Image-based Deep Reinforcement Learning for Autonomous Lunar Landing

Image-based Deep Reinforcement Learning for Autonomous Lunar Landing

Andrea Scorsoglio* and Roberto Furfaro †
Department of System & Industrial Engineering, University of Arizona, 1127 E. James E. Rogers Way, Tucson, AZ 85721

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

Brian Gaudet§
Department of System & Industrial Engineering, University of Arizona, 1127 E. James E. Rogers Way, Tucson, AZ 85721

Future missions to the Moon and Mars will require advanced guidance navigation and control algorithms for the powered descent phase. These algorithm should be capable of reconstructing the state of the spacecraft using the inputs from an array of sensors and apply the required command to ensure pinpoint landing accuracy, possibly in an optimal way. This has historically been solved using off-line architectures that rely on the computation of the optimal trajectory beforehand which is then used to drive the controller. The advent of machine learning and artificial intelligence has opened new possibilities for closed-loop optimal guidance. Specifically, the use of reinforcement learning can lead to intelligent systems that learn from a simulated environment how to perform optimally a certain task. In this paper we present an adaptive landing algorithm that learns from experience how to derive the optimal thrust in a lunar pinpoint landing problem using images and altimeter data as input.

I. Introduction

Pinpoint soft landing on planetary bodies is becoming increasingly important for future human exploration. The task is generally cumbersome 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. Within the overall system architecture, the lander Guidance, Navigation and Control (GNC) subsystem is mainly responsible for safely driving the lander to the surface. The task that GNC must carry out are 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). In particular, guidance algorithms generally comprise two major segments, i.e. a) a targeting algorithm and b) a trajectory-following, real-time guidance algorithm. The targeting algorithm 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. In general the first task is computed off-line on the ground. The real-time guidance algorithm than follows it. The desirable architecture though is an on-line algorithm that can compute on board and in real time both the target trajectory and the trajectory-following guidance. In this regard, some examples of closed loop landing guidance are the feedback ZEM/ZEV guidance [1], robust guidance based on time-dependent sliding [2], feedback linearization [3] as well as guidance algorithms based on hybrid control theory [4]. Recently, the ability to generate real-time optimal feedback guidance by solving on-board a sequence of open-loop optimal convex problems has also been explored for landing on planetary bodies[5].

Another important factor when designing a guidance algorithm for these kind of problems is the ability to cope with uncertain environment (i.e. uncertain gravity field), noisy measurements and actuator or sensor failure. This shows why it's important to create adaptive algorithms that are both accurate and able to work in these conditions. In the area

*PhD Student, Department of System & Industrial Engineering, University of Arizona, 1127 E. James E. Rogers Way, Tucson, AZ 85721
†Professor, Department of System & Industrial Engineering, University of Arizona, 1127 E. James E. Rogers Way, Tucson, AZ 85721
‡Professor, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, MA, 02139, USA
§Department of System & Industrial Engineering, University of Arizona, 1127 E. James E. Rogers Way, Tucson, AZ 85721

of adaptive guidance there have been some examples, mainly tackling the problem of external disturbances and fault identification. Although interesting, these methods rely on simplified models [6] and often require online computations that may be feasible to be implemented in a real time guidance algorithm [7]. On the other hand, Reiforcement Learning (RL) has shown good performance in dealing with uncertain and complex dynamics. In general this works by interacting with an environment with randomized parameters and learning the temporal variation of the states of the system as function of the changing environment. This results in a deployed policy whose internal states allow it to quickly adapt to the dynamics of the environment and complete the task optimally. Several works have demonstrated the power of RL with applications to robotic motion tasks REF. In particular for dynamic environments, meta Reinforcement Learning (meta-RL) [8, 9] has shown good performance, especially in aerospace application [10–14]. It has been demonstrated that an agent trained using meta-RL learns how to react to a dynamic environment both in terms of uncertain parameters (gravitational acceleration, mass of the spacecraft) and failure in the system itself (actuator failure).

In general guidance algorithms must also be integrated with the navigation system which is responsible for determining lander position and velocity. The most common approach to what is termed Relative Terrain Navigation (RTN) is based on the extraction of 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 [15] as well as tracking the landmarks during the relative motion (e.g. Natural Feature Tracking,[16]). Over the past few years, encouraged by advancements in parallel computing technologies and GPUs, there has been an increase interest in machine learning algorithms that can accurately process images for classification and regression tasks (e.g., image and video recognition [17], natural language processing [18], speech recognition [19] etc.). However, in the space exploration domain these techniques are still struggling to be recognized and there are only a few examples of machine-learning algorithms for autonomous guidance and navigation tasks. These examples include learning optimal feedback guidance via supervised learning [20, 21] and reinforcement learning [22] as well as RTN via convolutional neural networks.

In this paper, we propose a new approach based on meta reinforcement learning (meta-RL) that integrates guidance and navigation functions providing a complete solution to the lunar landing problem that integrates an image-based navigation to an intelligent guidance. More specifically, we design a simulation environment that is able to integrate the dynamics of the system and simulate image acquisition from on-board cameras. This is achieved by interfacing the simulator in Python with a ray tracer (i.e. Blender) that generates accurate images using lunar digital terrain models (DTM) and a physically based rendering engine. The images are then used to update a policy in real time using reinforcement learning. We take advantage of the latest discoveries in Convolutional Neural Net (CNN) and Recurrent Neural Nets (RNN) for image processing and Proximal Policy Optimization (PPO) to design our agent and learn the optimal policy for soft landing.

II. Problem formulation

A. Environment and sensor specification

The algorithm is being developed for a lunar soft landing scenario. We consider the 3 degrees of freedom (3DOF) problem. For this reason the lander is consider a point mass with throttable thrusters capable of creating a variable thrust in all three directions. The problem is described in a ortho-normal reference system centered on the nominal landing target. The equations of motion governing the dynamics of the problem expressed in the above mentioned reference frame are:

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

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

where $\mathbf{r} = [r_x \quad r_y \quad r_z]$ is the state in the target centered ortho-normal frame with the z-axis pointing up, $\mathbf{g}$ is the gravity vector and $\mathbf{T}$ is the thrust vector:

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

In this environment, the classical trajectory optimization problem takes the form:

$$\max_{t_f, \mathbf{T}} m_L(t_f) = \min_{t_f, \mathbf{T}} \int_0^{t_f} \|\mathbf{T}\| \mathrm{dt} \tag{4}$$

Subject to the dynamics constraints in 1, with the following boundary conditions:

$$\begin{aligned} \mathbf{r}(0) &= \mathbf{r}_0 \\ \mathbf{v}(0) &= \dot{\mathbf{x}}(0) = \mathbf{v}_0 \\ \mathbf{r}(t_f) &= \mathbf{r}_f \\ \mathbf{v}(t_f) &= \dot{\mathbf{x}}(t_f) = \mathbf{v}_f \end{aligned} \tag{5}$$

Where $\mathbf{r}_0$, $\mathbf{v}_0$, $\mathbf{r}_{t_f}$ and $\mathbf{v}_{t_f}$ and initial and final position and velocity vectors respectively. The lander is also subject to a path constraint in the form of a glide slope:

$$\theta_g = \arctan\left(\frac{\sqrt{r_y^2 + r_z^2}}{r_x}\right) < \hat{\theta}_g \tag{6}$$

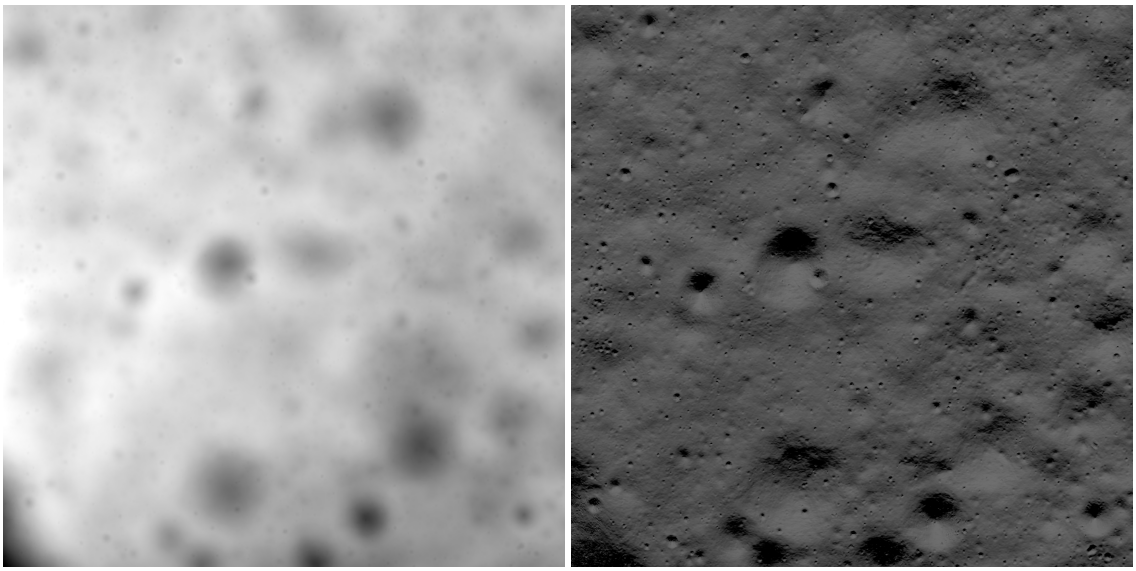
and a thrust constraint:

$$0 < T_{min} < \|\mathbf{T}\| < T_{max} \tag{7}$$

Moreover, since the powered descent on a planetary body like the Moon is initiated at an altitude that is small with respect to the radius of the body, the gravity $\mathbf{g}$ is considered constant during the entire mission.

B. Moon surface simulator

The goal is to train an agent to land on a planetary body autonomously using a radar altimeter and images of the ground captured by an optical sensor. The radar altimeter is assumed to be able to continuously feed the policy with altitude and vertical velocity measurements. The optical sensor is mounted on the ground-facing side of the spacecraft and is assumed to have a bore sight direction perpendicular to the ground at all times. The images are generated using a renderer based on raytracing technology. In particular we use Cycles within the open source Blender package. Specifically, Cycles is a physically-based path tracer for production rendering that computes light traces from all the light sources in the scene and their interactions with the objects. It gives a lot of freedom on how to set up the environment and since it runs in python its integration with the programming language is straightforward. In this case we used a Digital Terrain Model (DTM) taken from the Lunar Reconnaissance Orbiter (LROC) database [23]. Figure 1 shows a reduced contrast version of the DTM and a rendered view of the ground using the raytracer.



(a) DTM (b) Rendered image

Fig. 1 Apollo 16 landing site [23]

The DTM is a portion of the Apollo 16 landing site. It has a resolution of 1791x1791 pixels and is in 32 bit gray-scale format. The high bit depth of the image ensures that the ground elevation is computed with high accuracy. It was used in blender as a texture that cycles interpret as a displacement map. Specifically, the value of each pixel is used as a parameter that indicates the elevation of that pixel in the terrain model. The renderer than uses that information to calculate the light bounces of the light emitted by a sun-like source with the ground without actually using a real 3D shape. This is a technology normally used in games where a high frame rate is crucial for smooth transitions between images. This is achieved thanks to the reduction of the polygon count that this technique enables. In this case it allows for a very fast rendering of the observation image without loss of accuracy. Specifically, the rendering of an observation of size 16x16 pixels takes an average of 0.015 seconds using 5 light bounces and 20 samples. Blender also has a big advantage over other rendering engines: it is written in python. This means it allows for python scripts to run within the program itself. This allowed us to run the whole learning algorithm inside the renderer framework itself which also allowed us to pass the image observations to the learning algorithm directly as pixel data arrays eliminating the need of saving them on the hard drive. This speeds up the policy rollout phase of the algorithm quite a bit and allowed for effective and efficient training. The camera's focal length is fixed and set to 50 mm with a squared sensor of 16x16 pixel. We are considering increasing the sensor dimension but as of now this configuration has shown good performance. Increasing the pixel count could potentially increase performance as the policy could distinguish more features within the images but, in the meantime, the learnable parameters would increase dramatically which would ultimately lead to an increased learning time.

III. Guidance law

In this section we describe the Reiforcement Learning framework in which we cast the landing problem formalized in section II to generate a closed-loop guidance policy. The algorithm is capable of driving the spacecraft to a specified target state with quasi-pin-point accuracy, satisfying the problem constraints (e.g. glide slope and thrust).

A. Meta-Reinforcement Learning

In Reinforcement Learning (RL), an agent learns through repeated interaction with an environment how to complete a task. In general this environment is described as a Markov Decision Process (MDP) in which the state at a particular time depends on the previous state only. The MDP can be considered an abstraction of an environment with a continuous state space S, a continuous action space A, a state transition distribution $\mathrm{P}(x_{t+1}|x_t, u_t)$ that describes the probability of transitioning to the next state given a certain action and a reward function $r = \mathrm{R}(x_t, u_t)$ where $x \in \mathrm{S}$, $u \in \mathrm{A}$ and r is a scalar reward signal. In cases when the state cannot be observed directly or the information is affected by noise it is useful to introduce the definition of a partially observable MDP (POMDP). In this case the state x becomes an hidden state and what is actually available is its observation o through an observation function $\mathrm{O}(x)$. This is the case for example in image or LIDAR based navigation where the observation embeds the information about the state without it being explicitly available to the agent. The agent operates in the environment defined by the POMDP using a parametrized policy π_θ, generating an action u_t based off the observation o_t, receiving a reward r_{t+1} and observation o_{t+1}. It should be noted that in this case it is not true that each retains all the possible information about the system. The complete history of the sequence of states might be needed to predict the following states. The reinforcement algorithm task is to optimize the policy π_θ in order to maximize the sum of the rewards collected along a rollout episode. All the constraints such as minimum and maximum thrust, glide slope, terrain feature avoidance as well as the task goals can be included in the reward function and will be accounted for during training.

There are a lot of different kinds of reinforcement learning algorithms that have shown to perform well in a variety of tasks. What they all have in common is the fact that they need to perform a very large number of trials to be able to learn even the most trivial task. Task that we human learn after only an handful of repetitions. This is the intuition and motivation on which Meta-Reinforcement Leanrning (Meta-RL) is based on. The main reason why human learn so much faster than machine learning algorithms is that we use previous knowledge to learn new tasks. We essentially learn how to learn through experience. For instance, we use the empirical knowledge of physics of the world to learn very quickly how to interact with objects in the 3D space (e.g. stacking and moving objects). A machine instead, every time it has to learn a new task it has to do so from scratch. So in the objects staking example, it not only has to learn to stack objects on top of each other, it also has to learn how gravity effects the motion of the objects and the contact interfaces between them. So meta-RL formalizes this idea of "learning to learn" for machine learning algorithms.

The implementation of such concept is not straightforward. In general RL algorthms work by mapping an observation state to an action using some sort of neural network describing a policy. Learning is then achieved using roll-outs of

such policy as it interacts with a specific environment: the optimal policy is then the one that maximizes the cumulative reward:

$$\theta^* = \text{argmax}_\theta E_{\pi_\theta(\tau)}[R(\tau)] = f_{RL}(\mathbb{M}) \tag{8}$$

where θ^* are the optimal parameters of the parametrized policy π_θ, $R(\tau)$ is the cumulative reward and $\mathbb{M}$ is the MDP or POMDP.

In the case of meta-RL, the rollouts come from a distribution of tasks instead of a static single environment. This in theory allows the agent to learn how to adapt to a changing environment. Using the same formalization as for classical RL:

$$\theta^* = \text{argmax}_\theta \sum_{i=1}^{n} E_{\pi_{\phi_i}(\tau)}[R(\tau)] \tag{9}$$

where $\phi_i = f_\theta(\mathbb{M}_i)$ is a function of the i-th MDP corresponding to the i-th task. Meta-RL is then normally implemented using the following high level structure:

1. sample task i, collect data $\mathbb{D}_i$ (10)
2. adapt policy by computing $\phi_i = f(\theta, \mathbb{D}_i)$ (11)
3. collect data $\mathbb{D'}_i$ with adapted policy π_{ϕ_i} (12)
4. update θ according to $\mathbb{L}(\mathbb{D'}_i, \phi_i)$ (13)

Where the adaptation can be done in multiple rounds and the update of the parameters θ of the policy is done over multiple tasks. In practise there are two approaches to solve this problem. The first approach is to synthesize the policy as a recurrent neural network. These are particular kinds of networks where connections between nodes form a directed graph along a temporal sequence. This allows them to exhibit temporal dynamics behaviour. Unlike normal feed-forward nodes, recurrent nodes have internal states (which are normally referred to as hidden) that retain information about the temporal variation of the input data. This peculiar behaviour is what makes them suitable for meta-learning. What it's done in practice then is not as simple as plugging in a recurrent net as policy. The policy is indeed a recurrent network, what is different is that the hidden states of the recurrent net are kept constant through a specific task while the weights are learning across tasks using normal RL algorithms. This allows the net to retain in its hidden states the knowledge about the tasks it can face. This way it can eventually recognize a task by the temporal evolution of the input states and act accordingly. The high level algorithm can be seen in Algorithm 1. The second approach is fundamentally

Algorithm 1 Meta-RL - Recurrent network

```
procedure WHILE TRAINING
    for i in tasks do
        initialize hidden state h_0 = 0
        for t in timesteps do
            sample 1 transition D_i = D_i ∪ {(s_t, a_t, s_{t+1}, r_t)} from π_{h_t}
            update policy hidden state h_{t+1} = f_θ(h_t, s_t, a_t, s_{t+1}, r_t)
    update policy parameters θ ← θ − ∇_θ Σ_i L_i(D_i, π_h)
```

different as it utilize policy gradient to optimize a policy by moving in the direction that maximizes a particular objective function dependent on the specific policy used. It is different from the recurrent network method also in the sense that instead of having the knowledge of the different tasks embedded in the hidden states, it works by training the model on a distribution of task obtaining a general model and fine-tuning its parameters in the test-phase using a small number of rollouts. This is the basic idea behind the Model Agnostic Meta Learning [24] method which produces an agent that is not particularly good at any task at the end of the training phase, but specializes quickly (only a few updates, even one) to a new task at test phase. A pseudo-code implementation of this algorithm can be seen in Algorithm 2. In our case we used the first method as the temporal variation of states in a guidance problem clearly capture the nature of the task to accomplish. In particular we synthesize a policy π_θ using a recurrent network that the agent uses to interact with the POMDP that we introduced before. The action is used as a forcing term in the equations of motion which are integrated along a time-step and the observation along with the reward signal are recorded and fed back to the agent. In our case the observation is a combination of a grayscale image generated using a raytracer as explained in Section II.B and the vertical position and velocity coming from a radar altimeter. The agent uses this observation to produce another

Algorithm 2 Meta-RL - Policy Gradient Optimization

procedure WHILE TRAINING
  for i in tasks **do**
    sample k episodes $\mathbb{D}_i = \{(s, a, s', r)\}_{1:k}$ from π_θ
    compute the adapted parameters $\phi_i = \theta - \alpha \nabla_\theta \mathbb{L}_i(\pi_\theta, \mathbb{D}_i)$
    sample k episodes $\mathbb{D'}_i = \{(s, a, s', r)\}_{1:k}$ from π_ϕ
  update policy parameters $\theta \leftarrow \theta - \nabla_\theta \sum_i \mathbb{L}_i(\mathbb{D}_i, \pi_{\phi_i})$

action and the cycle repeats itself. The environment can terminate an episode by passing a done signal to agent. The termination signal could be either due to the agent completing the task or violating a constraint. Initially the actions taken by the agent are random, which allows the agent to explore the state and actions spaces and gather information about the environment and which action is to be preferred given a particular observation. The information about the goodness of an action is embedded in the reward signal which is similar to the cost-to-go in an optimal control problem. As the learning progresses, the exploration is reduced in favor of exploitation of the knowkedge of the environment. For most application (landing guidance is one of them), the policy is deployed in the field as a deterministic law in which exploration is switched off as this could lead to reduced performances.

Having introduced how the agent interacts with the environment, the learning algorithm follows directly. Considering the observation x_k provided by the environment to the agent at time-step k. Note that x_k does not need to satisfy in general the Markov property. It can in fact come from a POMDP as described above. As long as the agent has a way of understanding the time evolution of the observations, it will be able to learn. Each episode results in a sequence of observations which we will call trajectory. A step in each trajectory at time t_k can be represented as (x_k, u_k, r_k) where x_k and r_k are the observation and the reward returned by the environment and u_k is the action taken. The reward can be a function of both the observation and the action. The reward is then typically discounted the further it is in the future. This is done to put a finite horizon to the task and facilitate temporal credit assignment. Then the sum of discounted rewards for a trajectory can be defined as the return:

$$r(\tau) = \sum_{i=0}^{T} \gamma^i r_k(x_k, u_k) \tag{14}$$

where $\tau = [x_0, u_0, \ldots, x_T, u_T]$ denotes the trajectory and $\gamma \in [0, 1)$ is the discount factor. The objective function the RL methods seek to optimize is given by:

$$J(\theta) = \mathbb{E}_{p(\tau)}[r(\tau)] = \int_{\mathbb{T}} r(\tau) p_\theta(\tau) d\tau \tag{15}$$

where:

$$p_\theta(\tau) = \left[\prod_{k=0}^{T} p(x_{k+1}|x_k, u_k) \right] p(x_0) \tag{16}$$

where $\mathbb{E}_{p(\tau)}$ denotes the expectation of the reward over the trajectories. Now if we consider the action u_k as a stochastic function of θ, or $u_k \sim \pi_\theta(u_k|x_k)$, then the policy gradient expression becomes:

$$\nabla_\theta J(\theta) = \int_{\mathbb{T}} \sum_{k=0}^{T} r_k(x_k, u_k) \nabla_\theta log \pi_\theta(u_k|x_k) p_\theta(\tau) d\tau \approx \sum_{i=0}^{M} \sum_{k=0}^{T} r_k(x_k^i, u_k^i) \nabla_\theta log \pi_\theta(u_k^i|x_k^i) \tag{17}$$

Where the integral over the trajectory τ is approximated using the monte-carlo rolluts samples τ^i $p_\theta(\tau)$, given the environment's transition probabilities, $p(x_{k+1}|x_k)$, which in this case are the deterministic equations of motion. The expression in 17 is the policy gradient equation and is the basic concept on which REINFORCE algorithm is based. In general this is used as a baseline as this was later improved through the years. In particular it was shown that instead of using the actual reward $r_k(x_k, u_k)$, one can use the action-value function $Q^\pi(x_k, u_k)$ called normally Q-function, which is known as the Policy Gradient Theorem. Moreover, to reduce the variance of the policy gradient, a state dependent basis can be subtracted from $Q^\pi(x_k, u_k)$. This basis is normally called value function $V^\pi(x_k)$ and the quantity that is then used to approximate the policy gradient is the advantange function $A^\pi(x_k, u_k) = Q^\pi(x_k, u_k) - V^\pi(x_k)$. Moreover, this

function can be approximated using the rollouts by training a second neural net which is normally referred to as critic. This method is known as Advantage-Actor-Critic (A2C) Method and the policy gradient using this method becomes:

$$\nabla_\theta J(\theta) = \sum_{i=0}^{M} \sum_{k=0}^{T} \nabla_\theta log \pi_\theta(u_k^i | x_k^i) A^\pi(x_k^i, u_k^i) \tag{18}$$

Once the gradient is calculated, it is used to update the policy by simply moving in its direction:

$$\theta^+ = \theta^- + \beta_\theta \nabla_\theta J(\theta)|_{\theta=\theta^-} \tag{19}$$

where $\beta_\theta > 0$ is the learning rate.

B. Proximal Policy Optimization

What we used to optimize the policy in this case is a derivation of the A2C method. The Proximal Policy Optimization (PPO) approach [25] belongs to the family of policy gradient algorithms and has demonstrated state-of-the-art performances on many benchmark RL problems. It is developed as a derivation of the Thrust Region Policy Optimization (TRPO) Method [26]. This method formulates the policy optimization problem in a way such that the size of the gradient step taken during each iteration is restricted using some sort of dynamically calculated constraint. The TRPO policy update problem is formulated as:

$$\begin{aligned} \min_\theta \quad & \mathbb{E}_{p(\tau)} \left[\frac{\pi_\theta(u_k|x_k)}{\pi_{\theta_{old}}(u_k|x_k)} A_w^\pi(x_k, u_k) \right] \\ s.t. \quad & \mathbb{E}_{p(\tau)}[KL(\pi_\theta(u_k|x_k), \pi_{\theta_{old}}(u_k|x_k))] \leq \delta \end{aligned} \tag{20}$$

Where KL is the Kullback-Leibler divergence [27] between the present and old policy. The parameter δ is a tuning parameter that imposes a bound to the update. It is proven that of the update is bounded at each iteration by a parameter $C(KL)$ that depends on the KL divergence, the policy improves monotonically towards the optimal. This in general leads to prohibitively small update so Equation 20 with a constant constraint parameter is used in stead. Additionally, Equation 20 is approximately solved using the conjugate gradient algorithm, which approximates the constrained optimization problem given by Equation 20 with a linearized objective function and a quadratic approximation for the constraint. The PPO method approximates the TRPO optimization process by accounting for the policy adjustment constrain with a clipped objective function. The objective function used with PPO can be expressed in terms of the probability ratio $p_k(\theta)$ given by,

$$p_k(\theta) = \frac{\pi_\theta(u_k|x_k)}{\pi_{\theta_{old}}(u_k|x_k)} \tag{21}$$

Where the PPO objective function is then:

$$L(\theta) = \mathbb{E}_{p(\tau)} \left[\min[p_k(\theta), \text{clip}(p_k(\theta), 1-\epsilon, 1+\epsilon)] A_w^\pi(x_k, u_k) \right] \tag{22}$$

This clipped objective has been shown to maintain the KL divergence constrained which aids convergence by ensuring that the policy does not change drastically between updates.

PPO uses an approximation of the advantage function that is the difference between the empirical return (discounted reward) and a state value function baseline and gives information about how much better an action is with respect to the average action:

$$A_w^\pi(x_k, u_k) = \left[\sum_{l=k}^{T} \gamma^{l-k} r(u_l, x_l) \right] - V_w^\pi(x_k) \tag{23}$$

Where $\gamma \in [0, 1)$ is the discount factor and is closer to one the more the algorithm should care about rewards collected far into the future. The Value function $V_w^\pi(x_k)$ here is approximated by the critic and is learned using the cost function:

$$L(w) = \sum_{i=1}^{M} = V_w^\pi(x_k^i) - \left[\sum_{l=k}^{T} \gamma^{l-k} r(u_l, x_l) \right] \tag{24}$$

where M is the number of rollout trajectories. In practice, policy gradient algorithms update the policy using a batch of trajectories (roll-outs) collected by interaction with the environment. Each trajectory is associated with a single episode,

with a sample from a trajectory collected at step k consisting of observation x_k, action u_k, and reward $r_k(x_k, u_k)$. Finally, gradient ascent is performed on θ and gradient decent on w. The update equations are:

$$w^+ = w^- - \beta_w \nabla_w L(w)|_{w=w^-} \tag{25}$$

$$\theta^+ = \theta^- + \beta_\theta \nabla_\theta J(\theta)|_{\theta=\theta^-} \tag{26}$$

where β_w and β_θ are the learning rates for the value function, V_w^π, and policy, $\pi_\theta(u_k|x_k)$, respectively. In our case, we adjust the clipping parameter ϵ to target a KL divergence between policy updates of 0.008. This gives a good compromise between quick learning and targeting performances. The policy and value function are learned concurrently. The exploratory action distribution is a Gaussian distribution with mean $\pi_\theta(x_k)$ and a diagonal covariance matrix. Because the log probabilities are calculated using the exploration variance, the degree of exploration automatically adapts during learning such that the objective function is maximized.

C. Guidance law optimization method

The environment in this case contains the model of the lander, the equations of motion and all the physical constraints of the problem. At each time-step, it samples the action from the exploratory policy, integrate the equations of motion, calls the renderer to generate the observation and return the observation along with the state and the reward. It should be noted that when using a policy gradient method, there is no need to deploy the value function approximator (critic), only the policy is need to provide a control action. For this reason, we give the critic access to more information with respect to the policy network. In particular we pass the velocity error between the true velocity and the target velocity $\mathbf{v_{error}}$, the time-to-go t_{go} and the lander estimated altitude r_z. The policy instead has access to the optical images observations, the altitude r_z and the vertical component of the velocity $\dot{r}_z$. So the value function observations are:

$$obs_{VF} = [\mathbf{v_{err}} \quad t_{go} \quad r_z] \tag{27}$$

while the observations available to the policy π_θ are:

$$obs_{\pi_\theta} = [\mathbf{I} \quad r_z \quad \dot{r}_z] \tag{28}$$

where $\mathbf{I}$ is a 16x16 array representing the raw grayscale pixel data from the raytracer. The action space is continuous and corresponds to $\mathbb{R}^3$ as we are considering motion in a 3DOF environment. A schematic representation of the whole sytem can be seen in Figure 2.

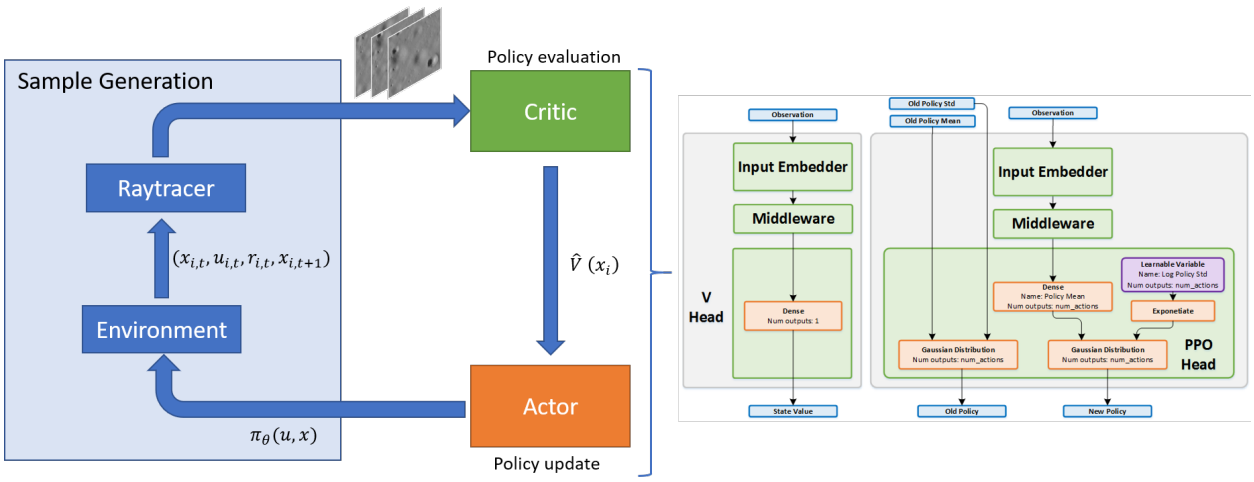


Fig. 2 Overall framework

The value function is implemented as a four layer fully connected neural network as the information about the motion are completely encoded within the full state observation and there is no need for a recurrent network. The layers specification and activation functions can be seen in Table 1: The policy instead has a mixed input, images and vector observations. In this case we process the two inputs separately. The images go through a series of two convolutional

Table 1 Value function network

Layer	Units	Activation function
FC1	60	tanh
FC2	17	tanh
FC3	5	tanh
FC4	1	linear

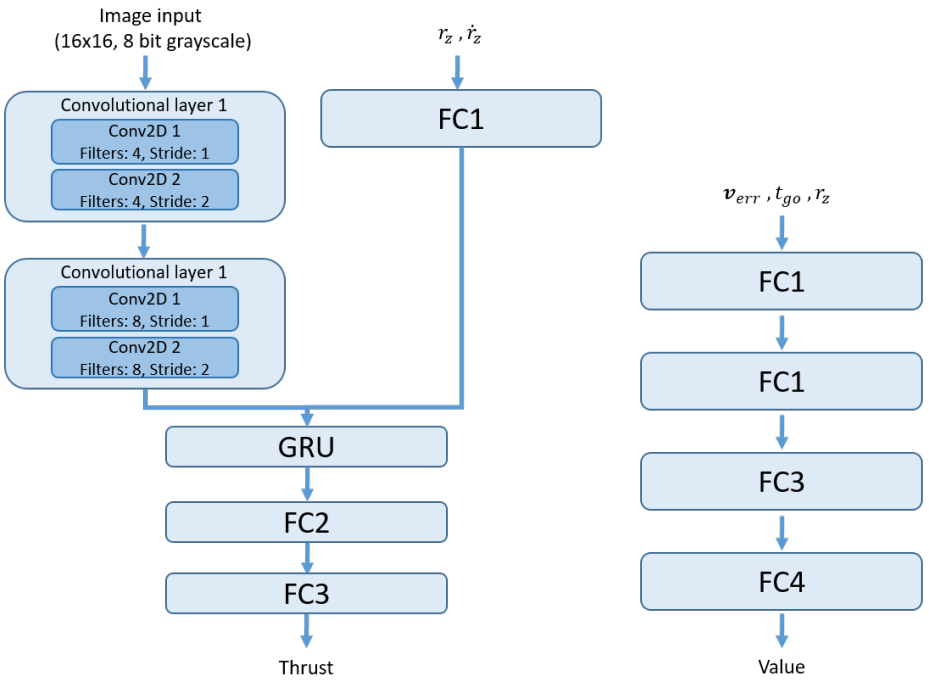
Table 2 Policy network

Layer	Units	stride	Activation function
CNN1	4	1	tanh
CNN2	4	2	tanh
CNN3	8	1	tanh
CNN4	8	2	tanh
GRU	73	-	-
FC1	30	-	tanh
FC2	30	-	tanh
FC3	3	-	linear

layer (CNN) that filter the image input and extract high level features. In particular each one of them is composed by two convolutional sublayers: the first is a normal convolutional layer with stride 1 that increases the number of channels by applying a combination of convolutional filters without changing the size of the output image, the second is another convolutional layer with stride 2 that substitute the pooling layer in classical convolutional nets. This has been shown to perform better in this kind of regression tasks as the image is still reduced in size thanks to the stride equal to 2 but the additional filters extract more information than a simple max-pooling. The vector observation data goes through a simple fully connected layer. The image input is then flatten and the two observations are concatenated. As explained in Section III.A, a recurrent layer is needed to capture the different tasks to meta-learn. Given these premises, the flattened observation is passed through a recurrent cell (GRU) that, es explained above, encodes the knowledge about the temporal evolution of the observations as a sequence is fed to the network. The network is then completed with 2 fully connected layers. Table 2 summarizes the architecture.

It is worth spending some more words on why using a recurrent network in this case allows to solve the meta-learning problem. As explained in Sectio III.A, the network has a recurrent cell whose internal states are kept constant for each single task while the weights are updated through gradient descent. This allows the network to distinguish the temporal evolution of each particular task and effectively adapt to the specific environment. During learning, this operatively works by unrolling the recurrent layer in time across different timesteps of a single sample trajectory. In our case we unroll the layer for 60 steps as suggested by [10, 13, 14]. Following the procedure in [13], the unrolling must be done in a manner consistent with processing a large number of episodes in parallel. This is somewhat cumbersome in the case of reinforcement learning as, for example, if we want to unroll the net for 60 steps for the forward pass through the network, the hidden state of the recurrent cell at step 61 is not available until the agent have completed the first 60 steps. To cope with this situation, we have to use the actual hidden state of the GRU cell as observation and add it to the rollouts. So the step of interaction with the environment not only returns the observation-action-reward tuple but also the hidden state of the policy. The forward pass through the policy then works by unrolling the batch of inputs prior to the recurrent layer, reshaping it according to the layer dimension and recurrent steps (in this case we consider 60 steps). More implementation details are given in the references [13].

The reward function is one of the most important things to set up in a reinforcement learning algorithm. The intuitive way to reward the agent in a soft landing task is to give a positive reward if the agent performs a successful landing (e.g. arrives close to the target with low speed) and penalize it if it violates a constraint (e.g. falls below ground or violates the glideslope constraint). These kind of rewards are considered sparse in the sense that the agent is rewarded or penalized only at specific time-steps during a rollout. It is know that sparse rewards are a tough problem to solve for reinforcement learning because given the randomization of exploratory actions, it is very unlikely that the agent will see



(a) Policy Network (b) Value Function Network

Fig. 3 Policy and value function network architecture

a successful landing more than an handful of times during a policy rollout batch. For this reason, we implemented the reward following a shaping function. This techniques allow to give a per-timestep reward to the agent and has been shown to work well in pinpoint landing tasks [11]. We summarize here the procedure. It works by giving the agent hints about a good trajectory to follow for a successful landing. Specifically we use the error with respect to a gaze heuristic potential function to drive the agent towards the target. It works by trying to keep the velocity aligned with the line of sight vector. This ensure pinpoint but not necessarily soft landing. To ensure that the final velocity is minimized, the agent agent estimates time-to-go as the ratio of the range and the magnitude of the lander's velocity and reduces the targeted velocity as time-to-go decreases. To achieve this according to [11], the reward function has the following form:

$$r = \alpha\|\mathbf{v} - \mathbf{v}_{targ}\| + \beta\|\mathbf{F}_B\| + \eta + \kappa(r_z < 0 \quad \text{and} \quad \|\mathbf{r}\| < r_{lim} \quad \text{and} \quad \|\mathbf{v}\| < v_{lim}) \tag{29}$$

Where:

$$\begin{aligned}
\mathbf{v}_{targ} &= -v_0 \left(\frac{\hat{\mathbf{r}}}{\|\hat{\mathbf{r}}\|}\right)\left[1 - \exp\left(-\frac{t_{go}}{\tau}\right)\right] \\
t_{go} &= \frac{\|\hat{\mathbf{r}}\|}{\|\hat{\mathbf{v}}\|} \\
\|\hat{\mathbf{r}}\| &= \begin{cases} \mathbf{r} - [0 \quad 0 \quad 5] & \text{if} \quad r_z > 5 \\ [0 \quad 0 \quad r_z] & \text{otherwise} \end{cases} \\
\|\hat{\mathbf{v}}\| &= \begin{cases} \mathbf{v} - [0 \quad 0 \quad -2] & \text{if} \quad r_z > 5 \\ \mathbf{v} - [0 \quad 0 \quad -1] & \text{otherwise} \end{cases} \\
\tau &= \begin{cases} \tau_1 & \text{if} \quad r_z > 5 \\ \tau_2 & \text{otherwise} \end{cases}
\end{aligned} \tag{30}$$

and:

- α is a term that penalizes the error with respect to the target velocity

Table 3 Hyperparamenters

v_0 (m/s)	τ_1 (s)	τ_2 (s)	α	β	η	κ
100	20	100	-0.01	-0.05	0.01	10

Table 4 Initial conditions

	Position		Velocity	
	min (m)	max (m)	min (m/s)	max (m/s)
Downrange	1500	2000	-100	-80
Crossrange	-250	250	-30	30
Elevation	3200	3500	-30	-20

- β is a term that penalizes the control effort
- η is a positive constant that encourages the agent to perform more steps avoiding collisions with the constraints
- κ is a bonus given for a successful landing (i.e. the final position and velocity are within a certain threshold)

This reward function allows the agent to trade off between tracking the target velocity, conserving fuel, and maximizing the reward bonus given for a successful landing.

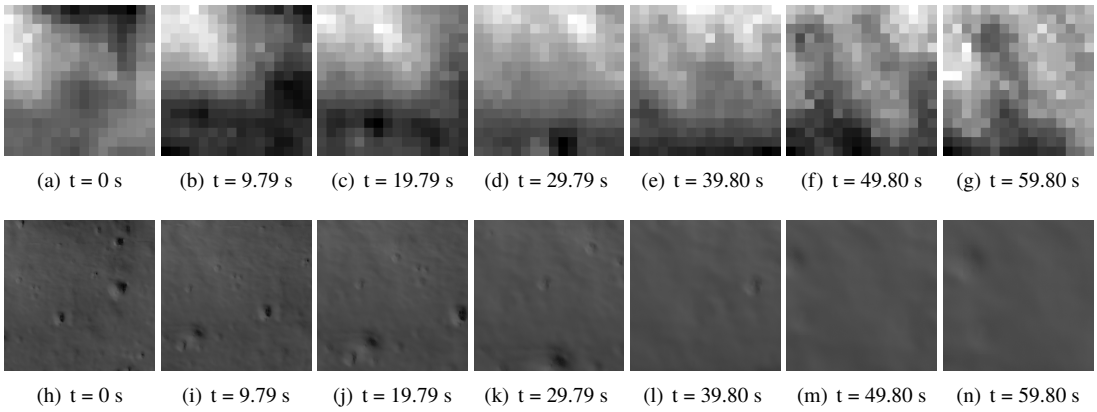
IV. Experiments results

We show here the performances of the algorithm on a pinpoint lunar landing task with engine failure and uncertain gravity field and initial spacecraft mass. The gravitational acceleration is selected randomly from a uniform distribution $+/-5\%$ of the nominal value and kept constant along a single episode. The lander initial mass is also set randomly over a uniform distribution $+/-5\%$ of the nominal value (1500 $[kg]$). The engine failure is simulated by reducing the thrust available in one of the three directions at random for the entire duration of the mission. The policy is optimized using PPO described in Section III.B. The observations that feed into the policy network are images that come directly as raw grey-scale pixel data from Blender plus vertical position and velocity. Note that they are the only information passed to the actor as explained in Section III.C. The critic instead has access to the velocity error with respect to the target velocity, the t_{go} and the altitude. The initial condition for each episode is sampled from the distribution described in Table 4. The target state is a position 1000 meter above the origin of the axis with a -1 m/s velocity in the vertical direction. We used this scenario instead of a trying to reach the ground because the DTM does not have a high enough resolution to be able to distinguish features at lower altitude. The images at lower altitudes would become almost identical to each other which would degrade the quality of the learning. We believe this is not a limitation of the algorithm, it is in fact a problem related the quality of the DTM. It should also be noted that the states that are used by the learning algorithm are instead shifted down by 1000 meters so that the target position is in fact the origin of the reference frame. The initial conditions seen by the learning algorithm are shown in 5. An example of the images passed to the policy during training can be seen in Figure 4 along with an higher resolution version of the same area created afterward..

The results relative to a trajectory using the policy in test mode (with exploration switched off) are shown in Figure 5 and Table 6. Table 6 shows the performance of the algorithm on a test trajectory using the learned policy. In particular it shows mean, standard deviation and maximum value of the norm of the terminal position and velocity errors. The

Table 5 Initial conditions for RL algorithm

	Position		Velocity	
	min (m)	max (m)	min (m/s)	max (m/s)
Downrange	1500	2000	-100	-80
Crossrange	-250	250	-30	30
Elevation	2200	2500	-30	-20



(a) t = 0 s (b) t = 9.79 s (c) t = 19.79 s (d) t = 29.79 s (e) t = 39.80 s (f) t = 49.80 s (g) t = 59.80 s

(h) t = 0 s (i) t = 9.79 s (j) t = 19.79 s (k) t = 29.79 s (l) t = 39.80 s (m) t = 49.80 s (n) t = 59.80 s

Fig. 4 Example image sequence. Top: normalized grayscale images used for training. Bottom: high resolution images

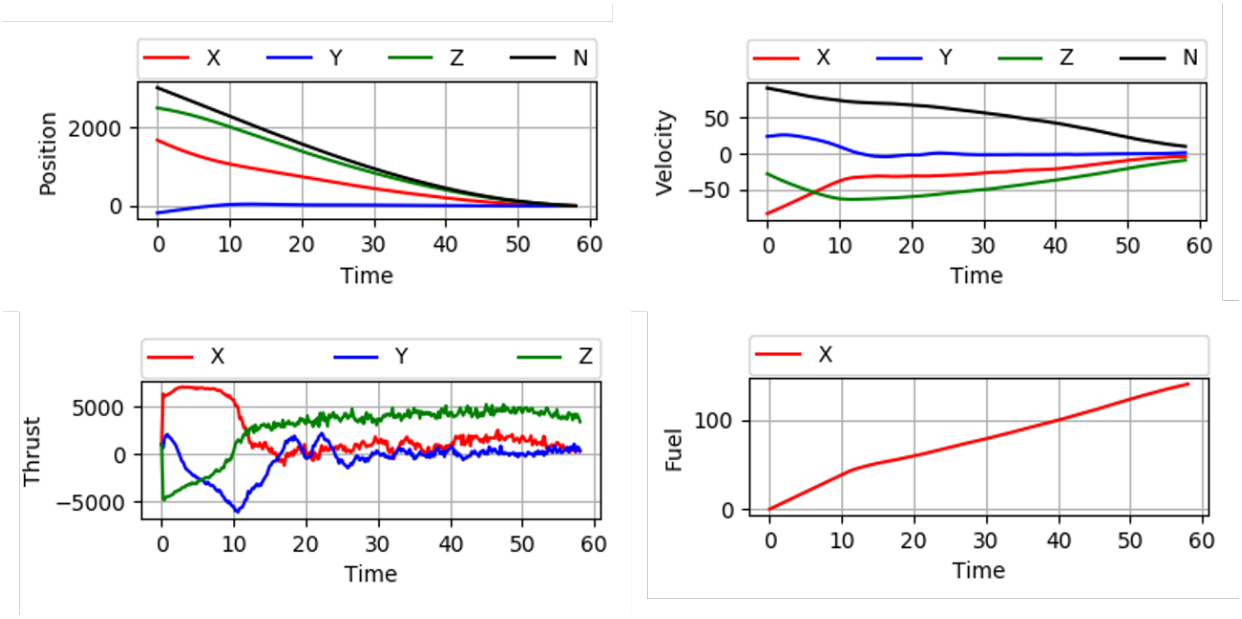
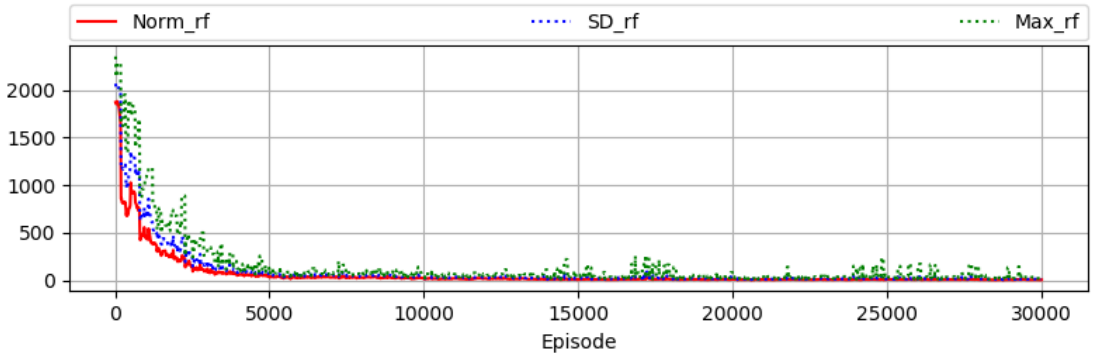


Fig. 5 Validation trajectory

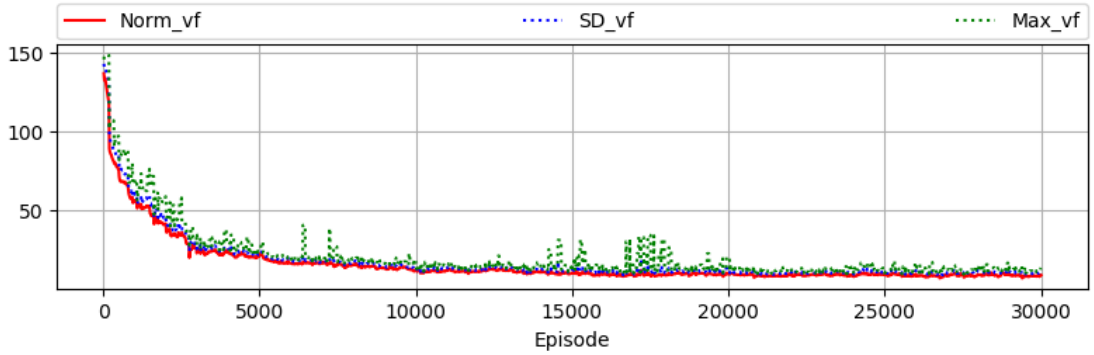
lander achieves fairly good performance given the size of the problem, managing to bring the lander close to the target with a contained velocity error even though at this stage the performance are still not suitable for true pinpoint landing. The learning curves are shown in Figures 6 which plot statistics for terminal position (r_f) and terminal velocity (v_f) as a function of episode, with the statistics calculated over the 10 episodes used to generate rollouts for updating the policy and value function. We also see from Figure 7, that the learning algorithm using the reward shaping function and PPO learns smoothly without ever diverging across consecutive iterations. It should be noted that the results shown here comes from a work in progress. There is still some experiments to be conducted, especially relatively to the networks architectures and the hyper-parameters.

V. Conclusion

In this paper we demonstrate how image-based navigation and real time guidance can be implemented using a single system based on convolutional-recurrent neural net. Using a meta-RL framework was in fact possible to design a closed look guidance algorithm that takes observations in the form of raw pixel data values taken by an optical down-facing



(a) Position error



(b) Velocity error

Fig. 6 Learning curves for final position and velocity error

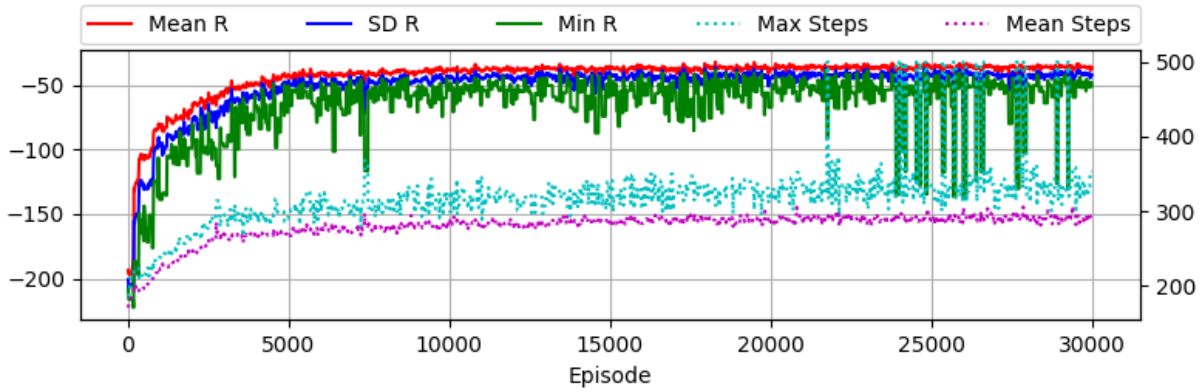


Fig. 7 Reward profile for 30000 episodes (45.7 hours)

camera and readouts from a radar altimeter as input and outputs thrust values that ensure targeting capabilities with quasi-pinpoint accuracy in a lunar landing task. Moreover, meta-learning allows the agent to adapt to an environment with uncertainty in the gravitational field and the initial mass of the lander. The agent also manages to adapt to a reduce thrust scenario that simulates an actuator failure. In this simulated scenario, the agent manages to reach the target with quasi-pinpoint accuracy across multiple trials. There is still an open question on how this algorithm would perform with a higher resolution DTM to perform real pinpoint landing. It is important to note in this regard, that the performance in terms of accuracy are remarkable. It is safe to assume that, provided a good enough DTM, it should perform at a comparable level in the full landing scenario. It should also be noted that this work demonstrates the high

Table 6 Performance statistics

	Terminal position norm (m)			Terminal velocity norm (m/s)			Fuel (kg)
	μ	σ	max	μ	σ	max	μ
GRU-RNN	4.5	1.2	6.4	9.19	0.7	10.1	141.1

level capabilities of an agent trained using meta-RL in a lunar landing task but the full potential is still to be unlocked. These results are to be taken as preliminary as we investigate more on how to obtain better performance. The results in this paper show that indeed the agent learns and could probably learn a more accurate solution just by training for more episodes. Moreover, there are endless possibilities as on how to build the actor and critic networks and some more experiments should be made in that regard, as well as trying different hyper-parameters such as KL divergence target, reward weights and number of policy rollouts per iteration.

This all in all demonstrates that RL has a lot of potential in the field of guidance and navigation in aerospace applications where data is limited and the ability to learn from simulated environment is pivotal. The ability to adapt to uncertain environments is key in this sense. Moreover, neural networks are inherently closed loop as the feed-forward pass through a net is extremely fast so they are suitable for real time implementation. There is still a lot to be done in this field. We are just starting to understand what RL and meta-RL can do for autonomous spacecraft guidance and control in complex environments. Being able to use images alongside other sensors data in real time as input is a great feature that is worth exploring on the path towards intelligent autonomous systems, as well as perhaps implement more knowledge within the network about hazardous area of the environment and its relevant features in general.

References

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

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

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

[4] Wibben, D. R., and Furfaro, R., "Terminal guidance for lunar landing and retargeting using a hybrid control strategy," *Journal of Guidance, Control, and Dynamics*, ????, pp. 1168–1172.

[5] Liu, X., Lu, P., and Pan, B., "Survey of convex optimization for aerospace applications," *Astrodynamics*, Vol. 1, No. 1, 2017, pp. 23–40.

[6] Prabhakar, N., Painter, A., Prazenica, R., and Balas, M., "Trajectory-Driven Adaptive Control of Autonomous Unmanned Aerial Vehicles with Disturbance Accommodation," *Journal of Guidance, Control, and Dynamics*, Vol. 41, No. 9, 2018, pp. 1976–1989.

[7] Huang, Y., Li, S., and Sun, J., "Mars entry fault-tolerant control via neural network and structure adaptive model inversion," *Advances in Space Research*, Vol. 63, No. 1, 2019, pp. 557–571.

[8] Nagabandi, A., Clavera, I., Liu, S., Fearing, R. S., Abbeel, P., Levine, S., and Finn, C., "Learning to adapt in dynamic, real-world environments through meta-reinforcement learning," *arXiv preprint arXiv:1803.11347*, 2018.

[9] Schweighofer, N., and Doya, K., "Meta-learning in reinforcement learning," *Neural Networks*, Vol. 16, No. 1, 2003, pp. 5–9.

[10] Gaudet, B., Linares, R., and Furfaro, R., "Six Degree-of-Freedom Hovering using LIDAR Altimetry via Reinforcement Meta-Learning," *arXiv preprint arXiv:1911.08553*, 2019.

[11] Gaudet, B., Linares, R., and Furfaro, R., "Deep reinforcement learning for six degree-of-freedom planetary powered descent and landing," *arXiv preprint arXiv:1810.08719*, 2018.

[12] Gaudet, B., Linares, R., and Furfaro, R., "Seeker based Adaptive Guidance via Reinforcement Meta-Learning Applied to Asteroid Close Proximity Operations," *arXiv preprint arXiv:1907.06098*, 2019.

[13] Gaudet, B., and Linares, R., "Adaptive Guidance with Reinforcement Meta-Learning," *arXiv preprint arXiv:1901.04473*, 2019.

[14] Gaudet, B., Linares, R., and Furfaro, R., "Adaptive Guidance and Integrated Navigation with Reinforcement Meta-Learning," *arXiv preprint arXiv:1904.09865*, 2019.

[15] Gaskell, R. W., Barnouin-Jha, O. S., Scheeres, D. J., Konopliv, A. S., Mukai, T., Abe, S., and Kawaguchi, J., "Characterizing and navigating small bodies with imaging data," *Meteoritics & Planetary Science*, Vol. 43, No. 6, 2008, pp. 1049–1061.

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

[17] Krizhevsky, A., Sutskever, I., and Hinton, G. E., "Classification with deep convolutional neural networks," *In NIPS*, 2012, pp. 1106—1114.

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

[19] Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A. R., Jaitly, N., and Kingsbury, B., "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, Vol. 29, No. 6, 2012, pp. 82–97.

[20] Furfaro, R., Simo, J., Gaudet, B., and Wibben, D., "Neural-based trajectory shaping approach for terminal planetary pinpoint guidance," *In AAS/AIAA Astrodynamics Specialist Conference 2013*, 2013.

[21] Sánchez-Sánchez, C., and Izzo, D., "Real-time optimal control via Deep Neural Networks: study on landing problems," *Journal of Guidance, Control, and Dynamics*, Vol. 41, No. 5, 2018, pp. 1122–1135.

[22] Gaudet, B., and Furfaro, R., "Adaptive pinpoint and fuel efficient mars landing using reinforcement learning," *IEEE/CAA Journal of Automatica Sinica*, Vol. 1, No. 4, 2014, pp. 397–411.

[23] "LROC RDR products," `http://http://wms.lroc.asu.edu/lroc/rdr_product_select`, ????

[24] Finn, C., Abbeel, P., and Levine, S., "Model-agnostic meta-learning for fast adaptation of deep networks," *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, JMLR. org, 2017, pp. 1126–1135.

[25] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O., "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[26] Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P., "Trust region policy optimization," *International conference on machine learning*, 2015, pp. 1889–1897.

[27] Kullback, S., and Leibler, R. A., "On information and sufficiency," *The annals of mathematical statistics*, Vol. 22, No. 1, 1951, pp. 79–86.