	1,104	8,832	0.0897	0.0902
	4	64	8,832	0.0668	0.0672
	5	64	17,216	0.0632	0.0637
u_2	2	1,104	8,832	0.0645	0.065
	4	64	8,832	0.0429	0.0431
	5	64	17,216	0.0416	0.0418

Multicopter - Time

	Layers	Units	#params	Train	Val.
u_1	2	1,104	8,832	0.232	0.234
	4	64	8,832	0.152	0.153
	5	64	17,216	0.145	0.146
u_2	2	1,104	8,832	0.105	0.104
	4	64	8,832	0.0852	0.0853
	5	64	17,216	0.0786	0.0789

Spacecraft model 1 - Mass

	Layers	Units	#params	Train	Val.
u_1	2	1,104	8,832	1243.78	1246.91-
	4	64	8,832	936.98	946.76
	5	64	17,216	930.71	938.16
u_2	2	256	2,048	0,00462	0,00460
	2	1,104	8,834	0,00379	0,00379
	2	2,048	16,384	0,00370	0,00371
	3	64	4,672	0,00307	0,00307
	3	128	17,536	0,00270	0,00272
	3	256	67,840	0,00263	0,00264
	4	64	8,832	0,00250	0,00252
	4	128	34,048	0,00241	0,00242
	5	64	12,992	0,00236	0,00237

Spacecraft model 2 - Mass

	Layers	Units	#params	Train	Val.
u_1	2	1,104	8,832	115.40	114.52
	4	64	8,832	82.31	82.15
	5	64	17,216	85.45	85.70
u_2	2	1,104	8,832	8.46	8.43
	4	64	8,832	6.56	6.52
	5	64	17,216	6.20	6.19
u_3	2	1,104	8,832	16.07	16.13
	4	64	8,832	12.32	12.43
	5	64	17,216	11.84	11.98

Table 3. Effect of the depth in networks trained to predict the control variables of the multicopter and spacecraft models.

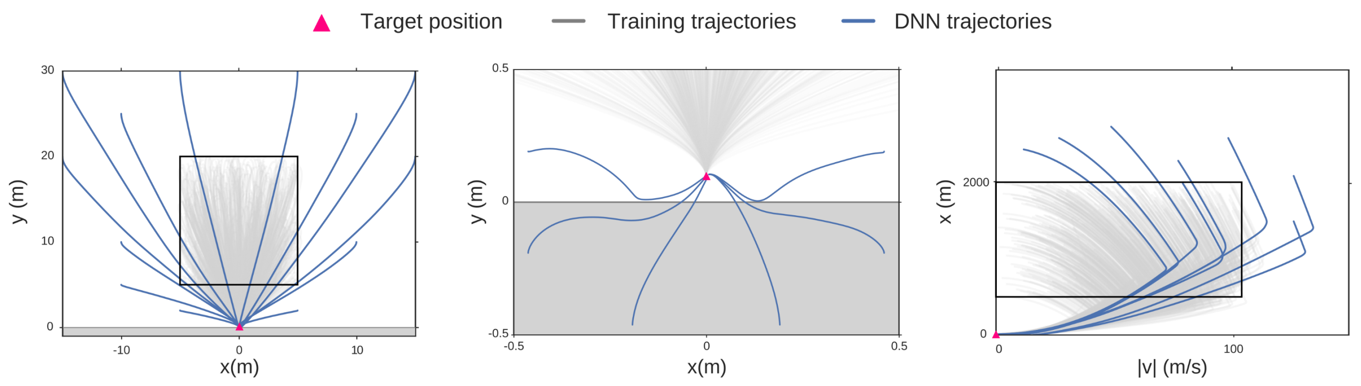

Fig. 5. Trajectories computed by DNNs from initial points outside of the training area both for the power optimal multicopter pin-point landing (left and center) and the mass optimal spacecraft landing (right).

5.3. Speed comparison

The DNN, compared to the original NLP solver, will only provide an advantage if its processing time is lower. A comparison between the CPU times is included in table 4, where it is possible to see that, even when we use a low number of nodes in the NLP solver (resulting in sub-optimal results) the networks are still up to 500 times faster. Although the NLP solver computes the full trajectory, if we want a reactive controller we will have to discard everything except the first action, while the DNN only needs to compute one action at a time.

Non linear programming (NLP) solver

	5 nodes	**20 nodes**	**50 nodes**
Qt	38.86 ms	168.71 ms	594.87 ms
Qp	32.60 ms	222.58 ms	1245.98 ms
S1	82.82 ms	280.70 ms	1830.57 ms
S2	72.95 ms	386.44 ms	3488.82 ms

DNN (network for each variable)

1 - 1140	**4 - 64**	**4 - 128**	**4 - 256**
0.048 ms	0.056 ms	0.068 ms	0.11 ms

Table 4. Time to compute the the control given the state. The time of the NLP solver using 5, 20 and 50 nodes is computed as the mean of 100 trajectories from random initial points. We show the NLP solutions for the four models: multicopter with power (Mp) and time (Mt) optimal control and the spacecraft models 1 (S1) and 2 (S2). The networks (one with 2 layers and 1,140 units and three with 5 layers and 64, 128 and 256 units) are evaluated 100,000 times to compute the average time. All times have been obtained with a Intel(R) Xeon(R) E5-2687W @ 3.10GHz CPU.

6. CONCLUSIONS

We have shown that deep neural networks (DNN) can be trained to learn the optimal state-feedback in a number of continuous time, deterministic, non-linear systems of interest in the aerospace domain and in particular in landing applications. 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 (HJB) equations is the underlying model being learned. The depth of the networks has a great influence on the obtained results and we find that shallow networks, while trying to approximate the optimal state-feedback, are unable to learn its complex structure satisfactorily.

Our work opens to the possibility to design real-time optimal control architectures for planetary landing using a DNN to drive directly the state-action selection. With this respect we show that 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 it is also safe in terms of avoiding catastrophic failures.

7. REFERENCES

[1] Behcet Acikmese and Scott R Ploen, "Convex programming approach to powered descent guidance for mars landing," *Journal of Guidance, Control, and Dynamics*, vol. 30, no. 5, pp. 1353–1366, 2007.

[2] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015, Insight.

[3] Sergey Levine, "Exploring deep and recurrent architectures for optimal control," *arXiv preprint arXiv:1311.1761*, 2013.

[4] Tianhao Zhang, Gregory Kahn, Sergey Levine, and Pieter Abbeel, "Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search," *arXiv preprint arXiv:1509.06791*, 2015.

[5] Sohrab Effati and Morteza Pakdaman, "Optimal control problem via neural networks," *Neural Computing and Applications*, vol. 23, no. 7-8, pp. 2093–2100, 2013.

[6] Yang Xiong, Liu Derong, Wang Ding, and Ma Hongwen, "Constrained online optimal control for continuous-time nonlinear systems using neuro-dynamic programming," in *Control Conference (CCC), 2014 33rd Chinese*. IEEE, 2014, pp. 8717–8722.

[7] P.V. Medagam and F. Pourboghrat, "Optimal control of nonlinear systems using rbf neural network and adaptive extended kalman filter," in *American Control Conference, 2009. ACC '09.*, June 2009, pp. 355–360.

[8] Emanuel Todorov, "Optimality principles in sensorimotor control," *Nature neuroscience*, vol. 7, no. 9, pp. 907–915, 2004.

[9] Frank L Lewis and M Abu-Khalaf, "A hamilton-jacobi setup for constrained neural network control," in *Intelligent Control. 2003 IEEE International Symposium on*. IEEE, 2003, pp. 1–15.

[10] Y. Tassa and T. Erez, "Least squares solutions of the hjb equation with neural network value-function approximators," *Neural Networks, IEEE Transactions on*, vol. 18, no. 4, pp. 1031–1041, July 2007.

[11] Max Berniker and Konrad P Kording, "Deep networks for motor control functions," *Frontiers in computational neuroscience*, vol. 9, 2015.

[12] Martino Bardi and Italo Capuzzo-Dolcetta, *Optimal control and viscosity solutions of Hamilton-Jacobi-Bellman equations*, Springer Science & Business Media, 2008.

[13] Randal W Beard, George N Saridis, and John T Wen, "Galerkin approximations of the generalized hamilton-jacobi-bellman equation," *Automatica*, vol. 33, no. 12, pp. 2159–2177, 1997.

[14] Dario Izzo and Guido de Croon, "Nonlinear model predictive control applied to vision-based spacecraft landing," in *Proceedings of the EuroGNC 2013, 2nd CEAS Specialist Conference on Guidance, Navigation & Control, Delft University of Technology*, 2013, pp. 91–107.

[15] Markus Hehn, Robin Ritz, and Raffaello DAndrea, "Performance benchmarking of quadrotor systems using time-optimal control," *Autonomous Robots*, vol. 33, no. 1-2, pp. 69–88, 2012.

[16] John T Betts, *Practical methods for optimal control and estimation using nonlinear programming*, vol. 19, Siam, 2010.

[17] Philip E Gill, Walter Murray, and Michael A Saunders, "Snopt: An sqp algorithm for large-scale constrained optimization," *SIAM review*, vol. 47, no. 1, pp. 99–131, 2005.

[18] Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun, "What is the best multi-stage architecture for object recognition?," in *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE, 2009, pp. 2146–2153.

[19] Xavier Glorot, Antoine Bordes, and Yoshua Bengio, "Deep sparse rectifier neural networks," in *International Conference on Artificial Intelligence and Statistics*, 2011, pp. 315–323.

[20] Yann A. LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller, *Neural Networks: Tricks of the Trade: Second Edition*, chapter Efficient BackProp, pp. 9–48, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[21] Xavier Glorot and Yoshua Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *International conference on artificial intelligence and statistics*, 2010, pp. 249–256.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1026–1034.

[23] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton, "On the importance of initialization and momentum in deep learning," in *Proceedings of the 30th international conference on machine learning (ICML-13)*, 2013, pp. 1139–1147.