AAS 18-363

DEEP LEARNING FOR AUTONOMOUS LUNAR LANDING

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

Over the past few years, encouraged by advancements in parallel computing technologies (e.g., Graphic Processing Units, GPUs), availability of massive labeled data as well as breakthrough in understanding of deep neural networks, there has been an explosion of machine learning algorithms that can accurately process images for classification and regression tasks. It is expected that deep learning methods will play a critical role in autonomous and intelligent space guidance problems. The goal of this paper is to design a set of deep neural networks, i.e. Convolutional Neural Networks (CNN) and Recurrent Neural Net-works (RNN) which are able to predict the fuel-optimal control actions to perform autonomous Moon landing, using only raw images taken by on board optimal cameras. Such approach can be employed to directly select actions with-out the need of direct filters for state estimation. Indeed, the optimal guidance is determined processing the images only. For this purpose, Supervised Machine Learning algorithms are designed and tested. In this framework, deep networks are trained with many example inputs and their desired outputs (labels), given by a supervisor. During the training phase, the goal is to model the unknown functional relationship that links the given inputs with the given outputs. Inputs and labels come from a properly generated dataset. The images associated to each state are the inputs and the fuel-optimal control actions are the labels. Here we consider two possible scenarios, i.e. 1) a vertical 1-D Moon landing and 2) a planar 2-D Moon landing. For both cases, fuel-optimal trajectories are generated by soft-ware packages such as the General Pseudospectral Optimal Control Software (GPOPS) considering a set of initial conditions. With this dataset a training phase is performed. Subsequently, in order to improve the network accuracy a Dataset Aggregation (Dagger) approach is applied. Performances are verified on test optimal trajectories never seen by the networks.

INTRODUCTION

The problem of autonomously landing on the lunar surface with pinpoint accuracy is challenging and may require a new class of navigation and guidance algorithms. Indeed, when integrated with sensors and thrusters as part of the on-board lander computing architecture, such algorithms must bring the spacecraft to the desired location on the lunar surface with zero velocity (soft landing) and very stringent precision (e.g., desired position achieved with accuracy less than 10 meters for

*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

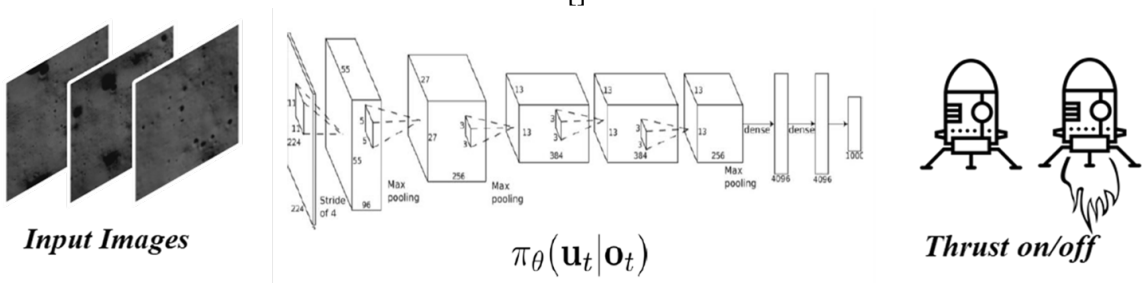
‖Charles Stark Draper Assistant Professor,Department of Aeronautics and Astronautics,Massachusetts Institute of Technology,Cambridge, MA 02139

pinpoint accuracy). Within the overall system architecture, the lander Guidance, Navigation and Control (GNC) subsystem is mainly responsible for safely driving the lander to the surface. GNC functions include a) determining position and velocity of the lander from sensor information (navigation) and b) determine/compute the appropriate level of thrust and its direction as function of the current state (i.e. lander position and velocity). Both guidance and navigation functions are generally designed separately and tend to be different area of research and development. Guidance algorithms are generally comprised of two major segments, i.e. a) a targeting algorithm and b) a trajectory-following, real-time guidance algorithm. The targeting algorithm explicitly computes the reference trajectory that drives the lander to the lunar surface, generally with minimum fuel and satisfying appropriate thrust and path constraints. The real-time guidance algorithm computes that acceleration command that must be implemented by the lander thrusters to track the reference trajectory for a precise and soft landing. The original Apollo real-time targeting and guidance algorithm [1], successfully implemented to land Apollo 11s Lunar Exploration Module (LEM) on the Moon's surface, was based on an iterative algorithm that generated a nominal trajectory consisting in a quartic polynomial. 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]. More recently, research on guidance and targeting algorithms for lunar landing exploited the latest advancement on both optimal and non-linear control. Some examples include gravity-turn based guidance [3], Feedback ZEM/ZEV guidance [4], robust guidance based on time-dependent sliding [5], feedback linearization [6] as well as guidance algorithms based on hybrid control theory [7]. Recently, the ability to generate real-time optimal feedback guidance by solving on-board a sequence of open-loop optimal convex problems has been explored for landing on planetary bodies[**?**]. Guidance algorithms must be tightly integrated with the navigation system which is responsible for determining actual lander position and velocity. On large and small planetary bodies, the most common approach to what is termed Relative Terrain Navigation (RTN), extract the spacecraft state from sequences of optical images. Estimating relative position and velocity from on-board cameras generally rely on extracting and correlating/registering landmarks on the planetary bodies [9] as well as tracking the landmarks during the relative motion (e.g. Natural Feature Tracking,[10]). Over the past few years, encouraged by advancements in parallel computing technologies (e.g., Graphic Processing Units, GPUs), availability of massive labeled data as well as breakthrough in understanding of deep neural networks, there has been an explosion of machine learning algorithms that can accurately process images for classification and regression tasks (e.g., image and video recognition [11], natural language processing [12], speech recognition [13] etc.). However, very little has been done in the space exploration domain to develop machine-learning algorithms for autonomous guidance and navigation tasks. A couple of examples include learning optimal feedback guidance via supervised learning ([14], [16]) and reinforcement learning ([15] as well as RTN via convolutional neural networks. In this paper, we propose a new approach based on deep learning that integrates guidance and navigation functions providing ad end-to-end solution to the lunar landing problem. More specifically, we design a class of Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) capable of learning the underlying functional relationship between a sequence of optical images taking during the descent and the thrust action. The method exploit the ability of CNNs to autonomously extract features and correlate such features to the fuel-optimal thrust action. The system learns in a simulated environment where fuel-optimal trajectories are computed via pseudo-spectral methods and spacecraft position and velocity are correlated directly through ray-tracing simulation to the related optical image taken by the on-board camera.

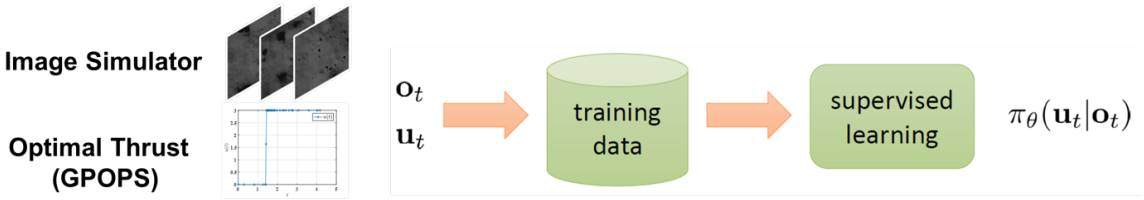
METHODOLOGY

The proposed approach to autonomous lunar landing relies on a combination of deep learning, computational optimal control and ability to generate simulated images of the moon surface. The overall goal is the "teach" a spacecraft to autonomously execute lunar landing by processing a sequence of optical images taken by the spacecraft on-board cameras. The overall approach is to train a set of CNNs and RNNs to map a sequence of optical images $\boldsymbol{o}_t$ into thrust actions $\boldsymbol{u}_t$ (Fig. **??**, (a)). Here, we are looking to learn fuel-optimal guidance policies $\pi_\theta(\boldsymbol{u}_t, \boldsymbol{o}_t)$, i.e. the probability of the thrust action given sequence of optical images. Ultimately, we are interested in imitating the true underlying fuel-optimal guidance policy. With reference to Fig. **??**,(b) we numerically compute a set of optimal trajectories using Gauss pseudo-spectral methods by sampling the initial position and velocity out of a specified distribution. For each state on the optimal trajectory, we simulated the image taken by the on-board camera. Images are synthetically generated by interfacing the camera model with a raytracer that simulates photons reflected out of a realistic Digital Terrain Model (DTM) of a patch on the lunar surface.

[]



(a) Schematic of a Deep Network for Thrust Prediction



(b) Imitation (Supervised Learning) Flow

Figure 1: Network Architecture and Imitation Learning Process

Machine Learning: Deep Neural Networks

Over the past few years, there has been an explosion of machine learning algorithms that can learn from examples, which is the basis of supervised learning. The supervised learning strategy consists of having the desired outputs available for a given set of input signals. Each training sample is comprised of the input signals and their corresponding outputs (*labels*). The goal of the network is to develop a rule,i.e. a procedure that classifies the given data, or alternatively learning the underlying functional relationship between input and output. This is achieved because the learning algorithm supervises the error between the produced outputs (label) with respect to the desired ones. The network is assumed to be trained when such error is within an acceptable value range. The error (or mislabeling for classification problems) is minimized by adjusting continually the synaptic weights and biases of the network. Such error is generally labeled as *loss function*. The agents (the network)

is training and evaluated using two types of sets of data, a *training set* and a *test set*. The training and test sets are composed of the labeled examples. The test set is only made available to verify the performances of the trained network. In this work, we design deep networks to learn the fuel-optimal relationship between navigation (optical) images and the fuel-optimal thrust action. More specifically, we are interested in processing a sequence of images (data) and predict the thrust action via the help of deep network. Here, we considered two types of deep networks, i.e. Convolutional Neural Networks (CNN) and Recurrent Neural Network (RNN).

Convolutional Neural Networks Neural networks and deep learning provide the possibility to find solutions for many problems related to image recognition. In particular, CNNs are used to classify an image or determining the particular content of an image by transforming the original image, through layers, to a class scores. The architecture of a CNN is designed to take advantage of the 2D or 3D $[width, height, depth]$ structure of an input image which is processed as pixels values. The basic CNN structure uses many types of layers which are 1) Convolutional Layer, 2) Pooling Layer, 3) Fully Connected Layer and 4) Output Layer.

Convolutional Layer This layer extracts features from the input volume by applying filters on the image. It is the most demanding layer in terms of computations of a CNN and the layer′s parameters consist of a set of learnable filters. Each filter is basically a matrix spatially smaller than the image which is scanned along width and height (2D case). Importantly, filters (or kernels) are the weights of this layer. In fact, as the filter is sliding on the input image, it multiplies its values with the original pixel values of the image and these multiplications are all summed up giving only one number as output. Repeating this procedure for all the regions on which filter is applied, the input volume is reduced and transformed and then passed to the *Max-pooling layers*.

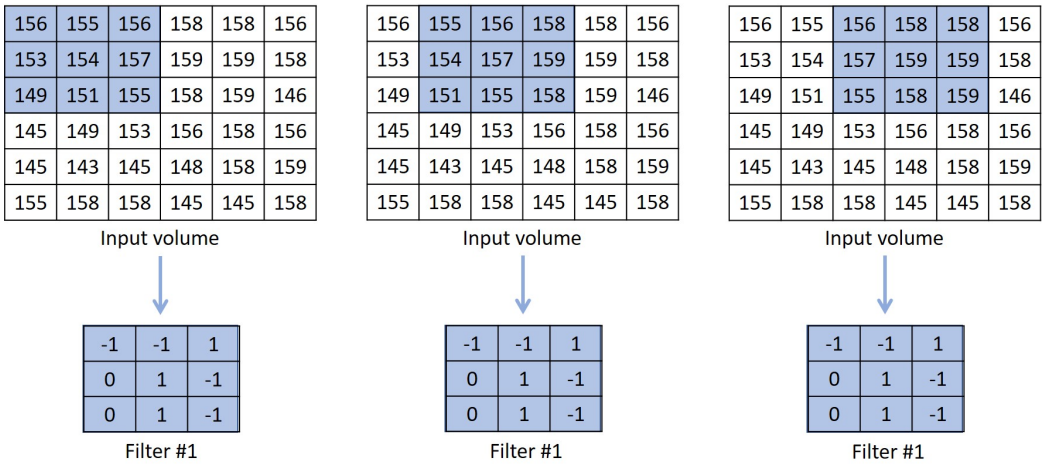


Figure 2: Example of filter application

In the Fig. 2 which represent an example of a filter application, it is possible to note that the filter moves one pixel at a time. The stride in this case is, in fact, equal to 1. This parameter determines how the input image volume reduces and normally, it is set such that the output volume is an integer and not a fraction. Moreover, in order to control the spatial size of the output volumes, an additional parameter is used: thes *Padding* parameter, which pads with zeros the border of the input image. If the padding is not used the information at the borders will be lost after each Conv. layer and this

will reduce both the size of the volumes and the performance of the layer. As a general rule, the hyperparameters necessary for a convolutional layer are: *Number of filters*, *Stride* with which the filter slides, *Padding* with which the input is padded with zeros around the border. An example of image transformed by the application of filters in a convolutional layer is shown in Fig. 3.

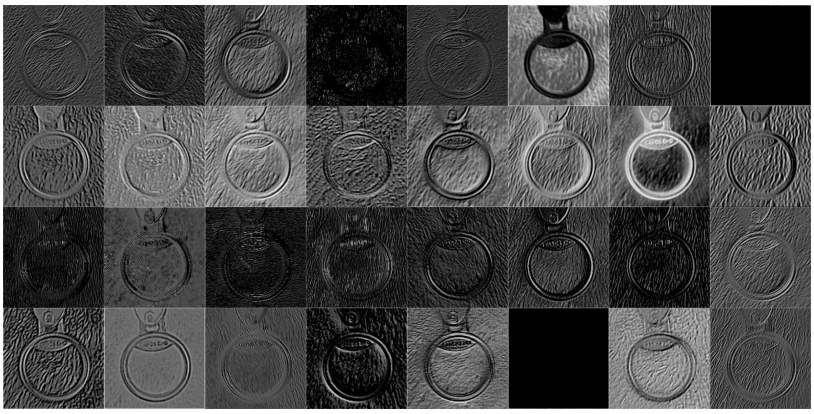


Figure 3: Example of filters application result on a image

Pooling Layer: it reduces progressively the spatial size of the input volume in order to control overfitting. The Pooling Layer operates independently on every depth slice of the input and there are different functions to be used, the common one is the Max-pooling. It uses the MAX operation taking only the most important part. Also for this layer two hyperparameters have to be chosen: the filter window (*F*) and the stride (*S*).

Fully Connected Layer: the purpose of the Fully Connected layer is to use the features arriving from the convolutional layers for classifying the input image into various classes. The last fully-connected layer uses a softmax activation function for classification purpose.

Recurrent Neural Network (RNN) Recurrent Neural Networks (RNNs) are neural networks specifically designed to deal with data sequences. The advantage of an RNN lies in its memory. Indeed, it stores information that have been computed few steps so far and uses them to improve the prediction and its accuracy. Fig. 4 shows an RNN unrolled into a full network to better understand how this net works. The working principle is as follows:

- x_i is the input at the time step t_i
- h_i is the hidden state at time step t_i. It represents the memory of the network which is computed based on the previous hidden state h_{i-1} and the input at the current step x_i.
- y_i is the output at the time step t_i

RNNs are able to use information coming from a long sequence, but experience has shown that they are limited to looking back only a few steps. Considering what said before, the most commonly used RNNs are the LSTM (Long-Short-Term-Memory). The difference is just the way in which it computes the hidden state. The memories in LSTMs are called *cells*. Internally these cells learn what to store in a long-term-state, what to erase from memory, and what to read from it. A typical cell is shown in Fig. 5.

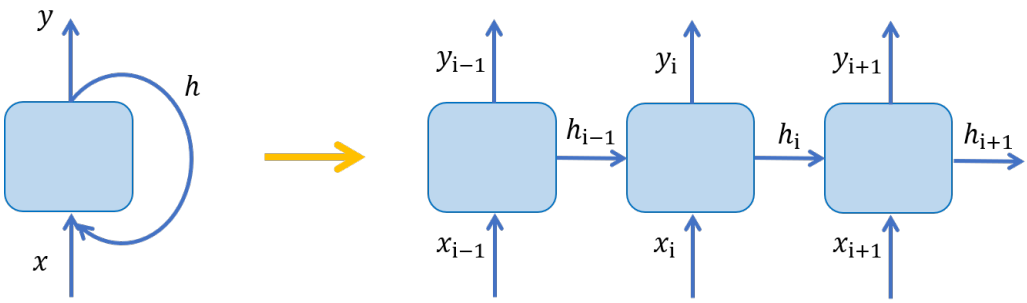


Figure 4: Recurrent neural network

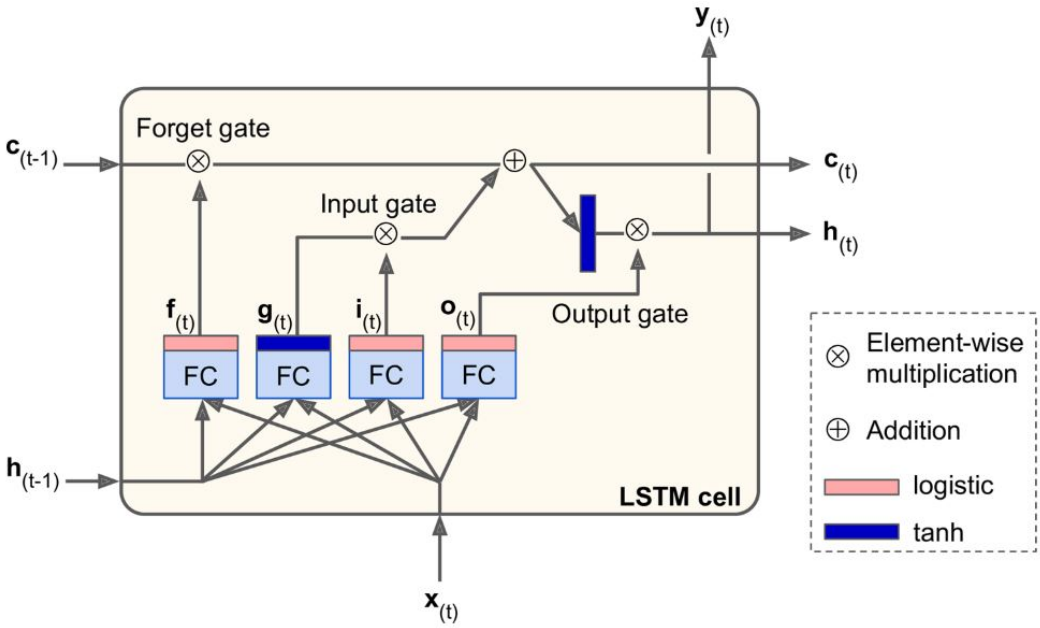


Figure 5: LSTM cell

As shown in Fig. 5, the hidden state is divided in two vector: the long-term state $\mathbf{c}_{(t)}$ and the short-term state $\mathbf{h}_{(t)}$. The incoming long-term state $\mathbf{c}_{(t-1)}$ goes through a *forget gate*, dropping some memories and then it adds some new memories with an adding operation (the memories to store are selected by the *input gate*). The result is $\mathbf{c}_{(t)}$ which is sent straight on without any other transformation. The long-term state $\mathbf{c}_{(t)}$ is copied and filtered by the *output gate* (which applies a *tanh* function on the input) and this operation produces the short-term state $\mathbf{h}_{(t)}$ (note that it is equal to the cell's output $\mathbf{y}_{(t)}$). Let us look at how the current input vector $\mathbf{x}_{(t)}$ is transformed. It is fed with the previous short-term state $\mathbf{h}_{(t-1)}$ to different fully connected layers that have different purposes:

- The main layer is the one with $\mathbf{g}_{(t)}$ as output. It analyzes the current inputs $\mathbf{x}_{(t)}$ and the previous $\mathbf{h}_{(t-1)}$ with a *tanh* function. The output is partially stored in the long-term state.
- The other three layers are called *gate controllers* and use logistic activation function which means that the outputs are in a range from 0 to 1 and, in particular, 0 closes the gate and 1 opens it. In particular:

- The *forget gate* (controlled by $\mathbf{f}_{(t)}$) controls which parts of long-term state should be erased.
- The *input gate* (controlled by $\mathbf{i}_{(t)}$) controls which parts of $\mathbf{g}_{(t)}$ should be added to the long-term state.
- The *output gate* (controlled by $\mathbf{o}_{(t)}$) controls which parts of the long-term state should be read and sent to the output terms ($\mathbf{h}_{(t)}$ and $\mathbf{y}_{(t)}$)

In conclusion, an LSTM cell can learn to recognize an important input, learn to store and preserve it for as long as it is needed and learn to extract information whenever is needed.

Training Set: Optimal Guidance Problem

To train deep networks to learn the fuel-optimal relationship between sequence of images taken during the descent and the thrust action, a set of optimal open-loop trajectories must be computed. The pair state-thrust for the computed optimal trajectories are examples of the functional relationship representing the sought optimal feedback guidance. Once the lander state is associated to the correspondent image taken by the spacecraft, the training set is complete. In its basic form, the fuel-optimal guidance landing problem can be formulated as follows:

$$minJ = \int_0^{t_f} \|\vec{T}\| \, d\tau \tag{1}$$

Subject to the physical constraints:

$$\begin{cases} \dot{\vec{r}} = \vec{v} \\ \dot{\vec{v}} = \vec{g} + \dfrac{\vec{T}}{m} \\ \dot{m} = -\dfrac{\vec{T}}{I_{sp}\, g_0} \end{cases} \tag{2}$$

And the following boundary conditions:

$$\begin{cases} \vec{r}(t_0) = \vec{r_0} \\ \vec{v}(t_0) = \vec{v_0} \end{cases} \quad \text{and} \quad \begin{cases} \vec{r}(t_f) = \vec{r_f} \\ \vec{v}(t_f) = \vec{v_f} \end{cases} \tag{3}$$

Additional path and thrust constraints can be also considered. For a planar 2D Moon landing, ff h is the altitude and d is the downrange, $\vec{x} = [d, h]$ and $\vec{v} = [v_d, v_h]$. The thrust $\vec{T}$ is $\vec{T} = [T_d, T_h]$ and $\mathbf{g}$ is the lunar gravity acceleration. I_{sp} is the specific impulse and g_0 is the reference gravity acceleration. There is no closed-form solution to the general landing guidance problem. Indeed, to compute fuel-optimal solutions, one needs to resort to numerical methods. For a thrust-limited case, it is very well-known that the solution is a singular type (i.e. bang-bang) which makes the problem numerically challenging. Here, we employ the General Purpose OPtimal Control Software (GPOPS II [17]) numerical MATLAB platform to compute the optimal trajectories and generate the appropriate training set.

Image Simulator

In order to generate a dataset of images taken during the Moon landing, a simulator has been developed. The simulator works by synthetically rendering via a ray-tracer and associating each image with the state of the spacecraft at each time step. Importantly, two simulators have been designed and implemented to simulate the camera images for both vertical and planar 2D Moon landing. The simulator has been written in *MATLAB* and the software which has been used to generate every single image is Persistence of Vision Ray Tracer or *POV-Ray*, which is a ray tracing program which generates images from a text-based description of a scene. In a *MATLAB* script, the trajectories coming from GPOPS are loaded and each state is read in a for loop where *POV-Ray* is called to render each image. In order to render an image, *POV-Ray* needs three important objects: the light source position, the camera position (and properties) and the objects to render. The light source, in this case, is the Sun and its position with respect to the Moon has been considered fixed during the entire simulation because the duration of each landing is so short (about one minute) that the variation of the Sun position is not relevant. The camera position, instead, changes at each time step according to the current state generated by GPOPS. In *POV-Ray* the camera position is expressed through a vector of three coordinates. In the 1D case, only the out-of-plane component changes according to the altitude, while the other two components are kept constant equal to 0. In 2D case, the out-of-plane component is conditioned on the altitude and one of the planar components changes according to the downrange, while the third one is kept equal to 0 (Fig. 6). In this way, an image is associated to each state. Since in GPOPS, at each state corresponds a control action, once all the images are generated it is possible to make a dataset in which each image is correlated with the control action. Such dataset is suitable for the training phase of the network. Datasets of more than 6000 images have been generated for both cases under study. The objects to render are described in a *POV-Ray* script in which the DTM and the texture are loaded and scaled according to the image size and to the real altitude range. In this script, the radiosity model and other options regarding the light and the camera properties are implemented. A very important aspect is the size of the image. The size in *POV-Ray* is expressed in pixel and both 1024×1024 and 256×256 images have been generated. First simulations proved that the set made up of the largest images requires excessive computational cost and allocation memory. For this reason, smaller images have been used for the successive simulations. They are quite small but the simulations have revealed that they are sufficiently large to obtain good results.

The images are greyscaled and not RGB: this helps a lot during the render of each image and during the training and test phase of the network, reducing the computational cost and the allocation memory. Moreover, there is no need to have RGB images considering the texture of the Moon surface. *POV-Ray* let the user set up some camera properties. The most important one is the angular field of view. For the research purposes, the camera angular field of view has been fixed at $20°$, which is a reasonable value for on-board navigational cameras. A lower value would have involved a too narrow field of view and a too small portion of the surface would have been seen, especially for the images taken from lower altitudes. In Fig. 7 some examples of the images related to the 1D vertical landing are shown.

DAgger Method to Improve Performance

An approach has been investigated to obtain further improvements of the deep networks 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

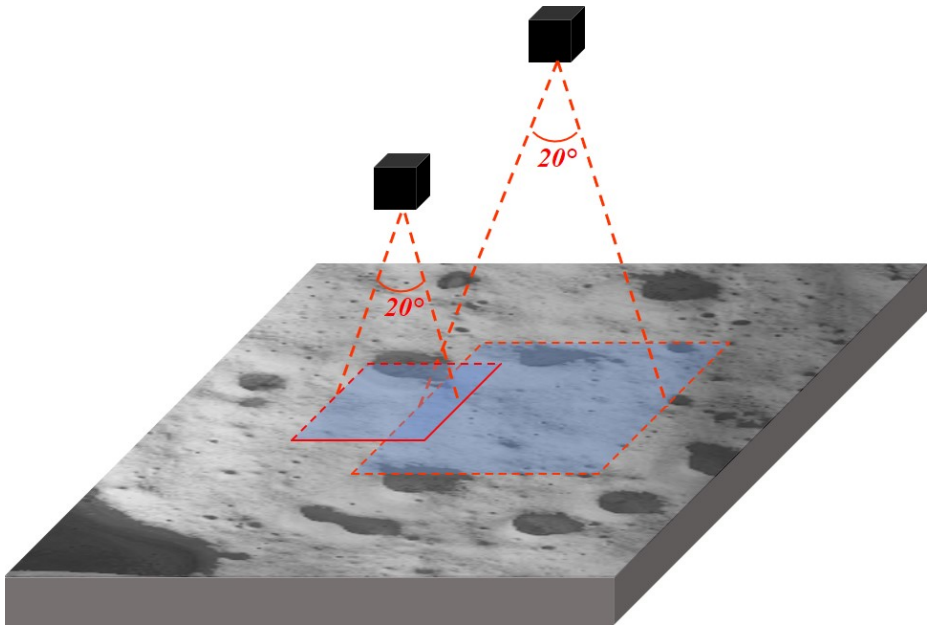
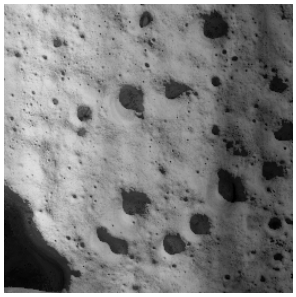
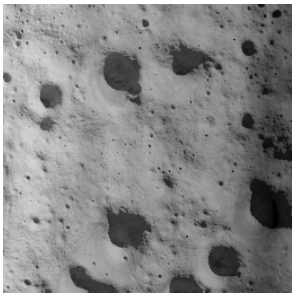


Figure 6: Moon image simulator scheme

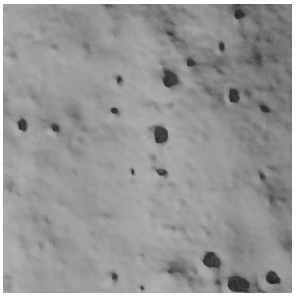


(a) Moon surface seen from 1500 m



(b) Moon surface seen from 986 m

(c) Moon surface seen from 464 m



(d) Moon surface seen from 140 m

Figure 7: Moon surface images taken from different altitudes

order to achieve the best performance. Here, a DAgger (Data Aggregation) approach has been employed. 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([18]):

1. Train the net (in this case a car) on a dataset $\mathbf{D}$ 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, when employed to enable autonomy in cars, the driver corrects online the errors done by the vehicle and record them to augment the database. Since it is not possible to exploit an human action/correction in space, the DAgger approach developed for this work is slightly different. Here, GPOPS is assumed to be the teacher. More specifically, the following steps are followed:

1. Train the net on a dataset $\mathbf{D}$ generated with GPOPS.
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 trajectories with error higher than the threshold in $\mathbf{D}_{wrong}$.
5. Use $\mathbf{D}_{wrong}$ to re-train the net and to improve the performances.

This iterative algorithm is specifically designed to train stationary deterministic guidance policies and it is generally shown to outperform previously developed approaches for imitation learning ([18])

IMPLEMENTATION AND RESULTS

In this section we describe the design, training and testing of two classes of deep networks specifically trained to learn fuel-optimal thrust policies from sequence of images taken during the powered descent toward the lunar surface. We considered a vertical (1D) lunar landing and a planar (2D) landing.

Vertical Lunar Landing: CNN Architecture, Training and Performance

The initial development consider a simpler vertical landing. Here a CNN has been designed, trained and tested to execute an autonomous vertical descent. Here, we discuss the dataset generation, the CNN architecture, training and testing phases.

Dataset generation A suitable dataset (i.e. training set) is needed for imtation (supervised) learning). In order to generate a such dataset, GPOPS has been employed solving the fule-optimal trajectories many times starting from different initial conditions.For this specific case, the initial mass of the spacecraft has been set equal to $1300\ \mathrm{kg}$. The altitude has been initialized between 1000 and 1500 meters. The initial vertical velocity v_{h_0} changes according to the altitude: when h is maximum, the velocity is maximum in modulus ($-11\ \mathrm{m/s}$), vice versa when h is minimum, the velocity also is minimum in modulus ($-6\ \mathrm{m/s}$). Unlike the initial conditions, the final ones have been kept constant for all trajectories. In particular, the final altitude h_f is equal to 50 meters and the final velocity is equal to $-0.5\ \mathrm{m/s}$. As it can be seen, the final condition is such that the spacecraft has not touched yet the ground and has a very small velocity, directed toward the ground. The nominal thrust T_{nom} is equal to $4000\ \mathrm{N}$. The maximum (allowable) thrust has been fixed equal to the 85% ($3400\ \mathrm{N}$) of the nominal thrust and the minimum (allowable) equal to 25% ($1000\ \mathrm{N}$). Because of the bang-bang nature of the problem, the network will have to perform only a classification on the thrust magnitude. In this way, 101 trajectories have been generated within the selected range of altitude.The parameters of the problem are shown in Tab. 1a.A summary of the initial conditions and one of the final ones are shown in Tab. 1b and in Tab. 1c.

Table 1: Parameters and state values for 1D problem

(a) Problems parameters

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

(b) 2D initial conditions

	value	unit
h_0	$[1.0,\ 1.5]$	km
v_{h_0}	$[-6, -10]$	m/s
m_0	1.3	ton

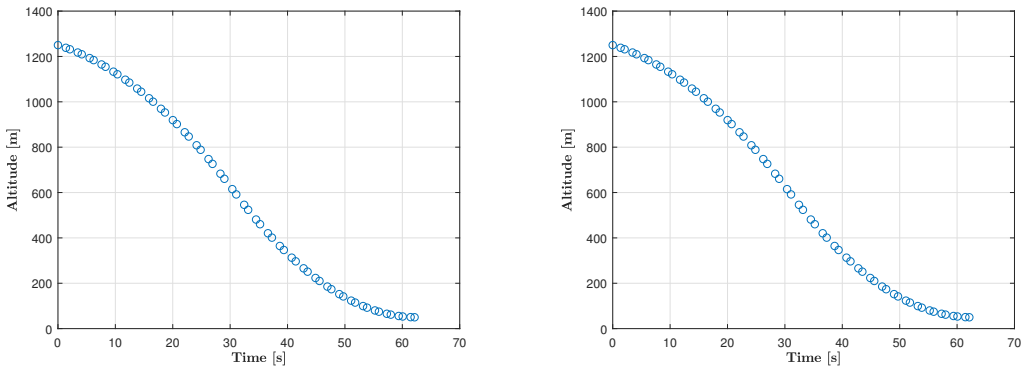
(c) 2D final conditions

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

Each trajectory computed by GPOPS has 61 points, resulting in 61 states and 61 control actions. For the vertical landing problem, each states has three elements, i.e. altitude, velocity and mass; each control action has only one component, i.e. the magnitude. The final dataset has been built by associating each state to each control action. This dataset has been divided in training set and test set; the first one is comprising 81 trajectories, the second one 20 trajectories.
Fig. **??** shows the altitude history for a sample trajectory in the dataset. In this case, the initial altitude is equal to 1250 meters. Fig. **??** shows the plot of the thrust magnitude (bang-bang) profile.

Network Architecture The CNN architecture has been defined via a systematic trial-and-error process. We explored and compared three types of architectures with different numers and type of layers. After analyzing and testing a variety of candidates, we converged to the following CNNs. The selected CNN is (Fig. 9) is comprised of 5 convolutional layers with following parameters:

- First Convolutional layer, $\left[\ 36\ \textit{filters}, \textit{size}\ 3 \times 3, \textit{stride}\ 2\ \right]$.
- Second Convolutional layer, $\left[\ 36\ \textit{filters}, \textit{size}\ 3 \times 3, \textit{stride}\ 4\ \right]$.



[]

(a) Altitude (b) Thrust

Figure 8: Optimal Altitude and Thrust History for the Lunar Vertical landing

- Third Convolutional layer, $\left[\ 72\ \textit{filters}, \textit{size}\ 2 \times 2, \textit{stride}\ 2\ \right]$.
- Fourth Convolutional layer, $\left[\ 72\ \textit{filters}, \textit{size}\ 2 \times 2, \textit{stride}\ 2\ \right]$.
- Fifth Convolutional layer, $\left[\ 72\ \textit{filters}, \textit{size}\ 2 \times 2, \textit{stride}\ 1\ \right]$.
- First Fully connected layer, $\left[\ 256\ \text{neurons}\ \right]$.
- Second Fully connected layer, $\left[\ 128\ \text{neurons}\ \right]$.
- Output layer, $\left[\ 2\ \text{classes}\ \right]$.

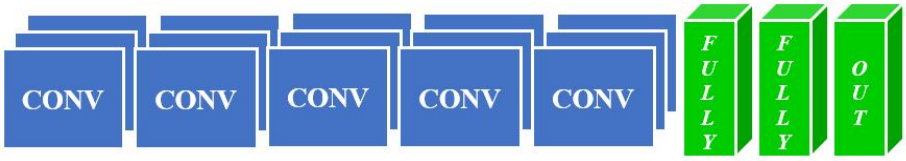


Figure 9: Selected CNN architecture

Tests have shown that better results can be achieved with network deprived of max-pooling layers. In fact, this kind of layer eliminates critical information needed by the network to be properly trained and to extract features for this specific application.Training phase and more details about third CNN results are described in following the section.

Training and Testing Phase The selected CNN has been trained to learn the correct thrust action needed to perform a fuel-optimal vertical 1D landing on the Moon. For this purpose, the states, coming from the dataset described above has been used as input for the 1D Moon images simulator. Here, a new dataset has been created by associating all the images to the relative control actions. Each input is comprised of three consecutive superimposed images arranged as input layers in the CNN. Since each image represents a static picture of the Moon surface, one image alone will not convey any information about the velocity of the lander. Feeding three sequential images, the CNN will be equipped with the ability to keep track of the lander velocity, hence improving the overall

accuracy. The sequence of images, belonging to each trajectory (61 states), has been divided in sets of three images. according to which each trajectory, is separated from the other. Since the fuel-optimal solution is a bang-bang type and since the thrust direction is only vertical, for this specific problem, the CNN has to only solve a classification problem, i.e. select a binary (min-max) value for the thrust as function of the lander position and velocity. Binary vectors of length two have been used to represent the two classes. When the thrusters are at their maximum, the correct label is $[0,\ 1]$ whereas the label is $[1,\ 0]$ at the minimum thrust value. Each input package of three sequential images has been associated to only one output label. From a control point of view, at time *t* the control action is predicted taking the image of the current state and those of state $t-1$ and $t-2$. The network has all the information contained in three consecutive frames to predict the correct label.

Before entering the CNN, all images are normalized between 0 and 1 with respect to the maximum value among all the images in the dataset. Each image enters the net as a square matrix 256×256. Therefore, the input feeds are three-dimensional matrices $3 \times 256 \times 256$ [number of images $\times$ image size $\times$ image size]. In order to train the network on an entire trajectory at each iteration, a batch of 59 inputs has been considered (also called mini-batch). Accordingly, 59 sequential input packages and 59 correct labels enter the net at each iteration during the training phase.

During the training phase, the cross-entropy for the binary problem (loss function) is minimized. The cross-entropy loss can be described as follows:

$$y_{cross-entropy} = -\sum_{j} t_j \log(y_{softmax_j}) \tag{4}$$

Where the softmax function is defined as follows:

$$y_{softmax} = \frac{e^{z_k}}{\sum_{j=1}^{J} e^{z_j}} \tag{5}$$

While implementing the Stochastic Gradient Descent (SGD) process, the Adam optimization algorithm is employed. The learning rate is adjusted as training unfolds by the mean of the decay parameter (i.e. another hyperparameter). The CNN has been developed and trained in a *Python-Keras* and *Pyhton-Tensorflow* environment. Since the network has to solve only a classification problem, there was no need to weight the loss function values associated to the model outputs.

In Tab. 2 the chosen hyperparameters are summarized.

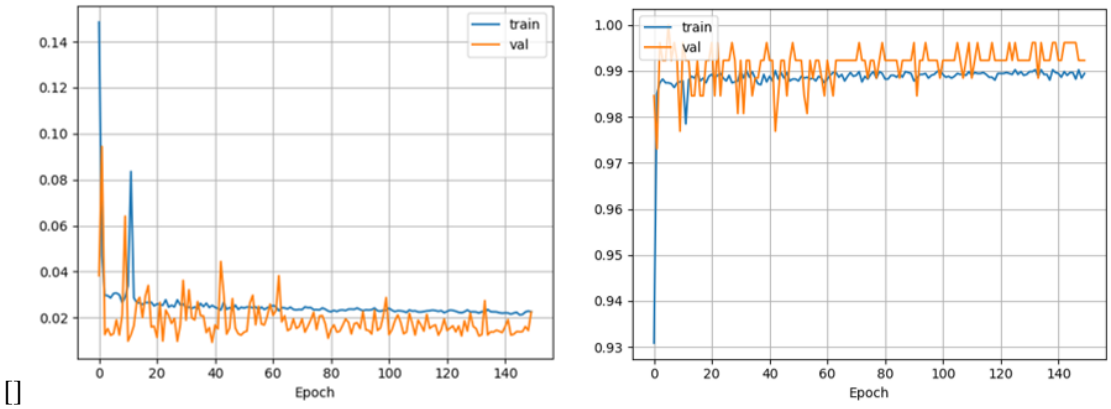
Table 2: Summary of the CNN 1D hyperparameters

Hyper-parameters	
Batch size	59
Initial learning rate	0.001
Decay rate	0.0001

Classification predictions can be evaluated using accuracy and the loss function associated is the *Cross-entropy loss* (Eq. 4) which indicates the distance between the model prediction and the true value of the output and where softmax function outputs a probability distribution (Eq. 5).

The training has been performed using the training set with 4941 images ($61\ \text{state} \times 81\ \text{trajectories}$). The network has been trained in 200 epochs, using in each one a batch of 59 inputs. The training set has been additionally split, as 5% of the dataset is made available for the validation during the training phase.

In Fig. **??** both loss and accuracy trends during the training phase are reported. The training phase has been performed with the help of a High Performance Computing (HPC) systems of the University of Arizona and it has taken about 12 hours to complete the training phase. Clearly, the hyperparameter space can be explored to reduce the loss function. Indeed, after 200 iterations, both accuracy and loss functions are still relatively noisy. The later implies that either the minimum has not been reached yet, or that the hyperparameters might require further tuning. Nevertheless, good performance are already achieved without further tuning.



[]

(a) Loss function trend

(b) Accuracy trend

Figure 10: Loss function and accuracy results during CNN 1D training phase

Next, the trained CNN is tested on trajectories outside the training set. The test set is made by 20 optimal descent trajectories. To visualize the trained net performances on the classification (accuracy), the corresponding confusion matrix has been reported in Fig. 11

Importantly, the CNN trained model has an accuracy of $97,63\%$. Next section will implement an approach to further improve the accuracy.

Accuracy improvement: DAgger approach After the training and test phase of the CNN, the DAgger method has been implemented to enhance the accuracy on the predictions. Since the global accuracy on test set performed by the trained CNN was found to be 97.63%, the trajectories predicted with an accuracy $< 98\%$ have been picked up from the test dataset. Indeed, 8 out of the 20 belonging to the test set exhibit a performance lower than 98%. To implement DAgger we aggregate the training test with the less accurate trajectories and then re-train the CNN on the new enlarged dataset (Fig. 12).

Finally the result for the applied strategy reaches an accuracy of 99.15% (Fig. 13). Indeed, the accuracy of the CNN on the predictions has been significantly improved.

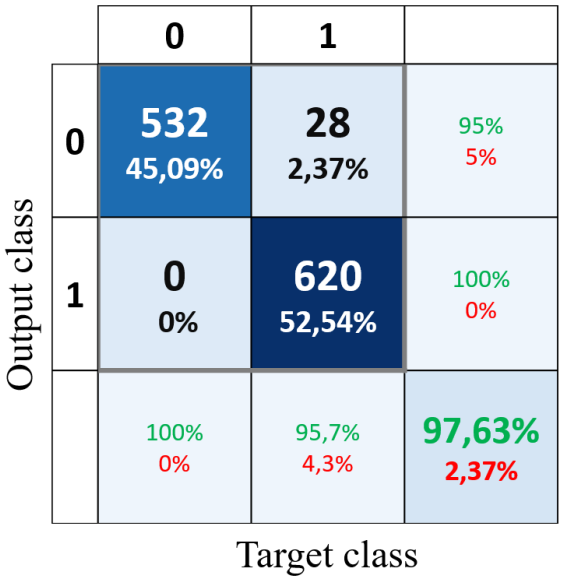


Figure 11: Confusion matrix CNN 1D

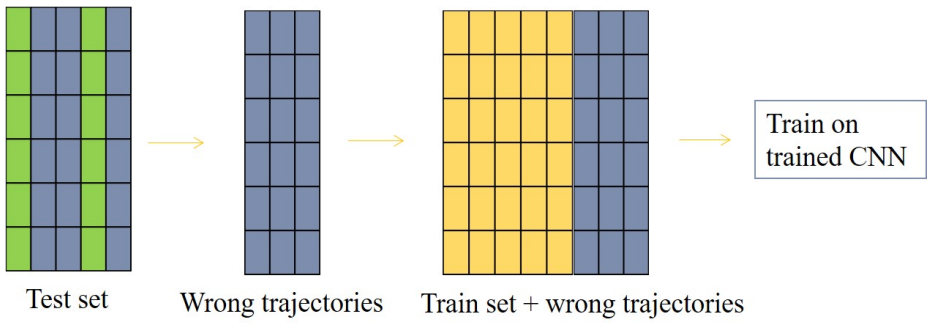


Figure 12: Applied DAgger strategy for CNN 1D

Planar Lunar Landing: RNN-LSTM-CNN Architecture, Training and Performance

Finally, we consider training a deep networks to execute a fuel-optimal planar (2D) landing using optical images. Here we considered a specific deep architecture that combines the features of RNN−LSTM networks with the CNN. The approach is similar to the one followed in the vertical lunar landing in the sense that optimal trajectories are computed via GPOPS and images of the moon surface are simulated as function of the lander position and velocity to generate the training set. However, in this case the deep network has to both learn thrust level (binary classification) and thrust direction. Importantly, we use a RNN architecture to capture the sequential nature of the problem, where the thrust action (magnitude and direction) is predicted as function of the previous states.Here, we discuss the dataset generation, the RNN-LSTM-CNN architecture, as well as training and testing phases.

Dataset Generation As for the vertical landing case, GPOPS has been employed to generate fuel-optimal trajectories. Here, initial conditions as drawn from a uniform distribution have been

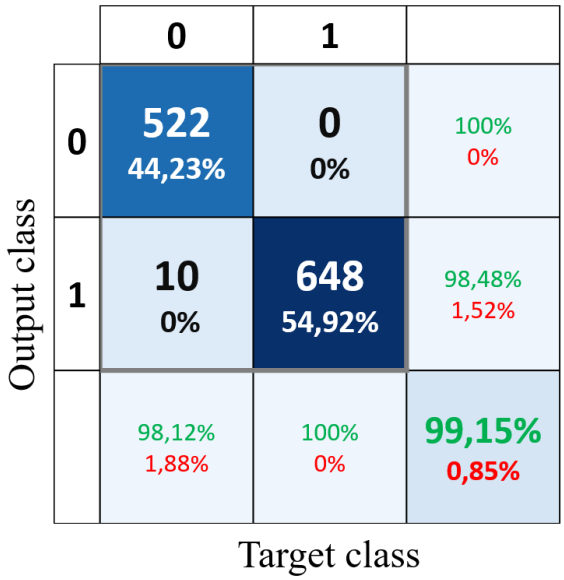


Figure 13: Confusion matrix after DAgger approach for CNN 1D

considered. 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 ranges between 1000 and 1500 meters. Fig. **??** shows a scheme 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), vice versa when 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 goes 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. Importantly, the final condition is such that the spacecraft has not touched the ground and has a very small downward velocity. The propulsion system is assumed to throttable with a nominal maximum thrust T_{nom} of 4000 N. The 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). Subsequently, 2601 trajectories have been generated within the selected two-dimensional portion of space. The parameters of the problem are shown in Tab. 3a. A summary of the initial and the final conditions is shown in Tab. 3b and in Tab. 3c.

Importantly, each trajectory computed by GPOPS has 61 points, i.e. 61 states (position and velocity) and 61 thrust actions. Each state is comprised of five elements (downrange, altitude, velocities and mass). Each control action has three components (magnitude, u_d and u_h). The thrust direction angle θ has been extracted for each couple of unit vector components, i.e. 61 angles have been computed per each trajectory. Therefore the thrust action is a vector with two components, i.e. magnitude and thrust angle. The final dataset has been built by associating each state to each control action. This dataset has been divided in training-set and test-set. the training set comprises 2409 trajectories, the test set 192. The image dataset has been generated as discussed in the vertical landing case.

Table 3: 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	$[1.5,\ 2.0]$	km
h_0	$[1.0,\ 1.5]$	km
v_{d_0}	$[-11,\ -15]$	m/s
v_{h_0}	$[-6, -10]$	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

Proposed Network Architecture The aim of the proposed deep network is to employ a sequence of images taken by the on-board camera to directly predict the fuel-optimal thrust action in a feedback fashion. Since with 2D problem (i.e. lander constrained to move on a vertical plane), the thrust action associated with each image sequence has two components: one for the thrust magnitude and one for the thrust angle (i.e. thrust direction). The proposed deep architecture is shown in Fig. 14. The incoming input (sequence of three consecutive images) is processed sequentially first by a CNN and then by a RNN−LSTM which are linked by a fully connected layer. The processed output of RNN−LSTM feeds two different branches where one branch performs a classification task classification and the other performs a regression task.

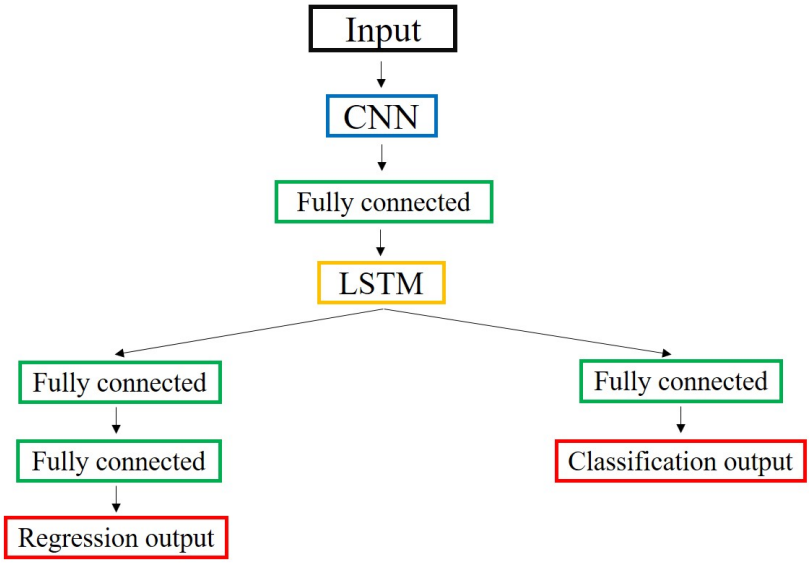


Figure 14: Proposed deep RNN for planar landing

The network is implemented in *Python-Keras* and it is comprised by:

- Input layer which takes three consecutive images at a time.

- Convolutional neural network (Fig. 15).

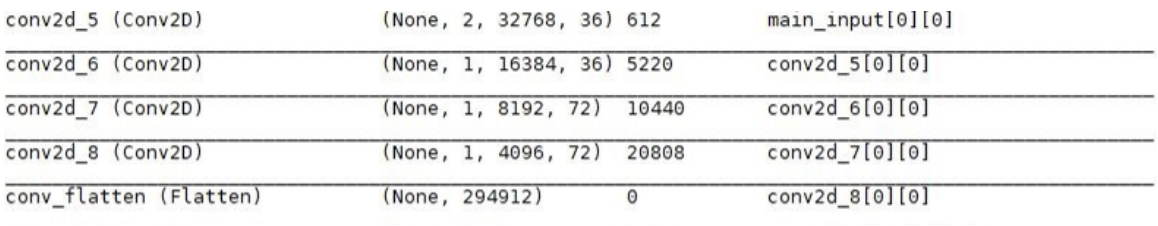


Figure 15: CNN melted in deep RNN network in *Pyhton-Keras*

Note that, the hyperparameters for the Convolutional layers have been tuned differently from the CNN 1D case. Importantly, the five (5) convolutional layers have been employed:

  - First Convolutional layer, $\left[\, 36 \textit{ filters}, \textit{size } 4 \times 4, \textit{stride } 2 \,\right]$.
  - Second Convolutional layer, $\left[\, 36 \textit{ filters}, \textit{size } 2 \times 2, \textit{stride } 2 \,\right]$.
  - Third Convolutional layer, $\left[\, 72 \textit{ filters}, \textit{size } 2 \times 2, \textit{stride } 2 \,\right]$.
  - Fourth Convolutional layer, $\left[\, 72 \textit{ filters}, \textit{size } 2 \times 2, \textit{stride } 2 \,\right]$.

  The last layer (flatten layer) allows to transform the CNN outputs in a row. This layer is necessary to link the CNN with the LSTM cell.

- Fully connected layer which is followed by a reshape layer as shown in line `dense_2` and line `reshape_2` in Fig. 16

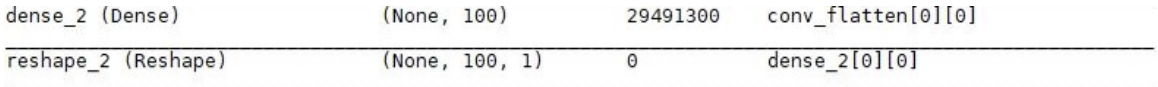


Figure 16: Input reshape layer before LSTM cell in *Pyhton-Keras*

These two layers have been used to reshape the input as requested for LSTM cell.

- Long-short-term-memory cell as decribed in the RNN section above

The proposed deep architecture results in 29,574,483 trainable parameters.

Training and test phase The training and testing phase follows the same lines as described in the CNN case. We employ SGD with Adams optimizer. The loss finction is the cross-entropy for the thrust magnitude (classification) and MSE for the thrust angle (regression). The mini-batch size has been set to 59. The original images (with a dimension 256×256) have been normalized between 0 and 1 and reshaped such that the input pack is a matrix 3×65.536 [number of images $\times$ pixels per image]. The latter is needed because each image is transformed in a row. The associated label (which consists of the thrust magnitude and the thrust angle) is the one corresponding to last image of each input matrix. Since the training phase is computationally demanding, the max epoch has been set to 200 while using only 80 trajectories (4880 images). Given that the number of trajectories for train is lower, no validation split has been done on dataset. The simulation have

been performed using the HPC (High Performance Computing) systems of University of Arizona, like Extremely LarGe Advanced TechnOlogy (El Gato), which uses specially designed hardware to achieve high performance economically, including NVIDIA K20X GPUs and Intel Xeon Phi 5110p Coprocessors. The selected hyperparameters are reported in Tab. 4.

Table 4: Summary of the Deep RNN-LSTM-CNN hyperparameters

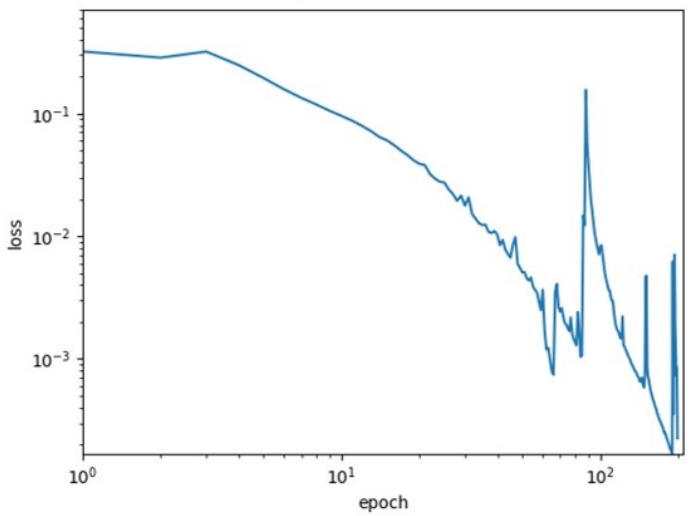
Hyper-parameters	
Batch size	59
Initial learning rate	0.001
Decay rate	0.0001
Regression loss weight	10
Classification loss weight	60

The results of training phase are shown in Fig. 17.

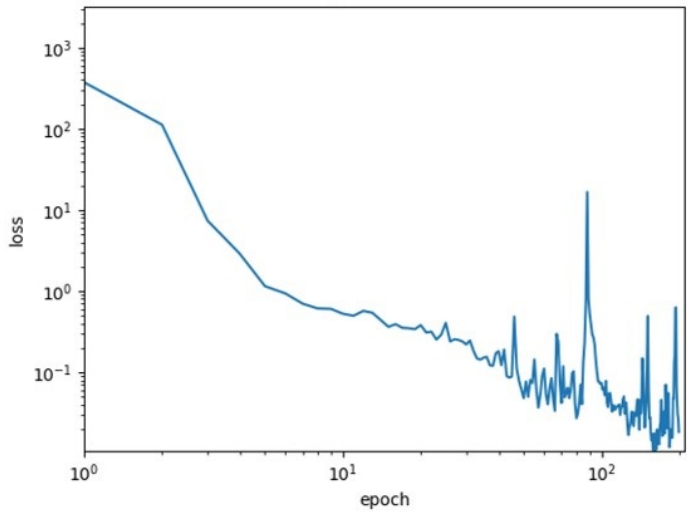
Notably, the training phase is noisy enough to affirm that no convergence to a global minimum has not been achieved yes, although as shown below, the network achieves good performances . This is due to the complexity of the training problem. In general, one can achieve less noisy loss functions during trainign by by further exploring the hyperparameter space to select values that ensure lower and smoother losses. This could lead to an increase of number of training epochs, a lower initial learning rate or a more precise choice of loss weights. Moreover, while in the regression a final flat trend can be highlighted, the classification loss may still require additional training epochs. Furthermore in the graphs (Fig. 17) some clear peaks are present and they are due to the change in the learning rate during the training phase. Although the model can be further trained to achieve higher performance, the results are very promising. The testing phase considered 30 fuel-optimal trajectories that have not been employed uring the training phase. The resulting confusion matrix is reported in Fig. 18.The accuracy on the thrust classification reaches the $98,51\%$ and the RMSE = 0.76° (Fig. 19).

CONCLUSIONS AND FUTURE EFFORT

To the best of our knowledge, this is the first attempt to design a set of integrated deep networks that can predict the fuel-optimal thrust magnitude and direction directly from a sequence of optimal images taken by the on-board camera system. A combination of deep CNNs and RNN-LSTMs can be trained to find the fuel-optimal guidance policy in a simulated but realistic environment. Such approach require the ability to 1) compute off-line open-loop, fuel-efficient trajectories and 2) simulate the optical images taken from an on-board camera along the optimal descent toward the lunar surface. Thus, a training set can be generated and deep networks can be trained to imitate the optimal guidance policy, i.e. learn the functional relationship between sequence of images and optimal thrust actions. We built two classes of networks of increasing complexity as function of the descent problem. In the first case, we considered a vertical landing problem where the spacecraft drops vertically in a fuel efficient fashion toward the Moon surface. In this case, a CNN has been trained and tested to predict the thrust level. The DAgger method has been implemented to achieve 99.15% accuracy on the testing set. In the second scenario, we considered a planar (2D) lunar landing problem where the spacecraft move to the desired state in a vertical plane. Here, we



(a) Classification loss of the deep RNN-LSTM-CNN during training



(b) Regression loss of the deep RNN-LSTM-CNN during training

Figure 17: Classification and regression losses evolution during training phase of the deep RNN-LSTM-CNN

designed and tested a RNN-LSTM-CNN network capable of predicting both thrust magnitude and thrust direction as function of a sequence of images. Performances on the testing set show that the network can achieve an accuracy of 98.51% on thrust level prediction and RMSE of 0.76^o on the thrust direction. Although deep networks are computationally expensive to train off-line, we have showed that deep networks can be in principle employed to effectively tackle the problem of autonomous landing.

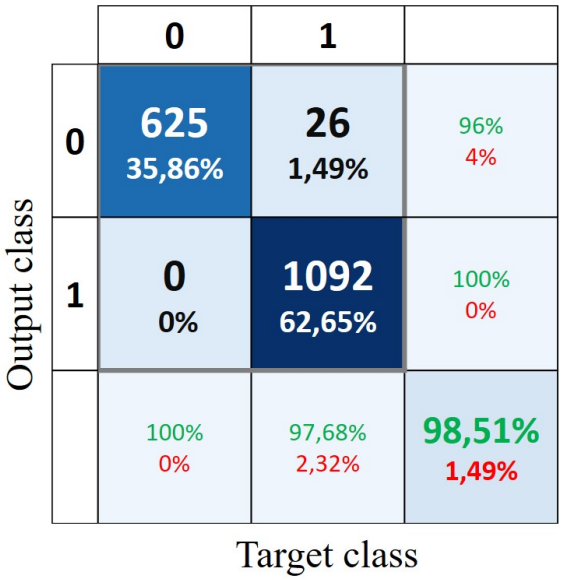


Figure 18: Confusion matrix for the deep RNN-LSTM-CNN Thrust Magnitude

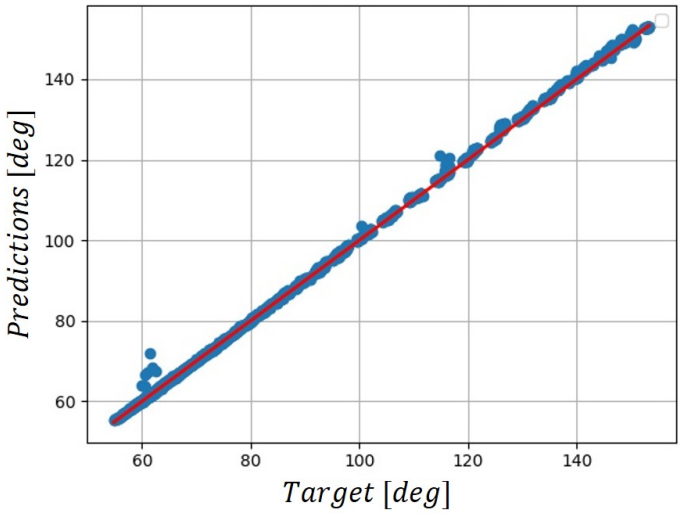


Figure 19: Regression for the deep RNN-LSTM-CNN Thrust Angle Predicition

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] Gaskell, R. W., BarnouinJha, O. S., Scheeres, D. J., Konopliv, A. S., Mukai, T., Abe, S., ... Kawaguchi, J. (2008). Characterizing and navigating small bodies with imaging data. *Meteoritics Planetary Science*, 43(6), 1049-1061.

[10] Lorenz, D. A., Olds, R., May, A., Mario, C., Perry, M. E., Palmer, E. E., Daly, M. (2017, March). Lessons learned from OSIRIS-Rex autonomous navigation using natural feature tracking. *In Aerospace Conference, 2017 IEEE (pp. 1-12). IEEE.*

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

[12] 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).*

[13] 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.

[14] 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).*

[15] 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.

[16] 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.

[17] 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.

[18] 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).