DOI: 10.13009/EUCASS2022-6308
9TH EUROPEAN CONFERENCE FOR AERONAUTICS AND SPACE SCIENCES (EUCASS)

A point of view on Machine learning for GNC for Space Applications: Towards non-linear control

Martine Ganet-Schoeller, Quentin Delaméa,, Lori Lemazurier and Simon Calonne
ArianeGroup
51/61 Route de Verneuil
78130 Les Mureaux
Contact : martine.ganet@ariane.group

Abstract
Autonomous Guidance Navigation and Control (GNC) for space vehicles covers a complete new GNC strategy including on-board health monitoring ability, decision-making algorithms and long horizon strategies for reconfiguration. The key features would be a new concept with less robust design and more adaptive and/or learning parts. This concept would be drastically less expensive during the design phase and safer in case of failure. Within the framework of its research activities, ArianeGroup has been investigating intelligent methods for GNC for many years. Recent advances in Artificial Intelligence and machine learning applications has widened the spectrum of GNC opportunities. In the frame of this paper, we focus on developing enhanced controllers that outperform the ones synthetized with classic feedback control techniques. A technological survey oriented our researches towards neural networks structures trained by reinforcement learning techniques for Non Linear Control. The application of these techniques to a simple, yet representative, industrial study case of launcher upper stage control gives insight on the methodology, and, opens interesting perspectives in coupling Artificial Intelligence and Automatic control.

1. Introduction

Increasing autonomy or increasing adaptation capability is a strategic challenge for future space vehicles. It should enable reducing mission preparation burden in case of complex mission, reducing ground support, reducing robustness constraints, increasing performance and increasing safety. Therefore, it is an enabler for new launcher market covering launch on demand, reusable launchers, smart upper stage or servicing.
Autonomous Guidance Navigation and Control (GNC) covers a complete new GNC strategy with, in a first step of development, an increase of the level of adaptability each Guidance Navigation and Control functions. Then, we should extend these evolutions to integrated GNC, taking into account interactions and enhanced mission analysis tools. Finally, we shall include on-board health monitoring ability, decision making algorithms and long horizon strategies for reconfiguration (such as for instance, replaning in case of one of main engines failures). The key features would be a new concept with less robust design (nominal or reduced domain) and more adaptive and/or learning parts. This concept would be drastically less expensive during the design phase and safer in case of failure.

Today, the combination of artificial intelligence and learning theory with H/W, sensors and engines evolutions, opens new perspectives for autonomous GNC. Indeed, since a few years, a new step has been climbed in Artificial Intelligence (AI) and its applications (games, cars, robotics domains …) and we have to take advantage of all this new reality, to expand it towards launchers GNC.
AI methods and tools, were already proposed during the last decades for launch vehicles guidance and control applications through the use of Neural Networks (NN). For instance, [1] made a comparison between PID, NN, and, Variable Memory NN (VRNN) for the attitude control of a launch vehicle. And [2] trained a feedforward backpropagation NN to determine the acceleration to null both position and velocity offsets for soft landing guidance. Promising results were obtained for both Guidance and Control applications, however they were constrained by the efficiency of training algorithms and they could benefit from enhanced training methods.
Reinforcement Learning (RL) is a training method that is particularly well suited to problems that include a long-term versus short-term reward trade-off, such as GNC functions. Although RL has existed from decades, the field has grown

DOI: 10.13009/EUCASS2022-6308

up recently with the raise of deep learning algorithms, and, according to recent publications, RL appears to be the best training method for guidance and path planning applications. [3] uses a deep neural network for spacecraft landing guidance, and, [4] an adaptive guidance system using reinforcement metalearning with recurrent policy and value function for Mars landing.

In this paper, we have developed a study case to gain insight in the methodology of machine learning and RL for nonlinear control. Our industrial study case is representative of one major control challenge that is the sloshing phenomena of the ergols in the tanks of a launcher upper stage (see Figure 1). Sloshing perturbation shall be actively controlled to avoid mission constraints such as maneuvers restrictions, and, to avoid adjunction of heavy anti-sloshing devices that reduce the launcher performance. We use a pendulum model for sloshing representation. In this application, we shall manage the motion of the ergols in the tanks of the nonlinear system (e.g. with saturations and thrust and draining ratio dependent) with a lot of uncertain parameters and perturbations.. It will highlight some advantages, drawbacks and lessons learned.
The paper is organized as follows. After an introduction on data science, artificial intelligence and machine learning for GNC applications, we will present the nonlinear study case, and then, we will describe the selected intelligent control method and its resulting performances. In conclusion, we will present the opened perspectives that should be deepened in the frame of a PhD.

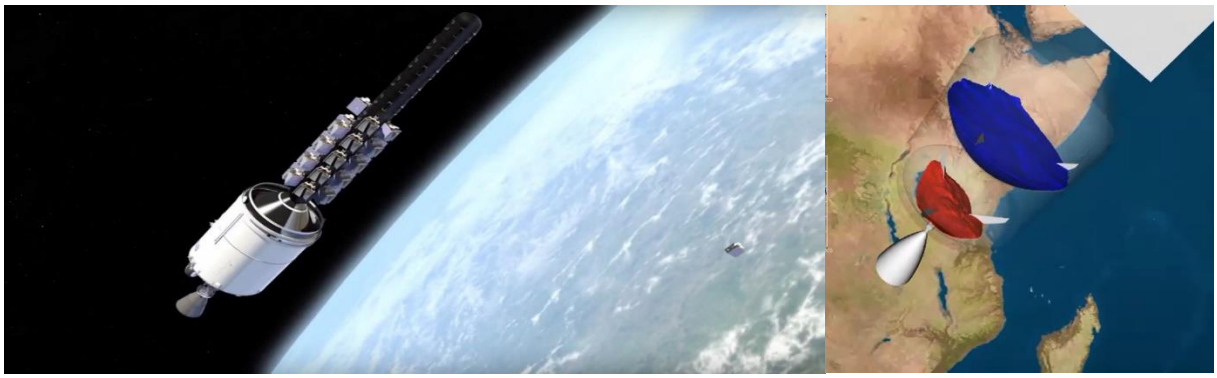


Figure 1: Illustrations of Ariane 6 upper stage for constellation P/L and ergol motion

2. Overview of data science, artificial intelligence and machine learning

Within the framework of its research activities, ArianeGroup has been investigating intelligent control methods for many years. Today we are taking advantage of recent advances in AI optimization techniques to find new GNC strategies.

2.1 Data science

Data has value, and its volume is increasing exponentially. Our processing capabilities also increase rapidly. Therefore, we need to find the rights methods to leverage correctly the value of our data. Luckily, what is now called **data science** is subject to many innovations, many of which are accessible through open source libraries. If the keyword data science is used a lot, it is hard to define its scope, as it can include everything related to data, from storage to sharing and to specific algorithms generating new information from data. This includes several fields:
Data storage and data consolidation, which allows humans to access the full amount of data needed to make a decision. In the frame of the transition to big data technologies, they also study new architectures to deal with more and more data.
Data visualization, which is a valuable assist for human intelligence to understand the content of a dataset, by showing key features distributions and providing a synthesis. They allow discovering patterns and correlations. It could be useful for GNC simulation analysis, post flight analysis through correlation analysis, use of surrogate modelling and complex phenomena understanding.
Machine learning, in which we make algorithms learn (hence the name artificial intelligence) the structure of data so that they can generate value. This will be the main focus of next subsection. As these techniques rely on statistics, they get more effective and can be more complex with big volumes of data, but they can still be used on datasets with only tens or hundreds of objects.

DOI: 10.13009/EUCASS2022-6308

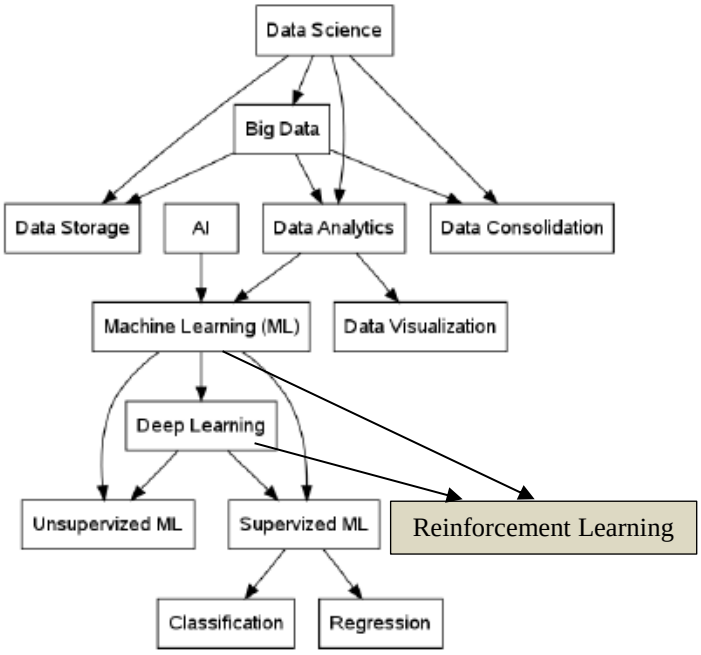


Figure 2: Data Science, AI and ML illustration (not exhaustive)

2.2 Artificial Intelligence and Machine Learning

In computer science, **Artificial Intelligence** (AI), sometimes called machine intelligence, is intelligence demonstrated by machines, unlike the natural intelligence displayed by humans and animals. AI is the field of study of intelligent agents or, in other words, any device that perceives its environment and takes actions that maximize its chance of successfully achieving its goals. Colloquially, AI is often used to describe machines (or computers) that mimic cognitive functions such as learning and problem solving.
AI was founded as an academic discipline in 1955 and AI research include reasoning, knowledge representation, planning, learning, natural language processing, perception and the ability to move and manipulate objects. For the last 20 years, AI techniques have experienced a resurgence following concurrent advances in computer power, large amounts of data, and theoretical understanding. AI techniques have become an essential part of the technology industry, helping to solve many challenging problems in computer science, software engineering and operations research.

Machine learning (ML) is a **subset of AI**. It is a fundamental concept of AI research since the field's inception and the study of computer algorithms that improve automatically through experience. ML algorithms build a mathematical model based on sample data in order to make predictions or decisions without being explicitly programmed to do so. ML algorithms are used in a wide variety of applications, such as email filtering and computer vision, where it is difficult or infeasible to develop conventional algorithms to perform the needed tasks.
Many techniques of non-deep machine learning (as opposed to deep learning) rely on tuning decision tools (functions, decision trees…) to help humans understand the structure of their data. As the decision tools are easily readable, we can easily explain and trust the outcomes of the training of these algorithms. **Deep learning** is a **part of machine learning** methods based on Artificial Neural Networks (ANN). When a problem is too complex for non-deep methods, and enough data is available, we can rely on **deep neural networks** (with multiple layers) that act like black boxes but provide new capabilities, like better time series and image processing. These models are not designed to totally replace models based on physics, but to work with them to improve knowledge and performance.

There exists three main types of learning from data (with deep, or, non-deep learning): supervised and unsupervised learning and, an extension of these algorithms that is **Reinforcement Learning (RL)**. Supervised learning allows to figure out the link between inputs data and the associated labels. Unsupervised learning search to identify and learn patterns in large dataset. RL purpose is to teach an agent how to behave and make decision in a given environment to achieve a given goal. Thus, RL differs from the two other paradigms because it doesn't need to bring a large dataset to learn the desired behavior. In RL data are generated interacting with the environment directly.

DOI: 10.13009/EUCASS2022-6308

3. GNC Autonomy, Neural Networks and Reinforcement Learning

Increasing GNC autonomy is one of the major challenges for next generation launchers. The potential of AI within this perspective is wide, and, in this paper, we will only deal with the use of Neural Networks as shown on Figure 3. In the scope of this study, we will use **Neural Network** structure trained by **Reinforcement Learning for the design of a nonlinear control algorithm.**

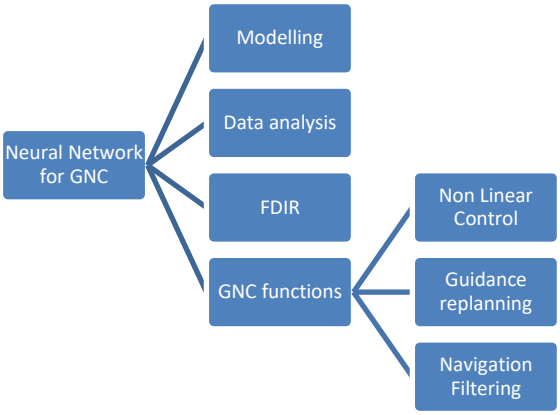


Figure 3: Illustration of some of the perspectives of using NN for GNC

3.1 Neural Network

Neural network are simple functions to approximate very complicated functions. It is “simply” a computer scientist view of function approximation.
An artificial neural network, usually called **neural network** (NN), is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal to other neurons. In other words, NN consist of inputs channels, cells body (or neurons) and output channel. Each neuron is associated with **weights**, bias, and **activation functions**. Weights and bias are adjusted as learning proceeds. The weight increases or decreases the strength of the signal at a connection; activation function can be linear or nonlinear. Typically, neurons are aggregated into layers. NN connection can be achieved though one single layer (perceptron) or multiple layers. An example of a neural network, called a fully connected multilayer perceptron, is presented on Figure 4.

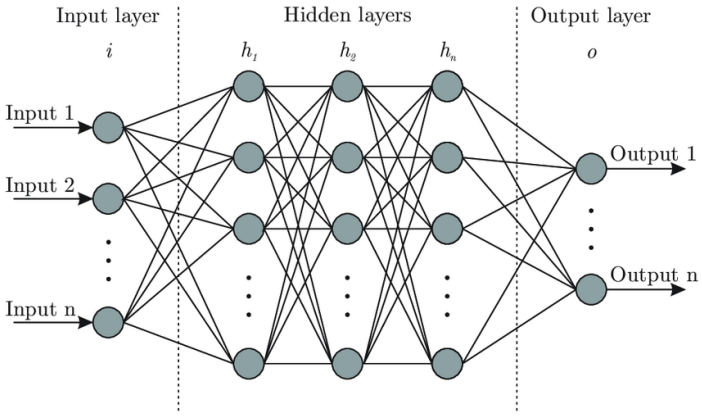


Figure 4: Artificial Neural Network

Simple, or complex, networks have their advantages and disadvantages. On the one hand, the advantage of simple NN (one single layer with linear activation function) is that thanks to the linearity of the system, the training algorithm will converge to the optimal solution (linear regression). This is not the case anymore, for nonlinear systems such as multiple layer networks for which training could be more complex. On the other hand, the disadvantage of single layer networks is their limited representational power to linear functions. On the opposite, the universal approximation

DOI: 10.13009/EUCASS2022-6308

theorem says that "only one layer of hidden units successes to approximate any function with many discontinuities to arbitrary precision, provided the activation functions of the hidden units are non-linear".
One can also view NN as weighted directed graphs, or, in discrete time, as a **non-linear time discrete filter**. It can be static when there is no delay, or, dynamic in case of delays.

3.2 Reinforcement Learning (RL)

"Reinforcement Learning is learning what to do (how to map situations to actions) so as to maximize a numerical reward signal. The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them" Sutton and Barto Reinforcement Learning – An Introduction
In RL we have an interactive environment that is able to provide a score (based on rewards and penalties) to the algorithm when it takes a decision. By iterating and constantly improving itself, the algorithm will be able of finding the best strategies that lead to the best score. This is of particular interest for GNC and FDIR functions.

RL for GNC people

Reinforcement Learning is a subset of Machine learning that is especially appealing for nonlinear control. Indeed, the concept of linear control is very simple: it is usually composed of estimation strategy, feed-forward and feedback control parts, and, allocation systems. It becomes more complex for **non-linear** systems, and, the interest of Reinforcement Learning is to help concentrating the whole controller into a single function (namely a Neural Network) directly designed from actions and observations.

The theory is wide, the notations are complex and different from control theory for a non-initiated person. We therefore seek to explicit here every new concept by linking it to control's concepts, so as to make RL theory understandable for someone with a background in control theory. **Reinforcement Learning** is a machine learning method where **agent** is interacting with the **environment** to approximate an **optimal policy**. It uses trials and errors and receives a delayed **reward** to evaluate its **action**. Associated to NN it is capable of solving complex tasks.

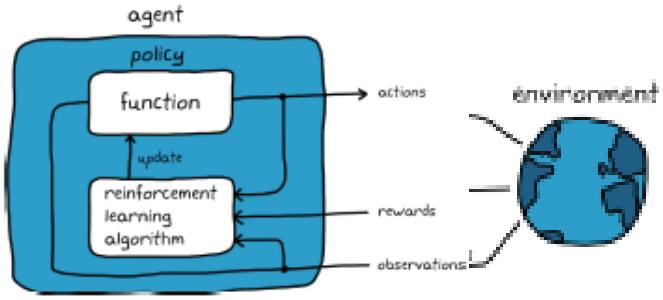


Figure 5 Illustration of RL notations (Mathworks courtesy)

Definitions

Agent is an abstraction of the control system. Its role is to learn how to achieve the control and to perform the desired goal. It is *a policy (or function) tuned by RL algorithm*

Environment represents the rest of the universe, it is everything outside the agent. It could be a simulator or the real world or environment

Action (A) represents all possible moves of the agent, e.g. actuation signals. In control notations, it is the *command u(t)*, the output of the control law.

State (S): represents the current situation returned by environment. In control notations S(t)=x(t), or Y(t) if only measured outputs are taken into account

Observations (O) is the state feedback signal. In control notations O(t)=Y(t)

Policy (π) is the strategy used by the agent. It corresponds to the ***control or guidance law***. It could be PID, Predictor/Corrector, NN …

Reward (R) is a real number characterizing how good, or not, an action is. It represents the *present***.** It is a short term return from environment (It could be any function: linear, quadratic, nonlinear, sparse reward +/-1 or 0 …).

DOI: 10.13009/EUCASS2022-6308

The notion of *return* is the discounted cumulative sum of rewards obtained for a given instant. The return concept is similar to LQR cost function in control theory. It is more general than LQR (quadratic function) since there is no restriction on the reward function structure. The reward has to be designed by the end user and it represents one of the most challenging part of using reinforcement learning

*Value (V) is the future***.** It is the expected long term return (total amount of reward an agent can expect to accumulate over the future of current state under policy π. It could be seen as the mission or *performance requirements* such as final consumption or accuracy for rendezvous. The value function concept is also used in Model Predictive Control or Optimal Control theory.

Qvalue (Q) is the long term return of current state S taking action A under policy π. It contains a *mapping of all possible* actions and resulting states.

Primal/Dual search: RL is closely linked to optimal control theory. In link with Bellman optimality principle, **RL** could be *policy search (primal)* or *value function based search (dual).*

Actor/Critic: It is a combination of primal and dual approaches (concurrent or combination of policy and value searches). An actor/critic learning system contains two distinct subsystems, one to estimate the long term utility for each state (e.g. NN to estimate the best value function) and another to take the decision and chose the optimal action in each state (e.g. NN to take the decision). The actor is the policy based part that generates new cases (could be NN or other control law). The critic is the value based part that takes the decision (could be NN or decision tree) as the agent explores the state space, the critic evaluates actions in the context of the policy's (on-policy or off-policy) current expected performance level (the advantage). This evaluation is then used to estimate a gradient for the parameterized policy.

Having all these relations in mind, from a pure practical point of view, for using RL algorithm for control application, we have to select the simulated environment (dynamics and perturbations), the structure of the policy (NN size and characteristics), the algorithm (e.g. PPO and the critic structure if any, and its hyper parameters such as learning rate …) and, last, but not least, the reward.

Some rough and non-exhaustive comparisons with control methods

One may also see RL as a stochastic approach and an extension of a lot of control methods:

- **Extension of LQR** to nonlinear system with non quadratic cost function.
- The stochastic part could be related to **fuzzy logic**
- **Optimal control theory** where perfect model would is replaced by the environment
- Systematic and automated tuning of the control parameters in **nonlinear** domain.
- On-line training could compared to **adaptive control**
- Using RL for off-line training is similar to using **Genetic Algorithms (GA).** However RL has an additional notion of progression/cumulative reward during each simulation.

3.3 Reward engineering

The design of the reward function is a fundamental step. Indeed, the reward function describes the purpose that the agent must fulfill. An inappropriate reward function will lead the agent to learn a different goal than the desired one. Although there are some cases of study for which the design of the reward function is very simple (it is the case of video games which intrinsically express a notion of success or failure), it is in most cases delicate to design a good reward function,
A reward function maps states and actions to real number$\forall s \in \mathcal{S}, \forall a \in \mathcal{A}, r(s,a) \in \mathbb{R}$. We distinguish two main types of reward functions:

- Sparse reward functions whose values belong to a countable set for any state-action pair;
- Dense reward functions which conversely have values in an uncountable set.

Generally, sparse reward functions are the simplest to understand. They allow to express easily the notion of success or failure. These types of reward functions are generally the easiest to design but have an important drawback. At the beginning of the training, the agent's actions are random and the probability of observing the rewards (i.e. of achieving success) is very low and therefore it can be infeasible in a reasonable amount of time. The second type of reward function is often preferred in continuous system control applications because these functions can be related to the cost

DOI: 10.13009/EUCASS2022-6308

functions used in optimal control theory. Finally, it is possible to take advantage of the two approaches by designing a dense-sparse reward function.
Reinforcement learning is a single-objective framework. Indeed, the objective of reinforcement learning algorithms is to optimize a single criterion: the reward. However, in most applications, the goal is to optimize a set of criteria. Therefore, most of the reward functions are built by combining the different criteria into a single criterion using weights. These weights are therefore hyperparameters that must be tuned.
There are an infinite number of possible reward function. So, in order to figure out which reward function is well suited we have to:

- choose the profile of the reward: linear, exponential, quadratic or other, convex reward function facilitates the convergence.
- decide how to combine the several terms: sum them, multiply them or other;
- mathematically formulate the performance goals, especially the smooth actuation one;
- Parameterize the relative weights.

The most relevant is to follow this methodology on a simple case and then step by step increase the complexity and adapt the reward function.

3.4 A little bit more about RL algorithms

Now that we have introduced the basic framework of reinforcement learning, we have to formalize how an agent learns. To maximize the expected return, an agent build what we call a policy.
To achieve the desired behavior the agent has to build a policy to map states to actions. All policies are not good and some policies are better than others. We can evaluate and compare policies using action-value functions. A policy is said to be better than another if its associated action-value function is greater or equal to the one of the other policy. From this emerges that there exists at least one policy that is as good as or better than all the other policies. This policy is called the optimal policy. The goal of reinforcement learning is to have an agent learn the optimal policy of a given problem. All optimal policies have the same unique value function which is the optimal value function. Finding this optimal policy by brute-force is absolutely intractable due to computational limitations. The agent has to find a way to approach the optimal policy in limited time.
Find out the optimal policy is not something easy. In the discrete case the resolution of the optimal bellman equations allow to find the optimal policy. However, this method is only applicable to very simple cases due to its computational limitations. In most classical problems finding the optimal policy is not guarantee.

Reinforcement learning provides a large range of algorithm to approach the optimal policy. There are a lot of methods to approach this optimal policy. In the followings we will introduce policy optimization a set of methods probably the most used to solve continuous control problems. These methods are iterative and seek to approach the optimal policy through successive policy improvement from a randomly initialized policy.
Approaching the optimal policy can be done by solving a constrained optimization problem. The disadvantage of classical gradient methods is their instability. Indeed, policy gradient methods suffer from the same problem. At each update we observe "jumps" i.e. the new calculated policy is very far away from the previous one. The updates are too large. To remedy this [5] introduced the Trust Region Policy Optimization (TRPO). The idea of trust region methods is to search the best policy improvement in a limited region called trust region. The formulation of this problem is the basis of the TRPO method proposed by [5]. The mathematical approach developed by authors shows that the TRPO method ensures to improve the policy at each update if the approximations made are reasonable. However, it appears that to be applied this method requires a complex second-order optimization that is very computationally intensive. To overcome this problem [6] introduced the PPO method which proposes an alternative to the TRPO method while keeping its philosophy.

There exist 2 main open source libraries in Python to use reinforcement learning algorithms: Keras-RL and Tensor Force. The literature on spacecraft and missiles control using reinforcement learning shows that policy gradient methods are widely used methods. Among policy gradient methods the most used are Trust Region Policy Optimization (TRPO) and its improvement the Proximal Policy Optimization (PPO). Indeed many papers in the literature mention the use of the Proximal Policy Optimization (PPO) algorithm, This algorithm is part of a new family of policy gradient methods for RL, which alternate between sampling data through interaction with the environment, and optimizing a “surrogate” objective function using stochastic gradient ascent. Whereas standard policy gradient methods perform one gradient update per data sample, these methods propose a novel objective function that enables multiple epochs of minibatch updates. The new methods have some of the benefits of trust region policy optimization (TRPO), but they are much simpler to implement, more general, and have better sample complexity (empirically).

DOI: 10.13009/EUCASS2022-6308

Algorithm 1 **Proximal Policy Optimization Algorithm with clipped objective**

Inputs:

- Initial policy parameters $\boldsymbol{\theta_0}$
- Initial value function parameters $\boldsymbol{\psi_0}$

Output: final policy parameters $\boldsymbol{\theta_N}$

for iteration $\boldsymbol{k = 1,2,\ldots N}$ **do**

Collect set of trajectories $\boldsymbol{\mathcal{D}_k = \{\tau_i\}}$ by running policy $\boldsymbol{\pi_{\theta_k}}$ in the environment.

Compute rewards-to-go $\boldsymbol{\widehat{R}_t}$

Compute advantage estimates, $\boldsymbol{\widehat{A}_t}$ (using any method of advantage estimation) based on the current value function $\boldsymbol{V_{\psi_k}}$.

Update the policy by maximise the PPO-Clip objective

$$\boldsymbol{\theta_{k+1} = \arg\max_{\backslash thet} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), g\left(\varepsilon, A^{\pi_{\theta_k}}(s_t, a_t)\right) \right)}$$

typically via stochastic gradient ascent with Adam.

Fit value function by regression on mean-saured error:

$$\boldsymbol{\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \widehat{R}_t\right)^2}$$

typically via some gradient descent algorithm.

end for

https://spinningup.openai.com/en/latest/algorithms/ppo.html

DOI: 10.13009/EUCASS2022-6308

4. Study cases

The purpose of this study is to explore new intelligent control strategies for launchers based on reinforcement learning. Our use case is the control of the boosted phases of the upper stage of a launcher. During this phase the launcher is reduced to its third stage and its payload(s). During the boosted phases, the thrust is provided by the main engine. This main engine pushes during predetermined durations separated by orbital phases.
In theory, just before the boost ignition (or re-ignition) the launcher is headed in the right direction and must simply push in the direction commanded by the guidance system during a given time. However in practice, a lot of disturbing phenomena led to the necessity of actively control the launcher during these boosted phases. Namely the initial conditions at the beginning or the boost (or reboost) are dispersed, and, we have to manage the motion of the ergols in the tanks. Furthermore, this control problem is typically nonlinear (with saturations and varying parameters such as thrust level), with a lot of uncertain parameters, and with perturbations (among which the main ones are deflection bias or delays).
We have defined an incremental approach with numerous study cases starting from simple rigid model without ergol motion and sloshing modes and without uncertainties, up to complete model. This incremental approach presents the advantage of building a solution step by step by stacking blocks through iterations, making the final solution easier to understand and to reproduce.

4.1 Objectives

For all the study cases, the control objectives are :

Stability that shall be ensured with sufficient margins

Recovery from initial condition at the beginning of the boost. The sloshing angle and sloshing angular rate, the attitude and the attitude angular rate may deviate from the nominal configurations that is zero deviation.

- During the boost the maximal sloshing angle is limited so as the maximal attitude and maximal angular rate.
- At the end of the boost duration, the desired attitude and rate final conditions shall be reached
- At the end of the boost duration, the sloshing angle and the sloshing rate shall be brought back to zero with a characteristic time low compared to the natural damping characteristic time

The *transverse velocity* at the end of the boost shall be *minimized*

The *maximal deflection and deflection rate* are limited

The thrust *deflection bias* shall be taken int account

And depending on the representativeness of the study case, we shall take into account some, or all, of the followings :

1. The *sloshing perturbations*. When tanks are not full, sloshing phenomena appear disturbing the launcher equilibrium and its performances
2. *Delays* in the loop (actuators, sensors, and computation)
3. *Functioning domain* : The range of prameters (mass, centering, inertia, thust level and sloshing characteristics) to cover the whole set of missions and payloads
4. *Uncertainties* : The uncertianties on the physical paramters : rigid and sloshing modes
5. *LPV* : The time variation of the thrust profile since the thrust depends on the engine states (transient states or steady state) and the sloshing parameters variation function of draining ratio.

NOTE – In RL could also replace the pendulum sloshing model by a real FLOW3D simulation.

4.2 Models

An overview of the incremental definition of our study cases is presented in this section. The results obtained for 2 study cases are presented in next section: the first one contains only the rigid dynamics and the second one contains 2 sloshing modes.

The system to be controlled contains:

- A measurement model delivering filtered attitude, angular rate and non-gravitational acceleration measurements

DOI: 10.13009/EUCASS2022-6308

- An actuator model with dynamics, bias and saturations
- A loop delay
- A launcher model with 2 tanks. The ergols sloshing in the tanks is modelled as a pendulum. It means that the fluid rising on the walls of the tank behaves like a static fluid at the bottom of the tank and a pendulum swinging from left to right attached above the tank. A state space representation is given hereafter for illustration for rigid dynamics and 1 sloshing mode.

Launcher dynamic with 1 sloshing mode

The state vector is $X = [\alpha \quad \dot{\alpha} \quad \theta \quad \dot{\theta}]^T$. Where the state variables are θ the launcher attitude, $\dot{\theta}$ the attitude rate, α the sloshing angle and $\dot{\alpha}$ the sloshing rate.

The launcher is controlled by acting on the thrust deflection and defection rate. We note β_c the commanded deflection angle, and of the nozzle β_p the bias on the commanded deflection angle. The achieved deflection angle is$u = \beta_r = \beta_c + \beta_p$.

Using the Euler and Newton equations we obtain the state representation of the system:

$$\dot{X} = AX + Bu$$

Where

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ a & da & 0 & 0 \\ 0 & 0 & 0 & 1 \\ g & dg & 0 & 0 \end{bmatrix} \text{ and } B = \begin{bmatrix} 0 \\ 0 \\ b \\ \frac{Tc \cdot Lt}{I} \end{bmatrix}$$

Matrices coefficients are:

$$a = -\omega^2 \left(1 + \frac{mb}{M} + \frac{mb \cdot La \cdot (La - Lp)}{I}\right)$$

$$da = -2\xi\omega \left(1 + \frac{mb}{M} + \frac{mb \cdot (La - Lp)^2}{I}\right)$$

$$g = -\omega^2 \cdot \frac{mb \cdot La \cdot Lp}{I}$$

$$dg = -2\xi\omega \cdot \frac{mb \cdot Lp \cdot (La - Lp)}{I}$$

$$b = -\frac{Tc}{M \cdot Lp}\left(1 - \frac{M \cdot Lt \cdot (La - Lp)}{I}\right)$$

Where:

- M launcher mass without sloshing masses;
- Xg coordinate of the launcher center of mass without sloshing masses;
- I launcher transverse inertia without sloshing masses;
- mb sloshing mass of the sloshing mode;
- γ launcher acceleration $\frac{Thrust}{M+mb}$;
- Lp pendulum length of the sloshing mode;
- ω eigen-pulsation of the sloshing mode under acceleration ($\omega = \sqrt{\gamma/Lp}$);
- ξ natural damping of the sloshing mode under acceleration ($\xi = \xi^{1g}\sqrt[4]{9{,}81/\gamma}$);
- Xa pendulum attachment point coordinate of the sloshing mode;
- La algebraic distance between pendulum attachment point and launcher center of mass without sloshing masses ($La = Xa - Xg$) for the sloshing mode;
- Lt algebraic distance between nozzle center of rotation and launcher center of mass without sloshing masses ($Lt = Xg - Xt$).

DOI: 10.13009/EUCASS2022-6308

5.Application and results

5.1 Interface Python / Matlab-Simulink for industrial purpose

For the implementation of the environment, we decided to use the native environment interface of Tensorforce to follow the logic of the library. The dynamic of the environment is implemented in a Simulink model which is interfaced with Python using a dll. We choose Simulink environment since it is commonly used by the GNC community for modelling and simulating the vehicle dynamics. Matlab/Simulink is also widely used for the design of linear controllers.

5.2 Incremental approach

A first straightforward approach would be to try to solve the entire problem directly. For instance we can apply a large range of RL methods to the problem with more or less complex architecture and figure out which ones have good performances. This sort of brute-force strategy is often adopted in machine learning by beginners because it requires few previous knowledge of the field and avoid missing a potentially interesting solution. However this strategy has huge drawbacks and shall not be used. In fact, trying a lot of solutions require a huge amount of time and computational resources making this approach most of the time result-less. Moreover, this strategy prevents one from understanding the methods used. Finally, even if a solution would be found, it would be a black box very difficult to explain and it would not be possible to transpose the solution to a similar problem.
A better strategy is to be inspired by existing works on similar problems and even, if it is possible, to reuse an existing solution and adapt it to our problem. This allows to find quickly which solution is well suited to our problem and which hyper-parameters may work. So, an iterative approach starting from a very simplified version of the problem and then complexify it step by step restarting from the previous solution is more relevant.

5.3 Results on a rigid test case

We started by training a neural network on the simple rigid dynamic model without deflection bias. Thus, the system is fully described by the simple equation $\ddot{\theta} = K_1\beta$ which is the equation of a double integrator. According to the linear control theory a double integrator is controllable, and, a closed loop second order dynamics can be imposed by using only two gains: one on the attitude K_p and one on its derivative (attitude rate)K_d. Based on this classical result we chose a very simple structure for the policy neural network. It is composed of only one hidden layer which is a dense layer with one neuron (see Figure 6). The design of the baseline network is also as simple as possible.

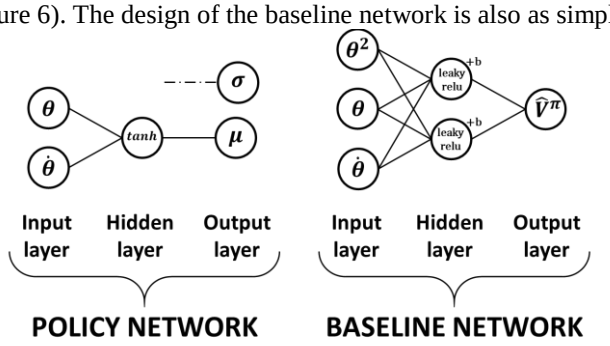


Figure 6: Policy and baseline structure for the rigid test case

The policy output is computed as follows.

$$\mu(t) = w_3 \tanh\left(w_1\theta(t) + w_2\dot{\theta}(t)\right)$$

A linearization of this expression shows that the policy neural network is very similar to the classical proportional-derivative controller from the control theory.
Then, we also take into account, in this study case, deflection bias, delays and uncertainties.
Through this application, we were able to gain insight the design methodology, the selection of the PPO algorithm hyper parameters and the definition of the simplest neural networks structure and the simplest reward (quadratic reward with time dependency to favor final recovery).
The agent was trained on more than 2000 episodes and then evaluated on 20 episodes. The temporal responses of the system for the 20 episodes of evaluation are plotted below. One may clearly see on Figure 7 that the recovery from

DOI: 10.13009/EUCASS2022-6308

initial conditions is easily achieved in a few seconds. Increasing the training duration could even improve these results illustration. Since we are using quadratic based rewards, one may roughly say that we were able to expand linear LQG control and estimation theory to non-linear controllers. This should be more deeply analyzed in the future in the frame of a PhD.

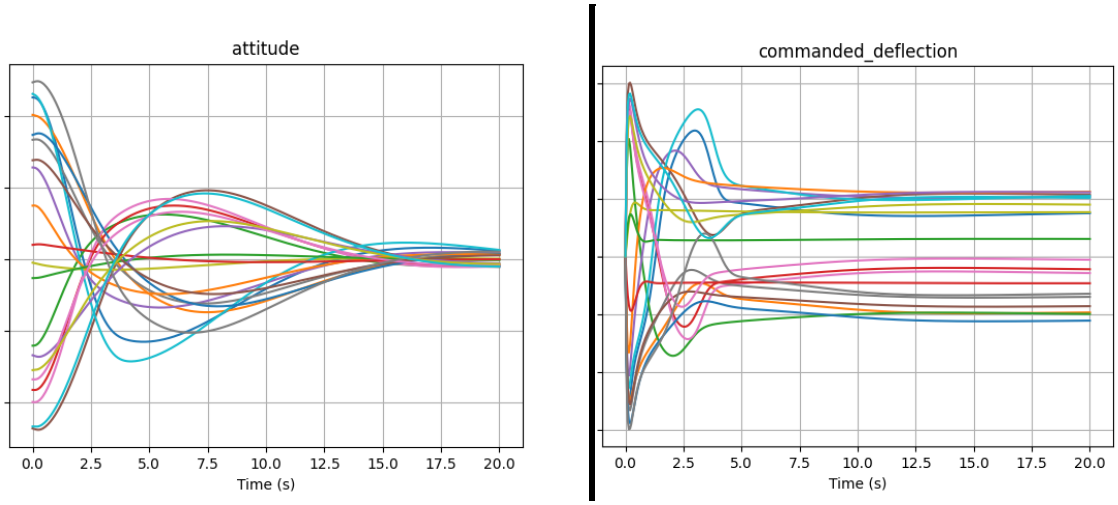


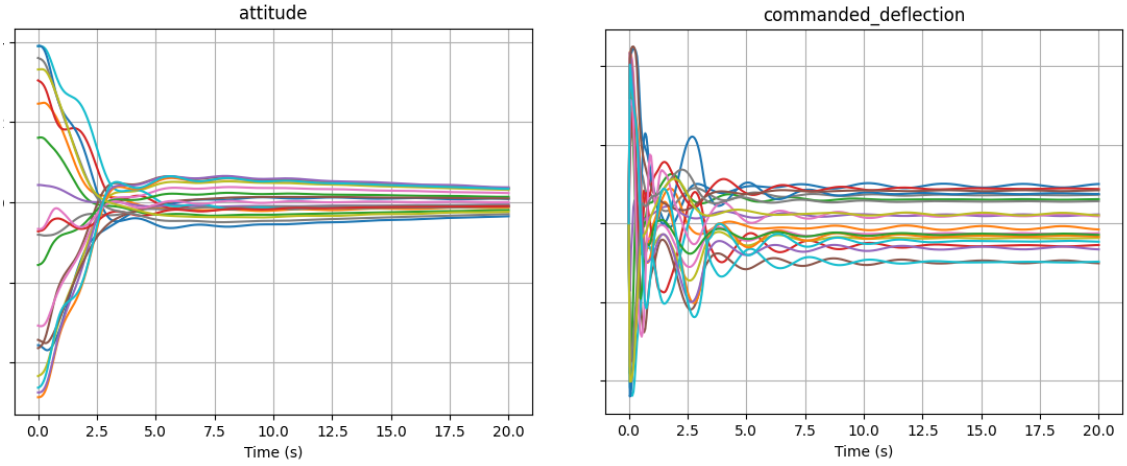
Figure 7: Simulation results – Rigid study case

5.4 Results with sloshing perturbation

In this second study case, we add two sloshing modes to the rigid dynamics, and, we use only the available measurements that are delivered by the Inertial Measurement Unit (IMU). Thus the sloshing angles are not known and the controller has to control the sloshing perturbations that can be seen in the measurements.
It was necessary to increase the size of the neural network and to increase the training number of episodes. The results are very convincing since the NN controller succeed in controlling simultaneously the motion of the sloshing modes and the rigid mode, in less than 10s (Figure 8). The second sloshing mode is not directly controlled in this example, therefore, some oscillations remain on the simulation results.

Such performance were not possible with the linear controller that we had designed at the beginning of this study. It takes either longer time to control all the phenomena, or, it enables dealing only with rigid mode control, or, sloshing mode control in the same amount of time. Furthermore one may notice that attitude and sloshing angle are always decreasing along the time, therefore, whatever the simulation duration is, the performance will be improved.

Remaining work is to extend this application to the whole system taking into account the control of the 2 sloshing modes, the LPV parameters and the whole mission coverage. Then we shall compare the results with linear controllers obtained by LQG or structured H∞ methods. An extension to using LQG or Lyapunov approach to guarantee the stability and robustness of the non-linear controller is also planned in the future research activities.



DOI: 10.13009/EUCASS2022-6308

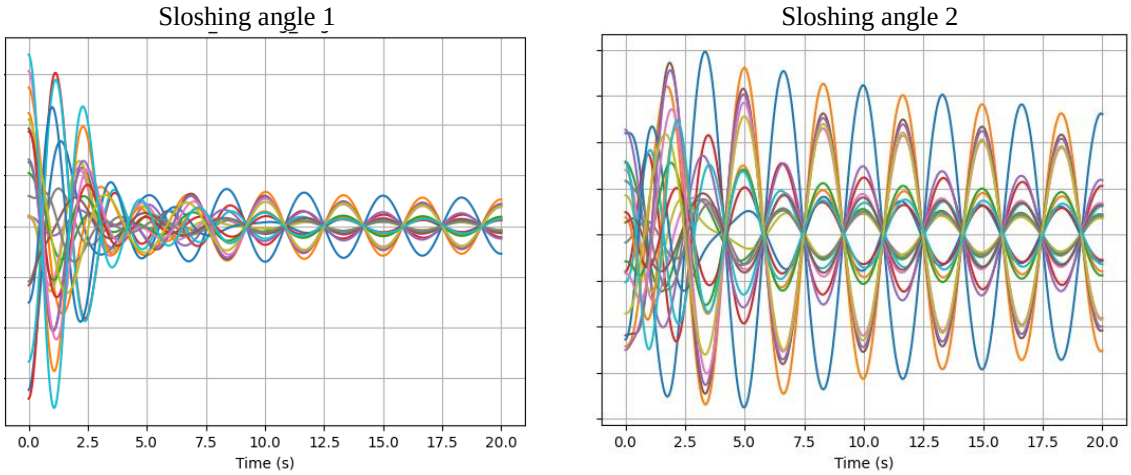


Figure 8: Simulation results Study case with two sloshing pendulum – Without sloshing angle measurement

5.4 Analysis

The stability and performance of the non-linear controller are evaluated thanks to time domain simulations. In addition, during learning we have access to the following information that give indications on the convergence of the algorithm. The reward/return directly illustrates the agent's learning. Indeed, if the absolute value of reward/return decreases during the learning it means that the agent improves and that the goal defined by the reward is more and more achieved by the agent. When a plateau is reached it is because the agent no longer learns. In contrary, if we observe no increase in reward/return over the training it means that there is no learning. Figure 9 represents rewards/returns obtained by successive updates of the agent, each time for different episode. It means that a local decrease may not reflect a bad update but maybe that the episode ran had less favorable conditions. It is therefore better to consider the average evolution.

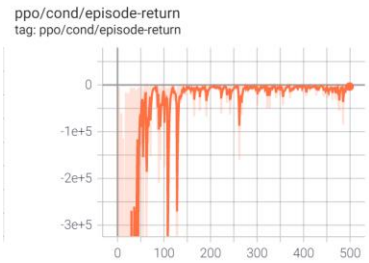


Figure 9: Example of Tensorboard plot of return over episodes

The weights and bias of the neural network can also illustrate the learning convergence. Sometimes some issues can be observed as shown on figure 10 where at the end of the training something happens that stops the training. A lot of work remains to be done on the analysis of the convergence of the algorithm. It is not the purpose of this paper where we only illustrates some useful results for engineering understanding

DOI: 10.13009/EUCASS2022-6308

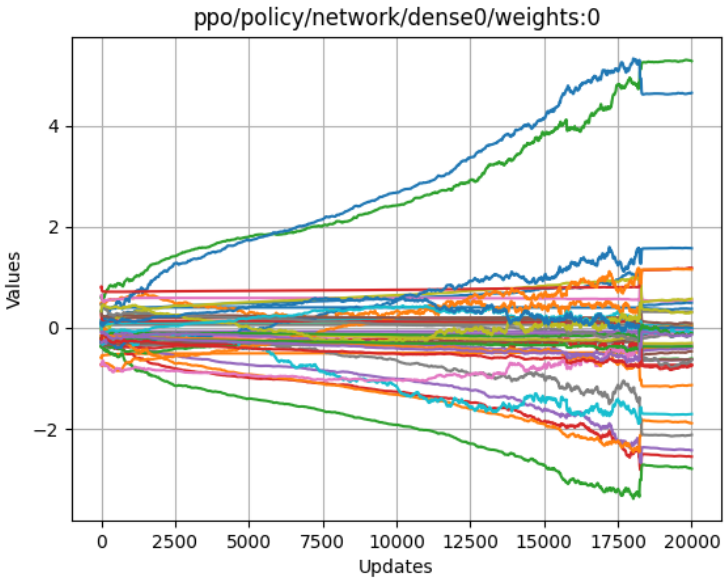


Figure 10: Illustration of policy weights during learning when an error occurs

6. Conclusion and perspectives

In this paper, we have gained insight the application of reinforcement learning, for training a non-linear neural network controller, for active sloshing control during launcher upper stage boost phases. Using an incremental approach we were able to understand the design methodology, select PPO algorithm hyper parameters, define the simplest neural networks structure, and, most of all, define simple and efficient reward function. We are now able to combine the time domain approach of reinforcement learning algorithms, and, the non-linear structure of neural networks, with our experience on frequency domain based linear control methods. On some study cases, representative of active sloshing control during launcher upper stage boost phases, the resulting non-linear controller performs better than a linear controller, that either takes longer time to control all the phenomena, or, that enables dealing only with rigid mode control, or, sloshing mode control in the same amount of time.

Remaining work is first to extend this application to the whole system taking into account LPV parameters and ensuring the whole missions coverage and compare the results with robust linear controller obtained by LQG or structured H∞ methods.

Then we still have to work on deeper understanding and simplification of the NN to try to enhance the reproducibility, and, expand the development towards transfer learning from one application to another.

Finally, even if in our application, the neural network is trained off-line, certification remains a key issue for on-board implementation. One perspective, to certify machine learning for on-board application, is to combine Automatic control field of competences (e.g. LQ or Lyapunov methods to guarantee the stability and robustness of the non-linear controller) with Artificial Intelligence ones, as illustrated on Figure 11. This is the topic of PhD cofounded by AGS and ONERA.

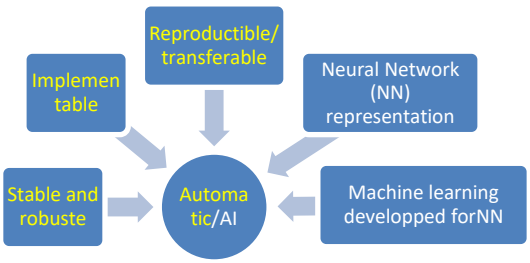


Figure 11: Illustration of unification of Automatic and IA fields of competences

DOI: 10.13009/EUCASS2022-6308

Acknowledgements

We would like to thank Dr Florian Peter for our fruitful exchanges on Simulink/Python interfaces.

References

[1] R. Sclafani and P.Shankar, "Variable Memory Recurrent Neural Networks for Launch Vehicle Attitude Control", AIAA Scitech 2015-1774
[2] G. Gelly_ and P. Vernis, Neural Networks as a Guidance Solution for Soft-Landing and Aerocapture, AIAA 2009-5664.
[3] RC. Sanchez-Sanchez, "Optimal Real Time Landing using Deep Networks, 2016.
[4] Brian Gaudet, Richard Linares, Roberto Furfaro, Adaptive Guidance and Integrated Navigation with Reinforcement Meta-Learning, Acta Astronautica · April 2020
[5] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel. Trust region policy optimization. 32nd International Conference on Machine Learning, ICML 2015, 3:1889–1897, 2015
[6] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal Policy Optimization Algorithms. pages 1–12, 2017. URL http://arxiv.org/abs/1707.06347

.