AAS 18-XXX (Pre-print)

A RECURRENT DEEP ARCHITECTURE FOR QUASI-OPTIMAL FEEDBACK GUIDANCE IN PLANETARY LANDING

Roberto Furfaro[*], Ilaria Bloise[†], Marcello Orlandelli[‡], Pierluigi Di Lizia[§], Francesco Topputo[¶], Richard Linares[‖]

Precision landing on large planetary bodies is an important technology that enables future human and robotic exploration of the solar system. For example, over the past decade, landing systems for robotic missions have been developed with the specific goal of deploying robotic agents (e.g. rovers, landers) on the planetary surface (e.g. Mars, Moon). Considering the strong interest for sending humans back to the Moon within the next decade, the landing system technology will continue to progress to keep up with the demand for more stringent requirements. Indeed, more demanding planetary exploration requirements implies a technology development program that calls for more precise guidance systems capable of delivering rovers and/or landers with higher and higher degree of precision. In this paper we design, test and validate a deep Recurrent Neural Network (RNN) architecture capable of predicting the fuel-optimal thrust from sequence of states during a powered planetary descent. Here, the principle behind imitation learning (supervised learning) are applied. A set of propellant-optimal open loop landing trajectories are computed using direct transcription methods (e.g. Gauss Pseudo Spectral methods). Such sequences comprise the training set (i.e. the teacher) employed during the learning phase. A Long-Short Term Memory (LSTM) architecture is employed to keep track of what has entered the network before and use such information to better predict the output. The RNN-LSTM architecture is trained validated and tested to evaluate the performance predictive performance. Finally, the results of a Monte Carlo simulations in Moon landing scenarios are provided to show the effectiveness of the proposed methodology.

INTRODUCTION

Autonomously landing on large and small planetary bodies in a fuel-efficient fashion and with pinpoint accuracy is an extremely challenging problem. Guidance algorithms that can generate fuel-efficient, closed-loop powered descent trajectories must bring the spacecraft to the desired location on the planetary surface with zero velocity (soft landing) and very stringent precision. Here, we define landing achieved with pinpoint accuracy, if the desired position achieved with accuracy less than 10 meters. The spacecraft Guidance, Navigation and Control (GNC) subsystem must integrate a set of critical functions that can drive the lander safely to the surface of the target body. Such

[*]Professor, Department of Systems and Industrial Engineering, Department of Aerospace and Mechanical Engineering, University of Arizona, Tuscon, AZ 85721. E-mail: robertof@email.arizona.edu

[†]Visiting Graduate Student, Department of Systems and Industrial Engineering, University of Arizona, Tucson, AZ, 86721

[‡]Visiting Graduate Student, Department of Systems and Industrial Engineering, University of Arizona, Tucson, AZ, 86721

[§]Assistant Professor, Assistant Professor, Dipartimento di Scienze e Tecnologie Aerospaziali, Politecnico di Milano, via La Masa 34, 20156 Milano, Italy

[¶]Assistant Professor, Assistant Professor, Dipartimento di Scienze e Tecnologie Aerospaziali, Politecnico di Milano, via La Masa 34, 20156 Milano, Italy

[‖]Charles Stark Draper Assistant Professor,Department of Aeronautics and Astronautics,Massachusetts Institute of Technology,Cambridge, MA 02139 functions include navigation, i.e. determining position and velocity of the lander from sensor information, and guidance, i.e. determine/compute the appropriate level of thrust and its direction as function of the current state. The typical approach to guidance and navigation for planetary landers relies on separate subsystems and algorithms design. Guidance algorithms usually include both a targeting algorithm and a trajectory-following, real-time guidance algorithm. The targeting algorithm computes either on-line or off-line a reference trajectory that drives the lander to the planetary surface. Such reference trajectory are determined to be as close as possible to fuel-efficient trajectories and must satisfy thrust constraints, both in magnitude and direction, as well as path constraints (e.g. glide slope). Once the reference trajectory is computed, it is made available to the real-time guidance algorithm which is responsible for determining the targeting acceleration command. The latter is implemented by the lander thrusters to track the reference trajectory for a precise and soft landing. Examples of such algorithm include the original Apollo real-time targeting and guidance algorithm [1]. The latter was successfully implemented by the Apollo 11s Lunar Exploration Module (LEM) to drive the spacecraft on the designated location on the Lunar surface. The guidance algorithm was based on an iterative process capable of generating a quartic polynomial as function of time. Importantly, the feedback Apollo real-time guidance was derived by approximating the nominal trajectory by a 4th-order McLaurin expansion of the reference trajectory [1],[2]. Since the original Apollo guidance algorithm has been implemented, over the past decade, novel guidance methodologies have been extensively researched. Examples include: 1) gravity-turn based guidance [3], 2) Feedback ZEM/ZEV guidance [4], 3) robust guidance based on time-dependent sliding [5], 4) feedback linearization [6] as well as guidance algorithms based on hybrid control theory [7]. All such methods seek to explore the latest advancement in both robust control and optimal control methodologies. Another example of propellant-optimal guidance approach can be found in [9], where the principle of convex optimization have been employed to generate real-time optimal feedback guidance for planetary landing as a sequence of open-loop optimal convex problems. Over the past few years, propelled by advancements in parallel computing technologies (e.g., Graphic Processing Units, GPUs), availability of massive labeled data and critical breakthrough in understanding hot deep neural networks process information to provide high-level of abstraction, there has been a new and renewed interest in machine learning algorithm. For example, the last generation of deep neural networks can accurately process sequence of data for both classification and regression tasks (e.g., image and video recognition [10], natural language processing [11], speech recognition [12], just to mention as few). In spite of such advancement and interest for machine learning approaches, very little has been done in the real of space space exploration. Indeed, very little can be found in the literature about autonomous guidance and navigation algorithms based on the latest advancement in deep learning. A couple of examples include learning optimal feedback guidance via supervised learning ([13], [15]) and reinforcement learning ([14] as well as RTN via convolutional neural networks [19]. Deep architecture have been explored both using Convolutional Neural Networks [21] and Reinforcement Learning for 6-DOF landing [20].

The goal of this paper is to design, test and validate a deep Recurrent Neural Network (RNN) architecture capable of predicting the fuel-optimal thrust from sequence of states during a powered planetary descent. RNNs are flexible neural architectures that can be efficiently trained to predict the next state from input time sequences. Within a powered descent Moon landing scenario, a deep RNN is designed to predict the thrust control action directly from the knowledge of the state. Here, the principle behind imitation learning are applied. Indeed, a set of optimal, fuel efficient trajectories simulating open-loop guidance toward the Moon surface are generated using direct transcription methods (e.g. Gauss Pseudo Spectral methods). Such sequences comprise the training set (i.e. the teacher) employed during the learning phase. A Long-Short Term Memory (LSTM) architecture is employed to keep track of what has entered the network before and use such information to better predict the output. Finally, the proposed RNN learns to imitate a fuel-efficient, closed-loop guidance for Lunar precision landing. After the problem formalization, in which the equations of motion are formulated, the procedure to generate the dataset is described. Subsequently, the proposed network architecture is analyzed and discussed and the results of the training and test phases are reported. A Data Aggregation (Dagger) method where the teacher is employed to correct the network mistakes is implemented to improve accuracy. Finally, the results of a Monte Carlo simulations in Moon landing scenarios are provided to show the effectiveness of the proposed methodology.

PROBLEM FORMULATION

A planar 2D Lunar landing has been considered, in which the variables of state are downrange and altitude (i.e. the motion is constrained to occur in the vertical plane). The state comprises five components: the spacecraft dynamical state position and velocity, and mass. The control action will have two components, one aligned with the vertical direction and one with the horizontal. The thrust vector is described by its magnitude and a two-component unit vector describing the thrust direction. The equations of motion are the following:

$$\begin{cases} \dot{\boldsymbol{r}} = \boldsymbol{v} \\ \dot{\boldsymbol{v}} = \boldsymbol{g} + \dfrac{\boldsymbol{T}}{m} \\ \dot{m} = -\dfrac{\|\mathbf{T}\|}{I_{sp}\, g_0} \end{cases} \tag{1}$$

Here, h is the altitude and d is the downrange, $\boldsymbol{r} = [d, h]$ and $\boldsymbol{v} = [v_d, v_h]$. The thrust $\boldsymbol{T} = T\boldsymbol{u}$ where T is thrust magnitude and $\boldsymbol{u}$ is the thurst direction.Additionally, $\mathbf{g}$ is the lunar gravity acceleration, I_{sp} is the specific impulse and g_0 is the reference gravity acceleration. Fig. 1 shows a simple scheme of the involved variables. These equations have been integrated to minimize the

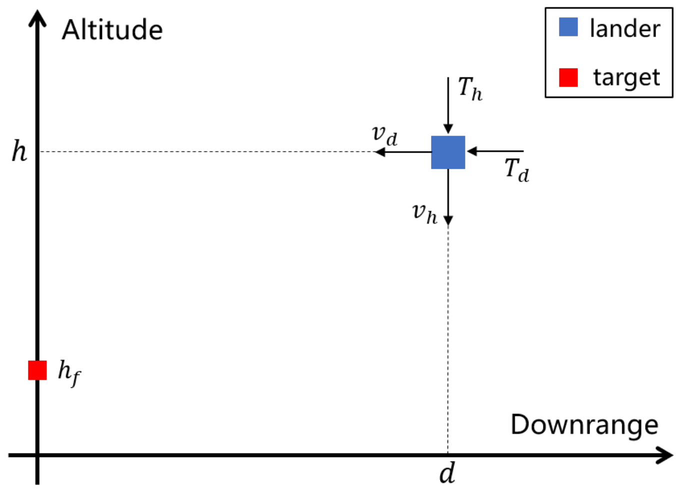


Figure 1: Involved variables in the selected reference frame

following cost function:

$$\underset{t_f,\mathbf{T}}{max} \quad m_L(t_f) = \min_{t_f,\mathbf{T}} \int_0^{t_f} \|\mathbf{T}\|\mathbf{dt} \tag{2}$$

Here, we are minimizing the overall fuel. Importantly, the final time t_f is not fixed. It is very well known that the solution of such problem has a optimal bang-bang burn profile, in which the thrust is alternatively maximum or minimum, or on/off. Initial conditions are given as initial position $\vec{r}_0$ (downrange and altitude) and initial velocity $\vec{v}_0$ (horizontal and vertical). The initial mass is set equal to m_0. Final conditions on each variable of state, except the mass, have been imposed. Since the final time is not fixed, only a lower limit for the mass value has been considered and set equal to the dry mass m_{dry}.

TRAINING SET GENERATION

Whenever dealing with supervised learning, a training set is required to train the network. In order to generate a suitable dataset, the problem just formalized had to be solved many times by considering a set of different initial conditions. For this purpose, an optimizer was necessary. The General Pseudospectral Optimal Control Software (GPOPS) has been chosen [16]. GPOPS is a *MATLAB* software package specifically designed to solve general non-linear optimal control problems, where systems are described by differential or algebraic equations. GPOPS has been chosen for its simplicity in problem implementation and for the ease in visualizing and handling states and control actions. In the followings, the most important steps for the implementation are illustrated. In GPOPS, the user has to define the system of equations of motion and the cost function. This equations are written in two separate scripts, while in the main code, the user defines initial conditions, final conditions and the bound constraints. These bounds are defined in terms of maximum and minimum values that each state and each control action can assume during the integration and optimization. Their value should be properly selected to avoid being too tight. General state-constraints can also be imposed. GPOPS needs a first guess solution to start the optimization process. Since the problem has to be solved multiple times, the free-fall solution has been selected as the initial guess solution for only the first set of initial conditions. Subsequently, each GPOPS solution has been considered as the initial guess for the next set of initial conditions. The latter helped the software to converge faster and more accurately.

As discussed above, the generation of a suitable training set requires running GPOPS with many initial conditions. The initial mass of the spacecraft has been set equal to 1300 kg. The downrange has been initialized between 1500 and 2000 meters, whereas the altitude is selected to be between 1000 and 1500 meters. Fig. 2 shows a schematic of the initial positions in the selected reference frame.

The initial downrange velocity v_{d_0} changes according to the downrange: when the downrange is maximum, the velocity is maximum in modulus (-15 m/s). Conversely, whenever the downrange is minimum, the velocity also is minimum in modulus (-11 m/s). The same reasoning has been applied for the initial vertical velocity v_{h_0}, that ranges from -6 to -10 m/s. Unlike the initial conditions, the final ones have been kept constant for all trajectories. In particular, the final downrange d_f has been set equal to 0, the final altitude h_f equal to 50 m and finally, both components of the final velocity (vertical and horizontal) equal to -0.5 m/s. As it can be seen, the final condition is such that the spacecraft has not touched the ground and has a very small downward velocity. Throttlable thrusters have been considered, in which the magnitude of the control action is bounded following the same strategy applied in [18]. The nominal thrust T_{nom} is equal to 4000 N. The

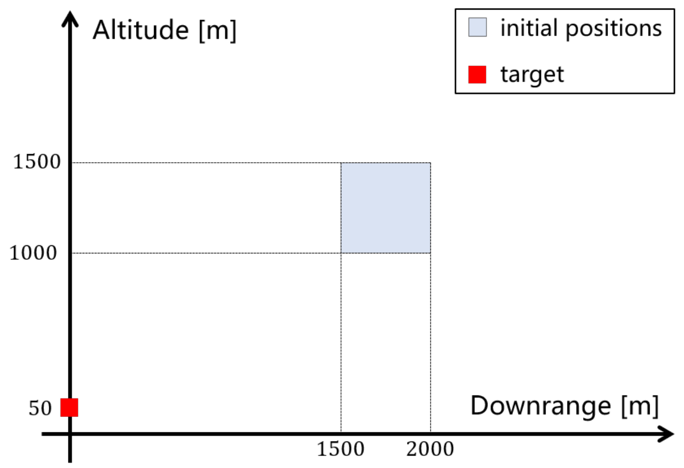


Figure 2: Initial positions in the selected reference frame

maximum thrust and minimum thrust have been fixed equal to 85% and 25% of the nominal thrust respectively (i.e., 3400 N and 1000 N respectively). Since the optimal burn profile is bang-bang, the thrust selection is binary, i.e. it can be either T_min or T_max. As such, the problem of synthesizing the thrust magnitude as function of state can be cast as a binary classification problem. However, learning the thrust direction as function of the spacecraft state is a regression problem. To understand how the thrusters direction has been taken into account, it may be useful to see how the equations of motion have been implemented in GPOPS. Indeed, one has five (5) equations of motion:

$$\begin{cases} \dot{d} = v_d \\ \dot{h} = v_h \\ \dot{v_d} = \dfrac{T}{m} \cdot u_d \\ \dot{v_h} = -g + \dfrac{T}{m} \cdot u_h \\ \dot{m} = -\dfrac{T}{I_{sp}\, g_0} \end{cases} \tag{3}$$

where T is the magnitude of the control action and u_d and u_h are the components of the thrust direction unit vector. To satisfy the unitary conditions of the thrust directions, we imposed that $u_d^2 + u_h^2 = 1$ must be satisfied at any time. Because of such constraint, the thrust direction can be described by a single angle as follows:

$$\theta = \arctan \frac{u_d}{u_h} \tag{4}$$

Designing a RNN capable of learning the thrust magnitude and thrust direction θ requires solving a classification problem for the magnitude of the thrust and a regression problem for the direction of the control action. The training set has been generated by computing 2601 fuel-efficient trajectories have within the selected four-dimensional (position and velocity) portion of space. The parameters of the problem are shown in Tab. 1a. A summary of the initial and the final conditions is shown in

Tab. 1b and in Tab. 1c.

Table 1: Parameters and state values for the 2D problem

(a) Problems parameters

	value	unit
m_{dry}	500	kg
g	-1.622	m/s^2
Isp	200	s
T_{nom}	4.0	kN
T_{max}	3.4	kN
T_{min}	1.0	kN

(b) 2D initial conditions

	value	unit
d_0	$\left[1.5,\ 2.0\right]$	km
h_0	$\left[1.0,\ 1.5\right]$	km
v_{d_0}	$\left[-11,\ -15\right]$	m/s
v_{h_0}	$\left[-6, -10\right]$	m/s
m_0	1.3	ton

(c) 2D final conditions

	value	unit
d_f	0	m
h_f	50	m
v_{d_f}	-0.5	m/s
v_{h_f}	-0.5	m/s

Each trajectory computed by GPOPS has 61 points, i.e. 61 states and 61 control actions. Each state has five elements (downrange, altitude, velocities and mass); each control action has three components (magnitude, u_d and u_h). Since the angle θ is extracted for each couple of unit vector components, 61 angles have been computed per each trajectory. The final training set has been built by associating each spacecraft state (position and velocity) to each control action. The overall fuel-efficient dataset has been further divided in training-set and test-set. The earlier set, which is employed train the RNN comprises 2409 trajectories. Conversely, the test-set, which is employed to evaluate network performances, comprises 192 optimal trajectories.

In Fig. 3 shows an example of an optimal trajectory included in the dataset. The initial downrange is $1750m$ and the initial altitude is $1250m$. Fig. 4 shows the bang-bang profile of the thrust magnitude. Fig. 5 shows the behaviour of the angle θ, that the thrust forms with the horizontal direction. Finally, in Fig. 6 all the trajectories are plotted. All trajectories are constrained to be in a region of space above a cone of 20°.

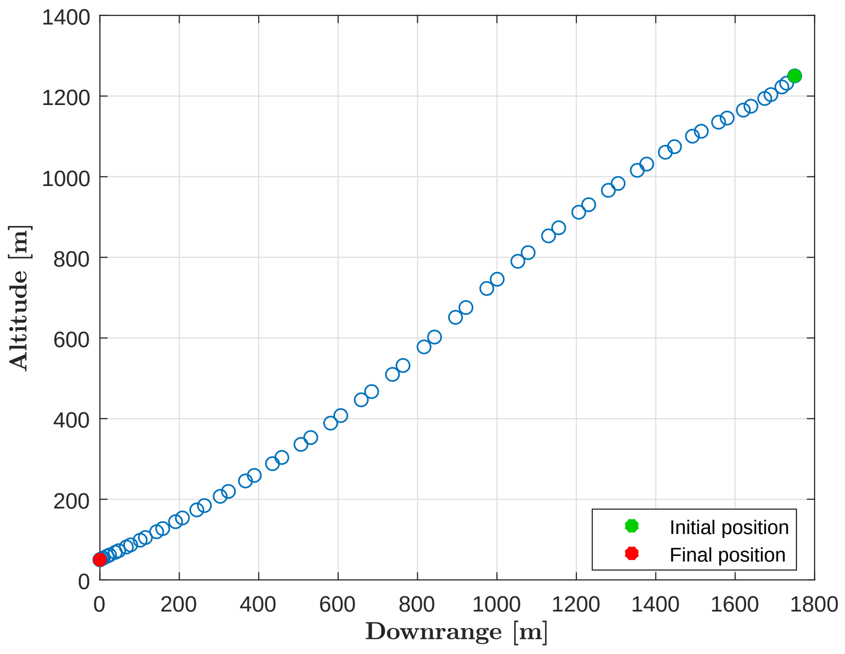


Figure 3: 2D trajectory

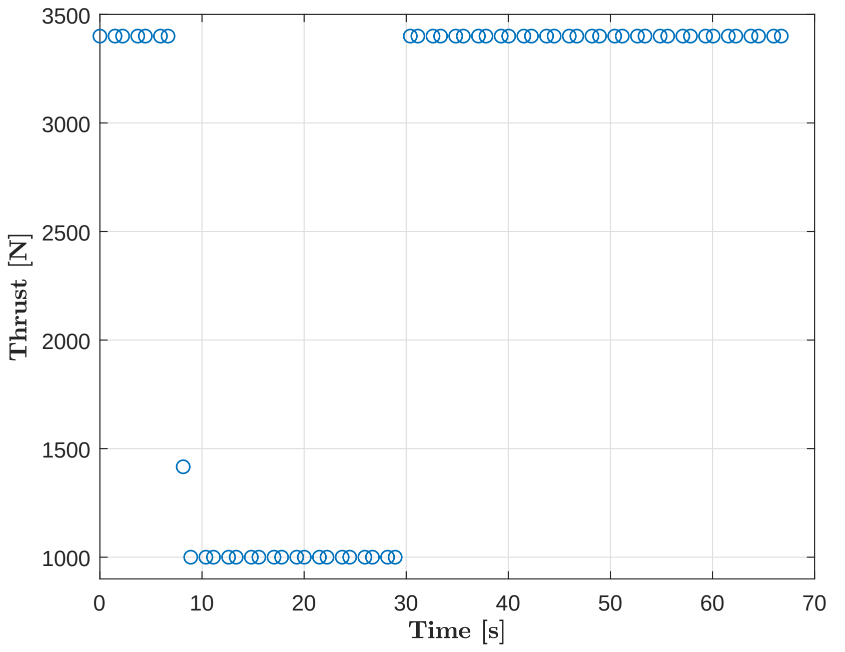


Figure 4: 2D thrust magnitude

PROPOSED RNN-LSTM ARCHITECTURE

The proposed RNN-LSTM network is designed to learn the relationship between sequences of optimal (fuel-efficient) spacecraft state-thrust from numerically computed trajectories. The idea is

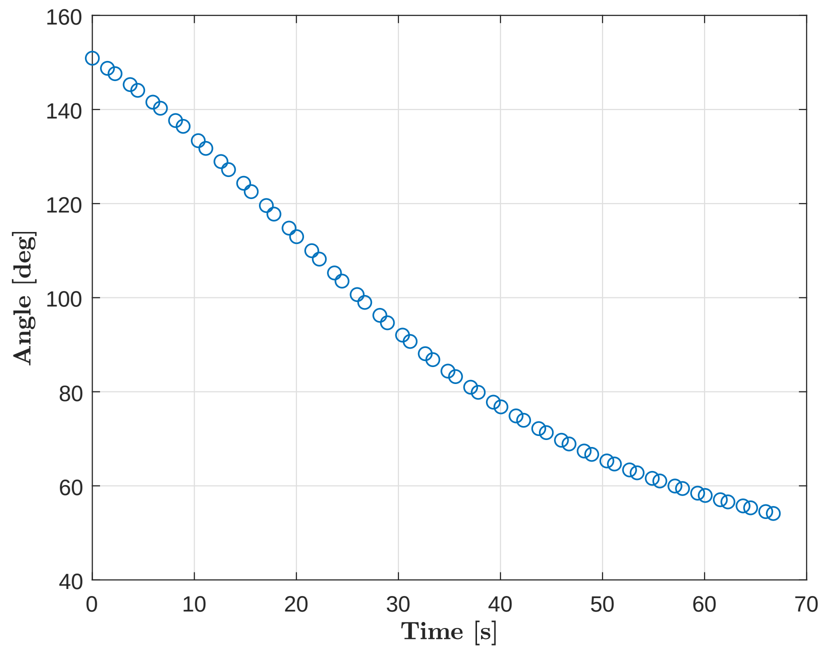


Figure 5: 2D thrust angle

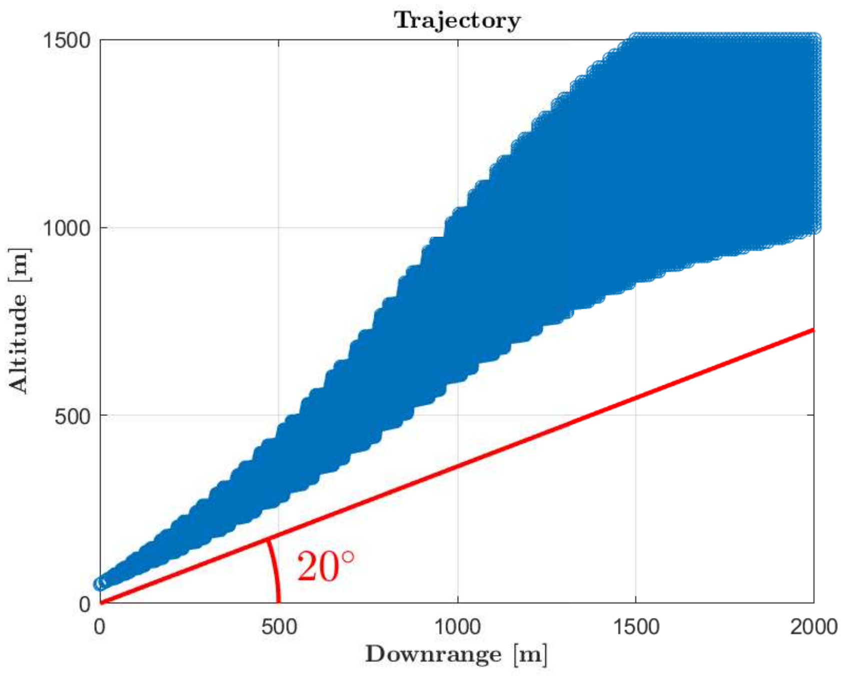


Figure 6: All 2D trajectories

to synthesize the optimal closed-loop controller via imitation learning, i.e. leaning to imitate optimal guidance solutions. For the 2D vertical plane lunar landing problem, the label associated to each state comprises two components, i.e. one associated to the magnitude of the thrust and the second one to the angle representing the direction in which the thrust is applied. Since the problem has a bang-bang control action profile, the thrust magnitude can take its maximum or minimum value only. Accordingly, the network architecture should be designed to classify the correct thrust level.Moreover, as shown in Section ., the thrust angle profile is smooth and continuous. Consequently, the network shall be designed to learn the optimal thrust angle value as function of the previous states. Fig. 7 shows the proposed network architecture. The input enters the LSTM block and the output is sent to two separate branches, i.e. one for classification and the other for regression. A set of fully connected layers link LSTM to the final output levels.

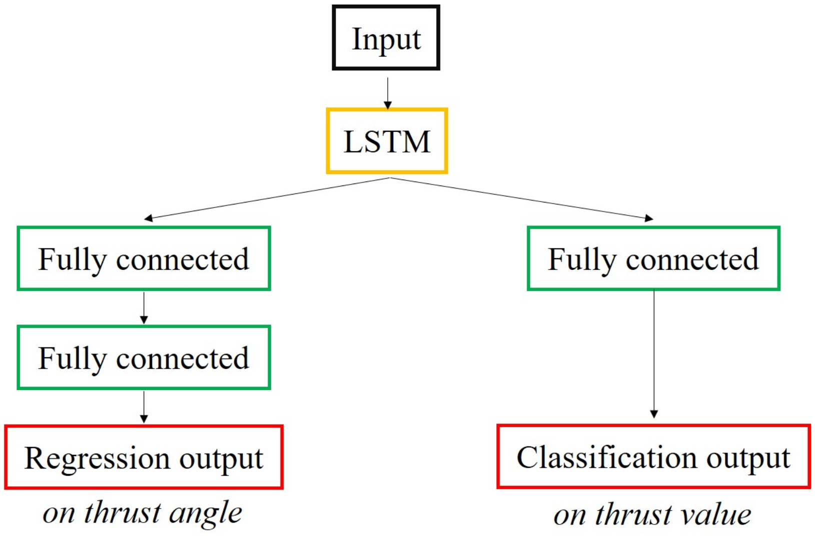
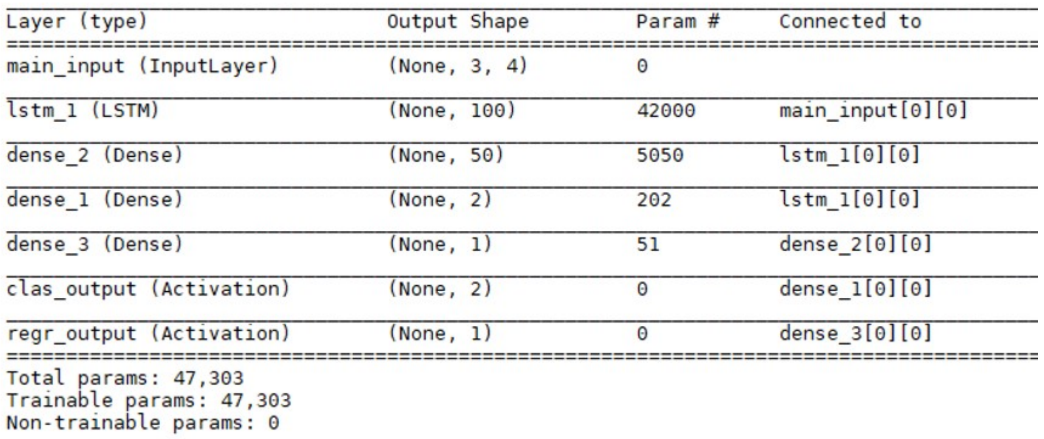


Figure 7: Proposed LSTM architecture

The network has been implemented in *Python-Keras* and the architecture is detailed in Fig. 8. It



```
Layer (type)                     Output Shape          Param #     Connected to
=====================================================================================
main_input (InputLayer)          (None, 3, 4)          0
_____________________________________________________________________________________
lstm_1 (LSTM)                    (None, 100)           42000       main_input[0][0]
_____________________________________________________________________________________
dense_2 (Dense)                  (None, 50)            5050        lstm_1[0][0]
_____________________________________________________________________________________
dense_1 (Dense)                  (None, 2)             202         lstm_1[0][0]
_____________________________________________________________________________________
dense_3 (Dense)                  (None, 1)             51          dense_2[0][0]
_____________________________________________________________________________________
clas_output (Activation)         (None, 2)             0           dense_1[0][0]
_____________________________________________________________________________________
regr_output (Activation)         (None, 1)             0           dense_3[0][0]
=====================================================================================
Total params: 47,303
Trainable params: 47,303
Non-trainable params: 0
```

Figure 8: LSTM architecture in *Python-Keras*

is useful to analyze how each component of the LSTM-net by going through the elements reported in Fig 8.

- LSTM cell is composed of 100 neurons (line `lstm_1` in Fig. 8) and is directly connected to the input layer.
- The branch of the neural network designed to perform the classification on the thrust value (line `dense_1` and `clas_output` in Fig. 8) is composed by a dense layer (fully connected layer) and an output layer, both of 2 neurons because the problem has two output classes. The fully connected layer is connected with the LSTM cell.
- For the regression on the thrust angle (line `dense_2`, `dense_3` and `regr_output` in Fig. 8) there are two fully connected layers (with 50 and 1 neurons) and the output layer. Also here the first fully connected layer is directly linked to the LSTM cell.

The selected architecture yields a number of 47,303 trainable parameters (weights and biases).

TRAINING AND TEST PHASE

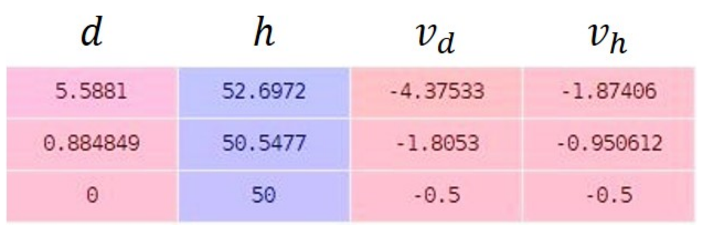
Training

Much effort has been put into selecting the proper length for the input sequence to answer the following question: How could the LSTM learn better from the input information? After a proper tuning, the chosen number of inputs is 3, i.e. spacecraft states and thrust at current and two previous instants. Indeed, after several simulations, we concluded that three sequential states gave the best results. This means that each trajectory (which is composed by 61 states) has been divided in sets of 3 sequential states and this division has been developed by following two strategies.

The first strategy has been to fed the RNN-LSTM with sequential states and without separating different trajectories as shown in Fig. 9. Note that in this case after one trajectory ends the new one starts. However, another strategy has been shown to outperformed the above one. Here, we fed the RNN-LSTM by separating the trajectories one with respect to the other. An example is reported in Fig. 10. This second strategy has been implemented to avoid confusion between trajectories. In this case, the training results are improved because the RNN$-$LSTM is able to learn the final state of the optimal trajectory (i.e. $[0, 50, -0.5, -0.5]$).

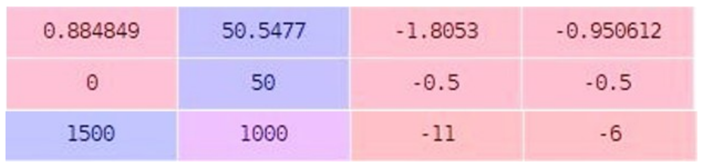
As for labeling the output (thrust magnitude), in each input of 3 states (the 3×4 matrices reported in Fig. 9 and Fig. 10) the associated label corresponds to the 3^{rd} row. During the training phase, at time t, the net is fed with an input comprising the state at time t, at time $t-1$ and $t-2$ as well as the control action at time t. Further, machine learning theory states that input data can be given to the network sequentially, either individually or by small groups or ”batches”. Indeed, the *batch size* is the number of samples (inputs) that are propagated through the network at each training epoch. It has been shown [22] that a multiple input (and therefore a reasonable value of batch size) has some advantages in the training phase even if there are no golden rules to choose it. In this work, the chosen approach is to have a batch of 59 inputs; in this way a complete trajectory is taken.

The RNN-LSTM is usually trained by minimizing a proper *loss function*. In fact, weights and biases are initialized as random values and as the training phase proceeds, they are updated according to a minimization algorithm. The standard learning algorithm for neural networks is backpropagation with gradient descent (or its variant stochastic gradient descent). In this woek, we used the *Adam optimizer* to minimize the loss function. The name is derived from ADAptive Moment estimation. It is well suited for problems that are large in terms of data and parameters and it can be used instead of the classical stochastic gradient descent procedure [23], which maintains a single



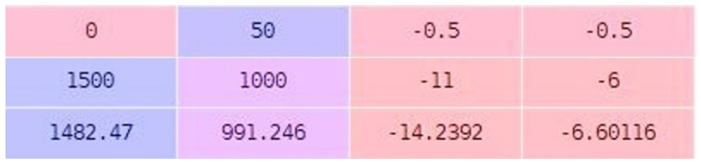
d	h	v_d	v_h
5.5881	52.6972	-4.37533	-1.87406
0.884849	50.5477	-1.8053	-0.950612
0	50	-0.5	-0.5

(a) 1st input



0.884849	50.5477	-1.8053	-0.950612
0	50	-0.5	-0.5
1500	1000	-11	-6

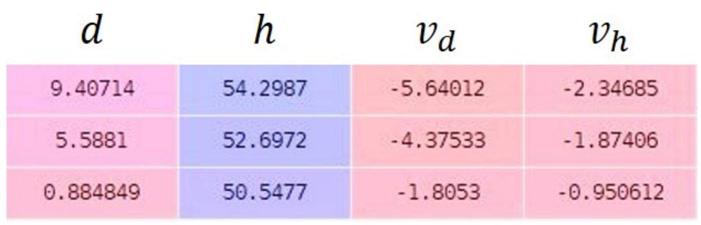
(b) 2nd input



0	50	-0.5	-0.5
1500	1000	-11	-6
1482.47	991.246	-14.2392	-6.60116

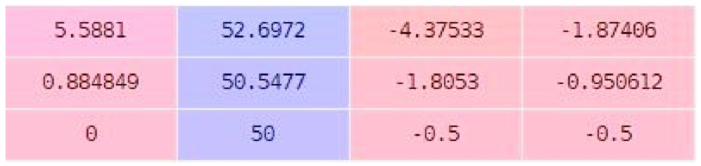
(c) 3rd input

Figure 9: Example of inputs shape for the first strategy



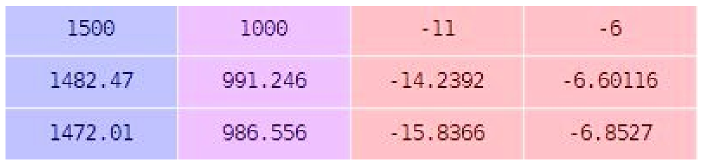
d	h	v_d	v_h
9.40714	54.2987	-5.64012	-2.34685
5.5881	52.6972	-4.37533	-1.87406
0.884849	50.5477	-1.8053	-0.950612

(a) 1st input



5.5881	52.6972	-4.37533	-1.87406
0.884849	50.5477	-1.8053	-0.950612
0	50	-0.5	-0.5

(b) 2nd input



1500	1000	-11	-6
1482.47	991.246	-14.2392	-6.60116
1472.01	986.556	-15.8366	-6.8527

(c) 3rd input

Figure 10: Example of inputs shape in second strategy

learning rate (termed alpha) for all weight updates and the learning rate does not change during training. Instead Adam optimizer adapts the learning rate as training unfolds thanks to the *decay* parameter. Note that the learning rate is a hyper-parameter that controls how much the algorithm is adjusting the weights of the network with respect the loss gradient. The lower the value, the slower the travel along the downward slope. While this might be a good idea (using a low learning rate) in terms of making sure that any local minima is missed, it implies that the time for convergence will be longer. Furthermore, the decay has an expression given by Eq. 5

$$lr_{new} = lr_{old} \cdot decay_{rate}^{\left(\frac{epoch}{decay_{step}}\right)} \tag{5}$$

In Tab. 2 the chosen hyper-parameters are summarized.

Table 2: Selected hyper-parameters of the RNN−LSTM

Hyper-parameters	
Batch size	59
Initial learning rate	0.0001
Decay rate	0.0001
Regression loss weight	10
Classification loss weight	60

Classification predictions can be evaluated using accuracy, whereas regression predictions can be evaluated using root mean squared error (RMSE). Concerning the accuracy evaluation, the loss function associated to this task has been the *Cross-entropy loss*.

Cross-entropy loss**:** it measures the performance of a classification model whose output is a probability value between 0 and 1. In particular it indicates the distance between the model prediction and the true value of the output. The cross entropy error is computed as:

$$y_{cross-entropy} = -\sum_j t_j \log(y_{softmax_j}) \tag{6}$$

Softmax function**:** Softmax function outputs a probability distribution suitable for probabilistic interpretation in classification tasks. This function takes a vector of real numbers and transforms it into a vector of real numbers in range [0; 1] which represents the probabilities and will be used for determining the class for the given inputs. The softmax function computes the ratio between the exponential of the input value and the sum of the exponentials of all the input values. The analytic formula is:

$$y_{softmax} = \frac{e^{z_k}}{\sum_{j=1}^{J} e^{z_j}} \tag{7}$$

The RMSE (Eq. 8) for regression, on the other hand, represents the sample standard deviation of the differences between predicted values and real values:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{I} (\bar{y_i} - y_i)^2}{n}} \tag{8}$$

where y_i is the real value and $\bar{y_i}$ the predicted one.

The training has been run for 1500 epochs over 2409 trajectories. In order to have a feedback during the training phase on the RNN-LSTM performances, the training set has been split such that 5% of the dataset has been used for validation. Validation datasets can be used for regularization by early stopping: the training is stopped when the error on the validation dataset increases, as this is a sign of overfitting to the training dataset. In Fig. 11 shows the behaviour of the loss function during the training process. The RNN-LSTM has been trained for 1,200 epochs. Importantly, the orange curve. corresponding to the model validation, shows that the RNN-LSTM does not overfit the data.

Test

The test phase has been executed to validate the performance of the final model after training. As already mentioned, the test set consists of 192 trajectories. To visualize the trained RNN-LSTM performances on the classification (accuracy), a *confusion matrix* has been computed. The number of correct and incorrect predictions has been summarized by counting and breaking them by each class: 0 is associated to minimum thrust, 1 to maximum. As shown in Fig. 12 the RNN-LSTM trained model has an accuracy of $99,73\%$. To evaluate if the classification accuracy is consistently high, MNIST images dataset has been taken as a reference, even if no images are involved in an RNN$-$LSTM. According to the state of art, tests on MNIST (which is a trivial dataset) can at most reach a precision of 99.79%. To obtain a good result in a more complex problem, a threshold of 99% on the accuracy is chosen as minimum target. On the other hand to evaluate the performances on regression the plot on the predicted regression angles has been considered. The results are reported in Fig. 13 which shows good performance with a final RMSE is $0.2°$.

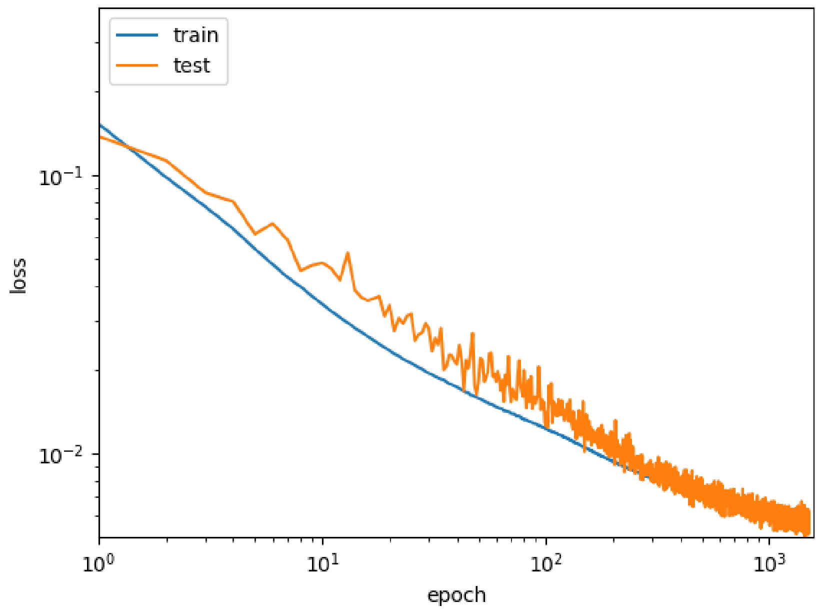
ACCURACY IMPROVEMENT: DAGGER APPROACH

An approach has been investigated to obtain further improvements of the net performances. Several algorithms have been proposed and are available in literature. The most promising is the Imitation Learning thanks to which the learner tries to imitate an external expert action in order to achieve the best performance. In particular in this work a DAgger (Data Aggregation) approach has been developed. The main advantage of this type of algorithms is that an expert teaches the learner how to recover from past mistakes. Nowadays, a classical application of DAgger is for autonomous vehicle and it applies, mathematically speaking, the following steps([17]):

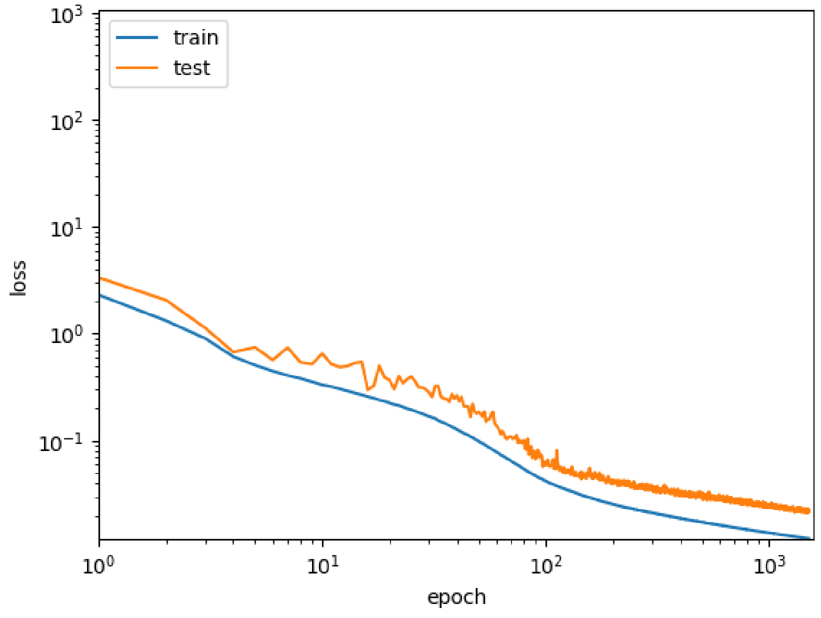
1. Train the net (in this case a car) on a dataset **D** made of **human** observations and actions.
2. Run the net to get performances and then a new set of observations $\mathbf{D}_\pi$.
3. Ask the **human** expert to label the new dataset with actions.
4. Aggregate all data in a new dataset $\mathbf{D}_{new} = \mathbf{D} \cup \mathbf{D}_\pi$.
5. Re-train the net.

Practically what happens with an autonomous car is that the driver corrects online the errors done by the vehicle. Since it is not possible to exploit an human action/correction in space, the DAgger approach developed for this work is slightly different and it goes throught the following steps:

1. Train the net on a dataset **D** generated with GPOPS.



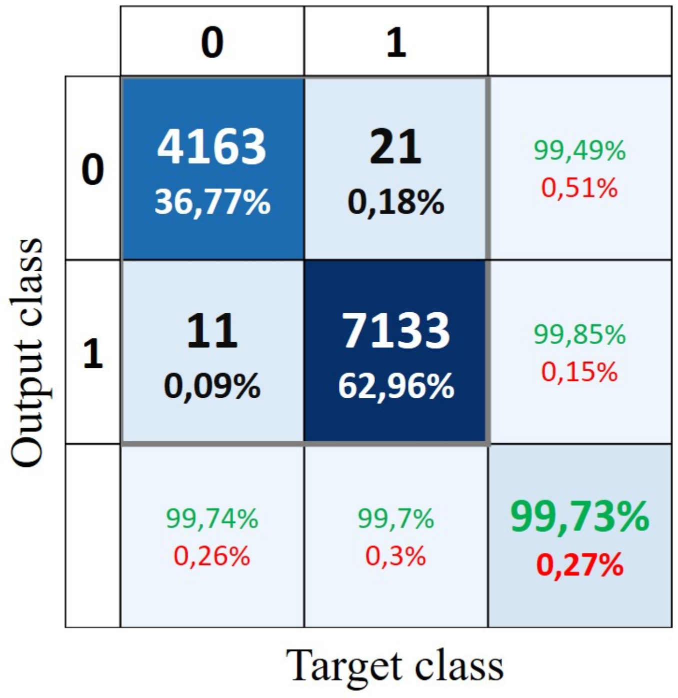
(a) Classification loss



(b) Regression loss

Figure 11: Classification and regression losses evolution during the RNN−LSTM training phase

2. Run the net to get performance by using the test set $\mathbf{D}_{test}$.
3. Check for which trajectories the trained model error is not acceptable in terms of classification accuracy and RMSE.
4. Pick up the wrong trajectories in $\mathbf{D}_{wrong}$.



Output class \ Target class	0	1	
0	4163 36,77%	21 0,18%	99,49% 0,51%
1	11 0,09%	7133 62,96%	99,85% 0,15%
	99,74% 0,26%	99,7% 0,3%	99,73% 0,27%

Figure 12: Confusion matrix RNN-LSTM

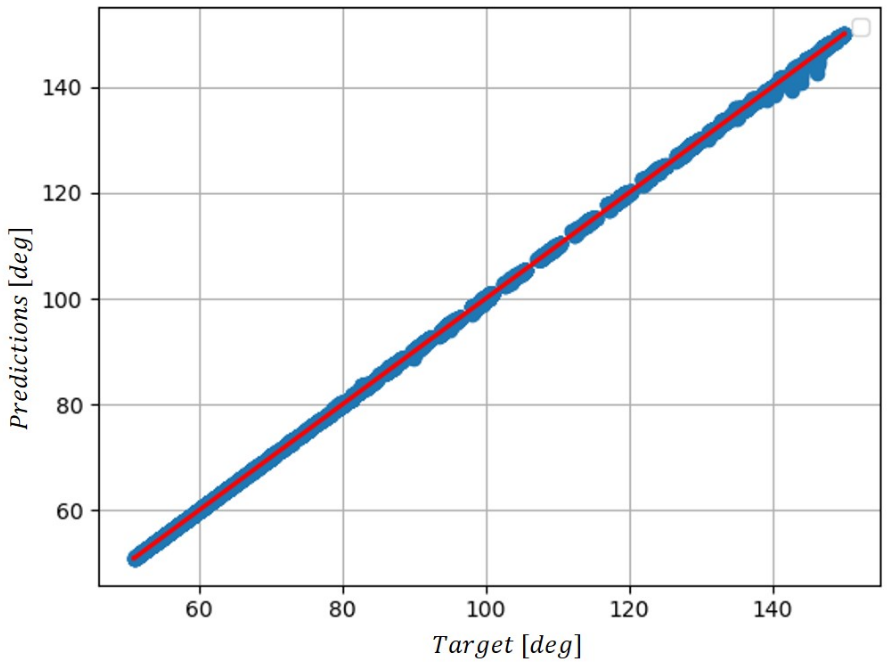


Figure 13: Regression on angles predicted with the trained RNN$-$LSTM

5. Use $\mathbf{D}_{wrong}$ to re-train the net and to improve the performances.

As reported in Section , the global accuracy in the test phase was 99.73% and the global RMSE = $0.2°$. The wrong trajectories have been picked up from the test dataset by applying the criteria shown in Tab. 3 on each prediction. With the trained RNN-LSTM and the abovementioned criteria,

Table 3: Criteria used to evaluate wrong predicted trajectories

Criteria	
Accuracy	¡99%
RMSE	¿0.3°

the wrong trajectories turned out to be 38 on the overall 192 test trajectories. Subsequently, the DAgger method has been implemented following two strategies:

- First strategy has been designed to train only on the trajectories with wrong predictions using the trained net model (Fig. 14).

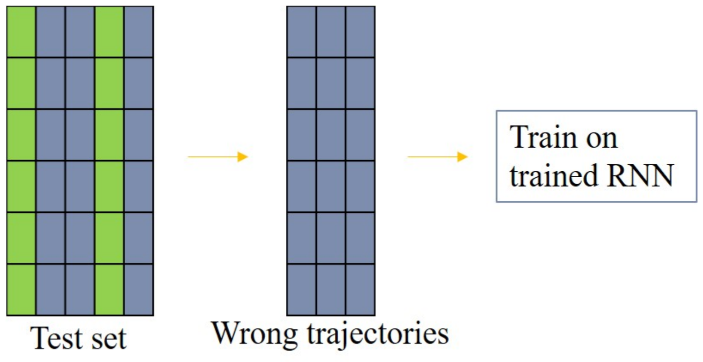


Figure 14: 1° strategy for the DAgger approach

- Second strategy has been conceived to enlarge the training test with the wrong trajectories and then re-train the net model (Fig. 15).

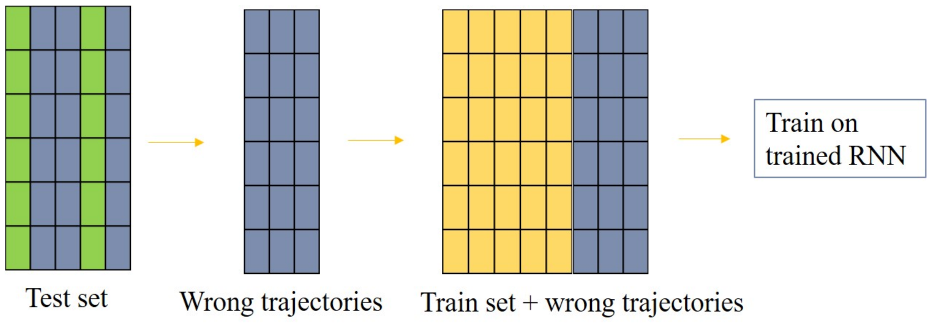


Figure 15: 2° strategy for the DAgger approach

In order to evaluate the performance of the RNN-LSTM trained with the DAgger methodology, a new dataset of 100 trajectories, that had never been used by the model, has been created.The net without DAgger achieves an accuracy of 99.23% and RMSE = 0.42° (Fig. 16). The model trained using the first DAgger strategy reaches an accuracy of 97.89% and RMSE = 0.44° (Fig. 17). Finally the results for the second DAgger strategy provide an accuracy of 99.25% and a RMSE = 0.40° (Fig. 18). All the results are summarized in Tab. 4.

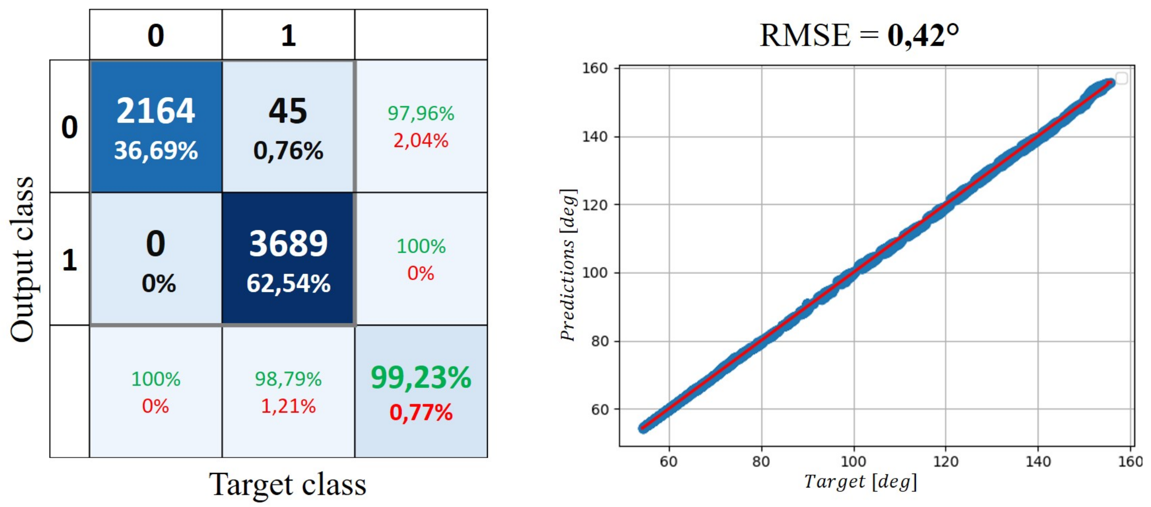


Figure 16: Accuracy and regression using the trained RNN−LSTM on 100 new trajectories

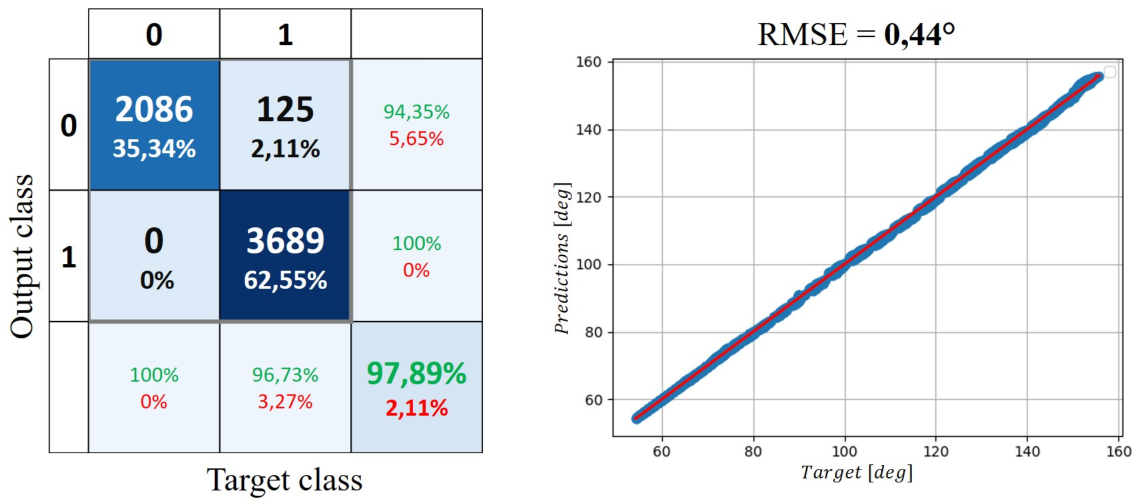
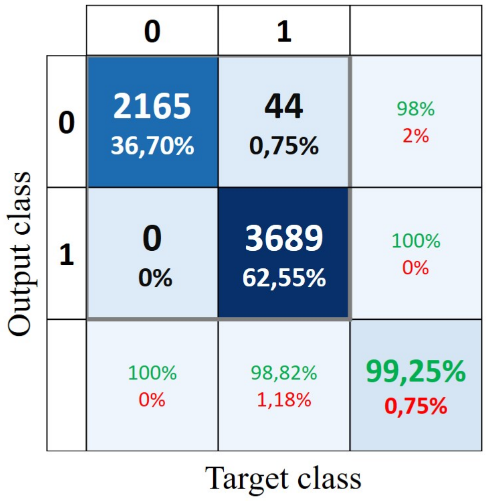


Figure 17: Accuracy and regression using the trained RNN−LSTM according to first DAgger strategy

In conclusion, it has been discovered that re-training the model only on wrong predicted trajectories means that the weights and biases are updated focusing strictly on the wrong predictions. Therefore, testing the new model on a new dataset shows a loss of generality. Instead, re-training on the enlarged set allows to achieve the best performance.



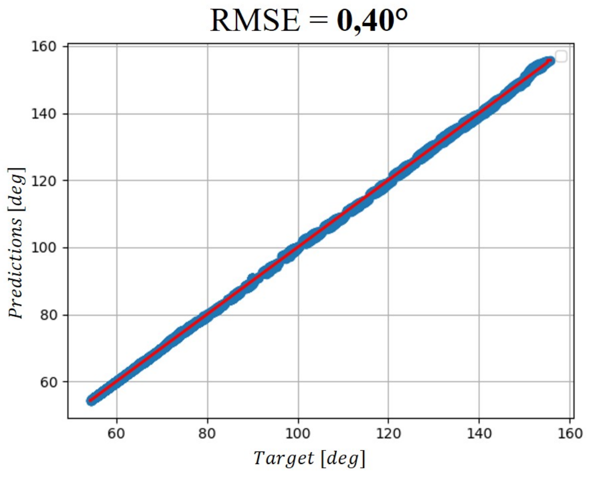


Figure 18: Accuracy and regression using trained RNN−LSTM according to second DAgger strategy

Table 4: Summary of the performances of the DAgger approaches

Model	**Accuracy**	**RMSE**
Trained net	99,23%	0.42°
1° *strategy*	97,89%	0.44°
2° *strategy*	99,25%	0.40°

PERFORMANCE ON A DYNAMICS SIMULATOR

Generally, the RNN−LSTM is able to achieve good results in terms of accuracy on the classification and precision in the regression. Another important test that has been performed is a simulation of the real-time RNN-LSTM performances. The goal is to verify if the spacecraft can be guided to the target above the lunar surface by a well trained neural network. For this purpose, a suitable dynamics simulator has been developed and coupled with the trained RNN−LSTM. Within the simulation environment, starting from an initial condition, the spacecraft is controlled only by RNN-LSTM. Additionally, the state at any instant is supposed to be known. Since the network input comprises three consecutive states, three initial consecutive states have to be available to initiate the simulation. The network takes this first input to predict the first control action, with which the dynamics are propagated throughout the duration of the time step, keeping the control constant. The propagation is performed by integrating the equations of motion with an ordinary differential equation solver based on a 4th order Runge-Kutta method. Once the integration step is completed, the newly generated state is aggregated with the last two states of the previous input, in order to prepare the new input for the network. A new prediction and a new dynamics propagation follow. The loop is repeated until the final time is achieved or until some criteria are satisfied. The first criterion imposes the loop to stop when an altitude of 50 meters (which is the lower limit of the training trajectories) is reached. The second criterion is triggered when the spacecraft starts to rise up in altitude. In fact, it has been discovered that, after the 50 m altitude is reached, the neural network tends to control the spacecraft in such a way that it increases its altitude, moving it away from the ground.

A simulation has been executed by considering the initial conditions of a trajectory taken from the testing set. Tab 5 shows a comparison between the theoretical optimal final state and the final state of a spacecraft completely controlled by the neural network which are demonstrated to be similar. In Fig. 19 the optimal trajectory and the predicted one are reported.

Table 5: Comparison between optimal and predicted final state using RNN−LSTM

	optimal	predicted	unit
downrange	0	-0.9	$[m]$
altitude	50	50.3	$[m]$
downrange velocity	-0.5	-0.015	$[m/s]$
altitude velocity	-0.5	-0.6	$[m/s]$

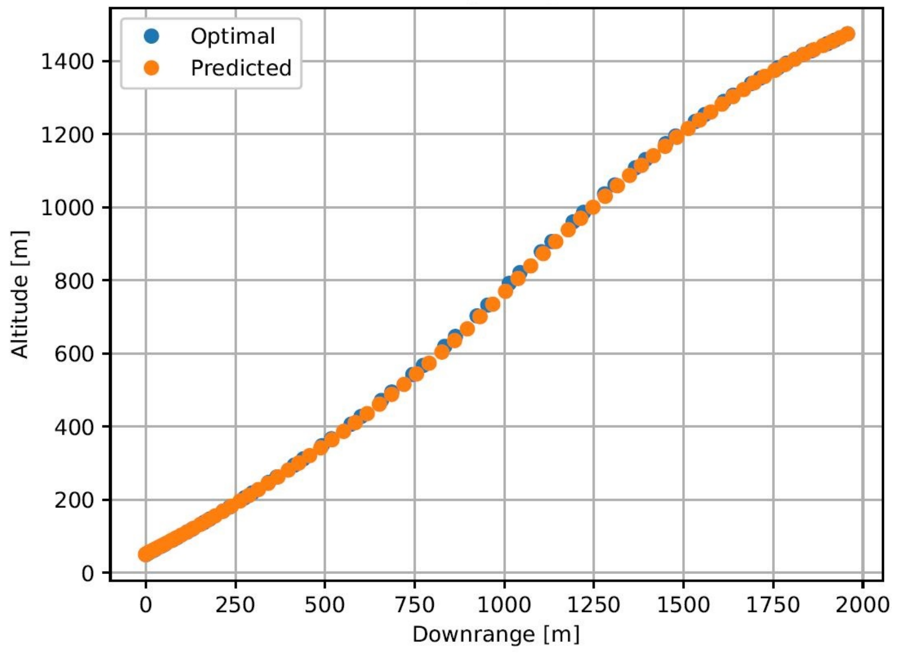


Figure 19: Comparison between optimal and predicted 2D trajectory

The simulations have shown that the network's performances are very sensitive to the time step extension. The time step is the only parameter that can be modified by the user. If the time step is reduced, the network will be required to process the input and give a control action more frequently, while if the time step is increased, a smaller number of predictions will be requested. In the first case, the control would be more accurate, but the network may not be trained to provide high frequency predictions. In fact, asking the network to predict the control actions more frequently than it was designed for, may lead to big errors due to the fact that the inputs would be very similar one to each other. In the second case, the accuracy of the prediction may drop, which could cause the generation of an inaccurate control. A good balance, in which the network is able to give correct predictions and the control is quite accurate despite the approximations, must be found between these two cases. For the example shown, the time step has been set equal to 0.95 s.

MONTE CARLO ANALYSIS

A Monte Carlo simulation has been performed to better characterize the errors of the neural network, in terms of landing position and landing velocity. For this purpose a new set of initial conditions has been generated, keeping fixed the initial altitude at 1250 meters and perturbing the initial downrange according to a Gaussian distribution centred at 1750 meters. In this way, a Gaussian distribution of initial conditions, centred in the middle of the region covered by the dataset (Tab. 1b), has been considered. Fig. 20 shows the distribution of initial downrange and their probability. Note that in this case, it is assumed that the initial conditions in the middle of the dataset are more probable than the peripheral ones. These initial conditions have been propagated using the dynamics simulator, described in the previous section. For each initial condition, the propagation, has been stopped once the spacecraft reached 50 meters of altitude. The output of this simulation is a series of plots, in which the final downrange, the final vertical velocity and the final horizontal one are shown.

As discussed above, an important issue is the selection of the time step. The first Monte Carlo simulation has been performed keeping the time step constant for each trajectory of the set. In this way, each initial condition has been propagated with the same number of control actions predicted by the RNN-LSTM. As reported in Fig. 21 the final state of the propagated trajectories are far from the optimal ones. The latter implies that this is not a good approach. In particular the final velocities are too high (more than -10 m/s for the horizontal component and more than -8 m/s for the vertical one).

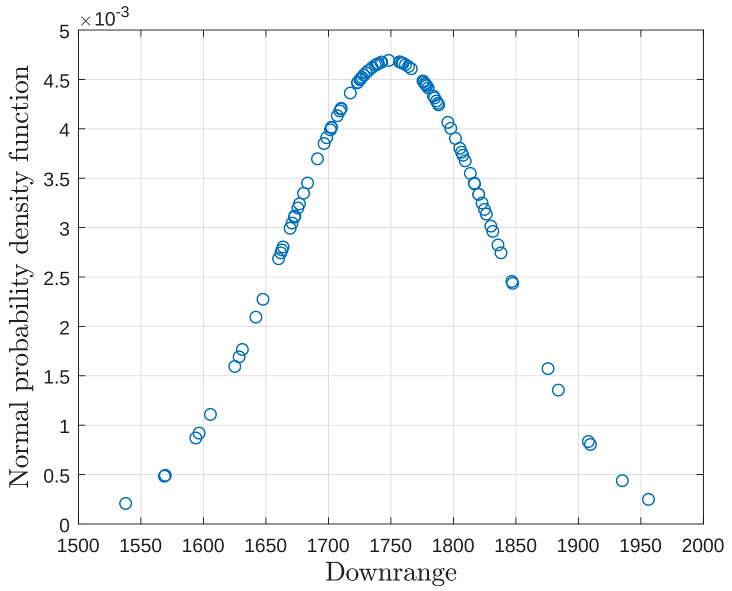
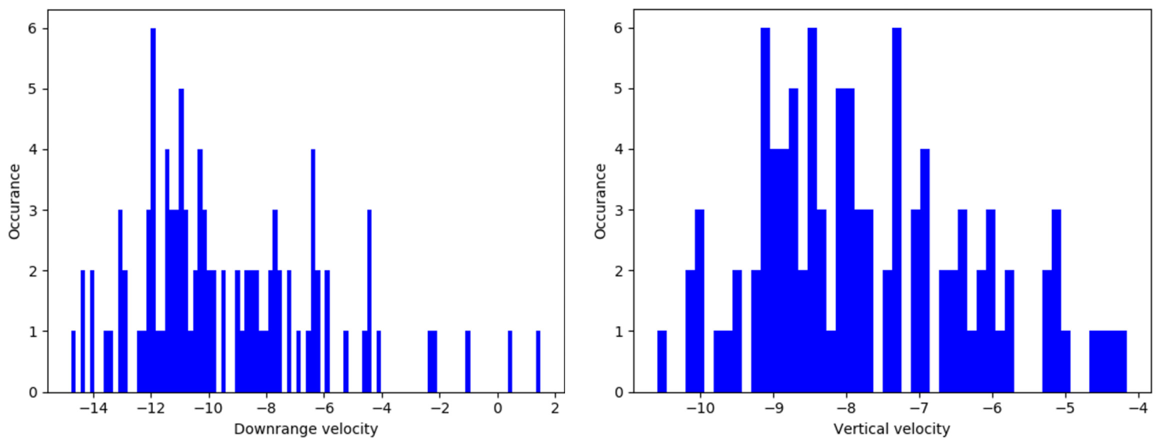


Figure 20: Downrange initial conditions for the Monte Carlo simulation



(a) Final downrange velocity (b) Final vertical velocity

Figure 21: Results of the first Monte Carlo simulations

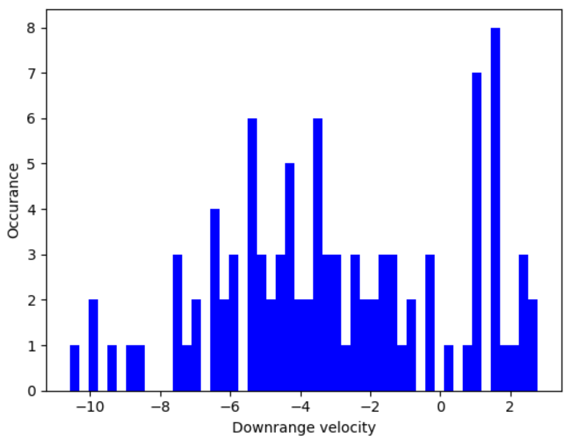
Indeed, the time step must be tuned for each initial condition. There is not a unique value of time step for which the RNN-LSTM provides good results, whatever the initial conditions are. For this reason, a second Monte Carlo simulation has been completed trying to find for each initial conditions the best value of the time step. This has been done in an automatic way using a *for* loop in which many time steps are tried and the one that provides the best results (in terms of final velocities and downrange position) is selected. The results of this second simulation are shown in Fig. 22.

The mean of the final downrange velocity is $-2.97\ \mathrm{m/s}$, while the mean of the final vertical one is $-2.2\ \mathrm{m/s}$. Finally, the mean of the final downrange is $6.23\ \mathrm{m}$. The latter are much lower than the results of the previous Monte Carlo simulation, but are still quite far from the optimal ones ($-0.5\ \mathrm{m/s}$ for both velocities and 0 meters of downrange). In Fig. 23, the final downrange are plotted. Notably, most of the final points shows a final downrange around zero (which is the target).

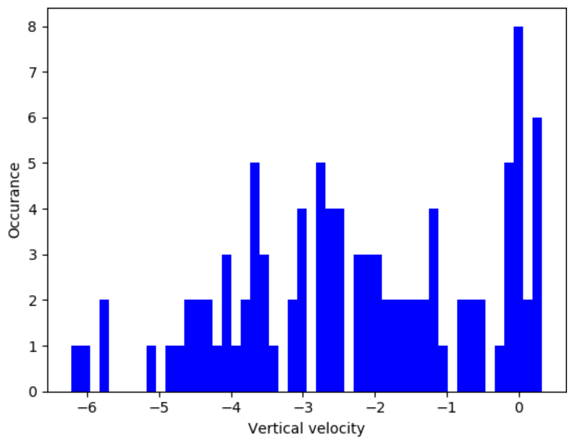
This result is due to the fact that the control applied during the propagation is not very accurate, since the control is kept constant during the integration/propagation phase. Such problem could be in principle solved by asking the net to process the inputs more frequently, so as to have a discrete but more accurate control during the landing. The limit of this approach is represented by the fact that the RN-LSTM network is not trained to take inputs at high frequency. Indeed, the trajectories within the training set do not have an adequate number of points. The considered optimal trajectories comprise only 61 states, i.e. one every 0.9 seconds. One solution would be to train the RNN-LSTM on trajectories with many more points.

CONCLUSIONS AND FUTURE EFFORT

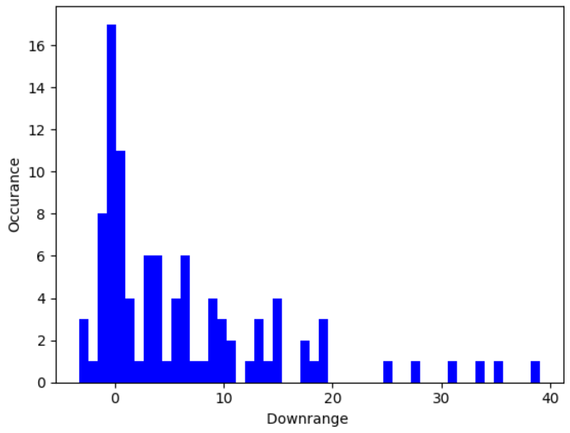
This paper demonstrated how to design and test a deep recurrent architecture capable of imitating a fuel-efficient guidance system for pinpoint planetary landing. Generally, fuel-efficient, closed-loop guidance algorithms are not available and numerical solutions of the propellant-optimal problem can be found only for open-loop trajectories. The proposed RNN-LSTM aims at learning the optimal thrust from sequence of numerically-computed open loop fuel-efficient trajectories.



(a) Final downrange velocity



(b) Final vertical velocity



(c) Final downrange

Figure 22: Monte Carlo second simulation results

The results show that the system achieves high training accuracy both on the training and test set. The recurrent nature of the proposed system efficient process information at previous time-steps to synthesize the current optimal thrust level and thrust direction. The DAgger approach has been implemented to improve accuracy. Monte Carlo simulations show that the system can achieve good performances when deployed in a simulated environment. Future work will be aimed at improving performances via increasing the number of training points and the test the system in a perturbed environment.

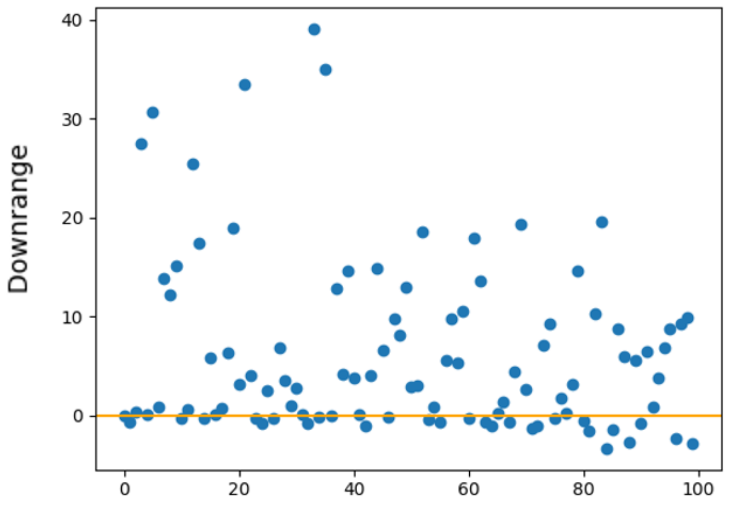


Figure 23: Final downrange distribution

REFERENCES

[1] Klumpp, A.R., Apollo Lunar Descent Guidance, *Automatica*, Vol. 10, No. 2, 1974, pp. 133-146.

[2] Klumpp, A.R., A Manually Retargeted Automatic Landing System for the Lunar Module (LM), *Journal of Spacecraft and Rockets*, Volume 5, No. 2, 1968, pp. 129-138.

[3] Chomel, C., T., and Bishop, R., H., Analytical Lunar Descent Algorithm, *Journal of Guidance, Control, and Dynamics*, Vol. 32, No. 3, 2009, pp. 915-927.

[4] Guo, Y., Hawkins, M., Wie, B. (2013). Applications of generalized zero-effort-miss/zero-effort-velocity feedback guidance algorithm. *Journal of Guidance, Control, and Dynamics*, 36(3), 810-820.

[5] Wibben, D. R., Furfaro, R. (2016). Optimal sliding guidance algorithm for Mars powered descent phase. *Advances in Space Research*, 57(4), 948-961.

[6] Wang, D. Y. Study Guidance and Control for Lunar Soft Landing, *Ph.D. Dissertation*, School of Astronautics, Harbin Institute of Technology, Harbin, China, 2000.

[7] Wibben, D. R., Furfaro, R. (2016). Terminal guidance for lunar landing and retargeting using a hybrid control strategy. *Journal of Guidance, Control, and Dynamics*, 1168-1172.

[8] Liu, X., Lu, P., Pan, B. (2017). Survey of convex optimization for aerospace applications. *Astrodynamics*, 1(1), 23-40.

[9] Acikmese, B., Ploen, S. R. (2007). Convex programming approach to powered descent guidance for mars landing.*Journal of Guidance, Control, and Dynamics, 30(5),* 1353-1366.

[10] Krizhevsky, A., Sutskever, I., and Hinton, G. E. ImageNet classification with deep convolutional neural networks. *In NIPS*, pp. 11061114, 2012.

[11] Socher, R., Bengio, Y., Manning, C. (2013). Deep learning for NLP. *Tutorial at Association of Computational Logistics (ACL), 2012, and North American Chapter of the Association of Computational Linguistics (NAACL).*

[12] Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A. R., Jaitly, N., ... Kingsbury, B. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6), 82-97.

[13] Furfaro, R., Simo, J., Gaudet, B., Wibben, D. (2013, August). Neural-based trajectory shaping approach for terminal planetary pinpoint guidance. *In AAS/AIAA Astrodynamics Specialist Conference 2013 (pp. Paper-AAS).*

[14] Gaudet, B., Furfaro, R. (2014). Adaptive pinpoint and fuel efficient mars landing using reinforcement learning. *IEEE/CAA Journal of Automatica Sinica*, 1(4), 397-411.

[15] Snchez-Snchez, C., Izzo, D. (2018). Real-time optimal control via Deep Neural Networks: study on landing problems. *Journal of Guidance, Control, and Dynamics*, 41(5), 1122-1135.

[16] Patterson, M. A., Rao, A. V. (2014). GPOPS-II: A MATLAB software for solving multiple-phase optimal control problems using hp-adaptive Gaussian quadrature collocation methods and sparse nonlinear programming. *ACM Transactions on Mathematical Software (TOMS)*, 41(1), 1.

[17] Ross, S., Gordon, G., Bagnell, D. (2011, June). A reduction of imitation learning and structured prediction to no-regret online learning. *In Proceedings of the fourteenth international conference on artificial intelligence and statistics* (pp. 627-635).

[18] Furfaro, Roberto and Simo, Jules and Gaudet, Brian and Wibben, Daniel (2013). Neural-based trajectory shaping approach for terminal planetary pinpoint guidance, *AAS/AIAA Astrodynamics Specialist Conference 2013*

[19] Campbell, T., Furfaro, R., Linares, R., Gaylor, D. (2017, January).A deep learning approach for optical autonomous planetary relative terrain navigation.*27th AAS/AIAA Space Flight Mechanics Meeting, 2017. Univelt Inc.*

[20] Gaudet, B., Linares, R., Furfaro, R. (2018). Deep Reinforcement Learning for Six Degree-of-Freedom Planetary Powered Descent and Landing.*arXiv preprint arXiv:1810.08719.*

[21] Furfaro, R., Bloise, I., Orlandelli, M., Di, P. (2018). Deep Learning for Autonomous Lunar Landing. In *2018 AAS/AIAA Astrodynamics Specialist Conference* (pp. 1-22).

[22] Geron A. (2017). Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems. *O'Reilly Media, Inc.*

[23] Kingma, Diederik P and Ba, Jimmy (2011). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*