A Machine Learning Approach to Bio-Inspired Perching and Prey-Capturing

Research Project Code Resume

Author:

Arroyo Ramo, Andrea

Tutors:

Bauerheim, Michael
Jardin, Thierry
Rachelson, Emmanuel

Table of Contents

Table of Figures

1. Introduction

This document is just an aid to understand the developed code for the application of Deep Deterministic Reinforcement Policy Gradient algorithm applied to determined problem. This is the case of a flat plate without drag forces.

First, an explanation of the current problem to solve; second, files structure and, finally, the file by file code development.

2. Flat plate model

The implemented model is the flat plate model. It is the body motion in the air of a flat plate under gravity effect, its only acting aerodynamic force is lift; drag forces are neglected. See Figure 1: Flat plate model simplification. The x-y reference frame is linked to the centre of mass of the flat plate, however it does not rotate with respect to the horizontal plane.

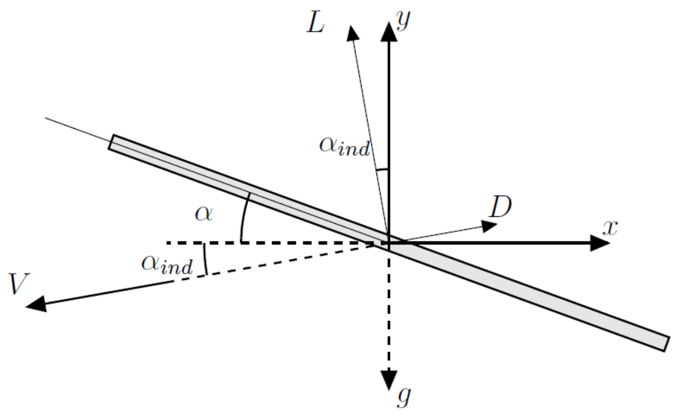


Figure 1: Flat plate model simplification. The x-y reference frame is linked to the centre of mass of the flat plate, however it does not rotate with respect to the horizontal plane.

The model has its associated set of differential equations that define the path and velocities on of the flat plate under low perturbances/angles assumption.

X axis:

$$m \cdot a_x = -D = 0$$

$$a_x = \frac{\partial u}{\partial t} = 0 \rightarrow u = cst$$

$$u = \frac{\partial x}{\partial t}$$

Y axis

$$m \cdot a_y = -m \cdot g + L\,; L = \frac{1}{2}\rho \cdot S \cdot V^2\big(2\pi(\alpha + \alpha_{ind})\big)$$

$$a_y = \frac{\partial v}{\partial t} = -g + m_r \cdot V^2(\alpha + \alpha_{ind})$$

$$v = \frac{\partial y}{\partial t}$$

Where:

- $m_r = \frac{\rho \cdot S \cdot \pi}{m}$ is the reduced mass.
- $\alpha_{ind} = \mathrm{atan}\left(\frac{v}{u}\right)$ is the induced angle of attack, which modifies the effective angle of attack that depends on the ascending or descending flat plate trajectory. The induced drag force is neglected since the induced angle is small, its contribution is also very reduced.

3. Reinforcement Learning algorithm

The chosen Reinforcement Learning algorithm is the Deep Deterministic Reinforcement Policy Gradient algorithm. It is already implemented by the Département Ingénierie des Systèmes Complexes (DISC). Nevertheless, the given implementation uses an environment (*gym* environment) that has been created and tested to be used in Artificial Intelligence development. This environment acts as a black box where the action to perform is introduced and the next state, reward and other parameters are extracted.

The main difficulty of this code is to be able to replace *gym* environment by the set of differential equations that compose the flat plate model.

At this point the desired path to be followed is the straight line between initial and final position.

4. Code files

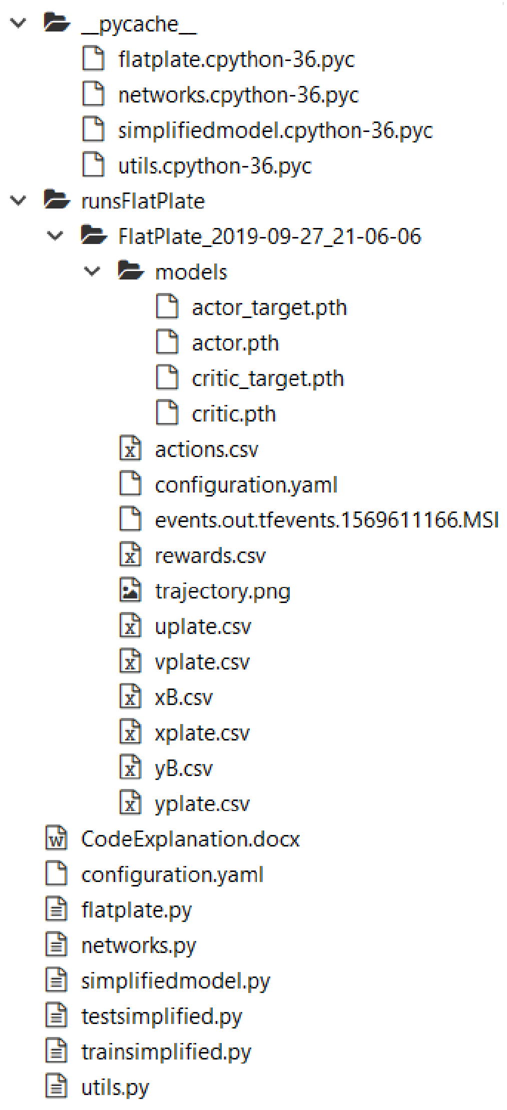
In this section there is a list with all the *.py* and other files types used for coding the DDPG algorithm for a flat plate as well as the folders structure.

- *configuration.yaml*: set up configuration for the DDPG, initial and final conditions of the model.
- *utils.py*: set of useful functions to get information about directories and information treatment.
- *networks.py*: designed neural network. The actor and critic with their own networks.
- *model.py*: (previously named *simplifiedmodel*)_implementation of the DDPG algorithm. It decides the action to be performed according to the information of previous actions (the obtained reward, efficacy of the action…) according to the algorithm.
- *flatplate.py*: substitution of *gym* environment by the flat plate one. It has its own functions in order to mimic the capabilities of *gym* environment
    - *__init__()*: initialisation of all variables and constants to define the flat plate properties.
    - *flatplate*: set of differential equations that must be solved to compute the dynamics of the flat plate's evolution.

    - *step(action)*: with a given action, it returns
        - *compute_reward()*: reward computation derived of the performed action. Ponderation to how close to the straight line path between initial and final position.
        - *isdone()*: check if criteria to finish the current episode are fulfilled (be near the final position or exceed the x coordinate value).
    - *reset()*: restart the problem to the original point A
- *trainsimplified.py*: main code. Train the model with the specific flat plate and the DDPG algorithm.
- *testsimplified.py*: test the obtained values of the model with training.

Each time *trainsimplified.py* is run, it creates a folder (*runsFlatPlate_year-month-day_hour-minute-second*). It contains the generated model when training that precise run inside the folder *model*. Also, it has different .csv files with the trajectory, velocity and rewards per episode and step.

The content in .pth files defines the characteristics of the DDPG model generated. It is loaded to test the model or to continue with the training in other occasion.



```
v _pycache_
    flatplate.cpython-36.pyc
    networks.cpython-36.pyc
    simplifiedmodel.cpython-36.pyc
    utils.cpython-36.pyc
v runsFlatPlate
  v FlatPlate_2019-09-27_21-06-06
    v models
        actor_target.pth
        actor.pth
        critic_target.pth
        critic.pth
      actions.csv
      configuration.yaml
      events.out.tfevents.1569611166.MSI
      rewards.csv
      trajectory.png
      uplate.csv
      vplate.csv
      xB.csv
      xplate.csv
      yB.csv
      yplate.csv
  CodeExplanation.docx
  configuration.yaml
  flatplate.py
  networks.py
  simplifiedmodel.py
  testsimplified.py
  trainsimplified.py
  utils.py
```

Figure 2: Folder development with all the files and sub-folders

5. Main file

The main file is *trainsimplified.py*. Its structure is:

```
Flat plate environment creation and initialisation
DDPG model creation
Initiate episode
        Start the steps for the episode
                Reset the environment
                Action selection
                Step computation to obtain the next state and reward
                Give information about action results to the model
                Next step
        Next episode
Save data
```

6. Additional information

The reward is computed by means of the relative coordinate system with the parameters:

- θ: angle between ideal path and current path.
- ρ_{ij}: modulus distance from point i to point j.

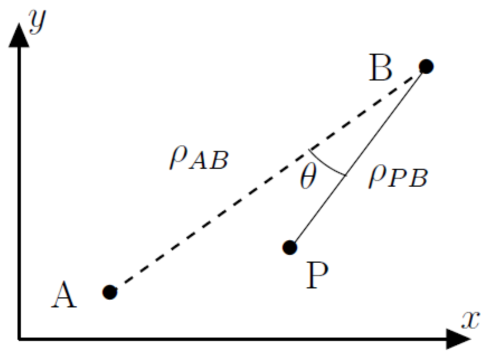


Figure 3: Reference system used to compute the reward

It gets the form:

$$r = 1 - k_1 \left(\frac{\rho_{PB}}{\rho_{AB}} + K_2 \cdot |\theta| + k_3 \left(1 - \frac{\rho_{PB}}{\rho_{AB}} \right) |\theta| \right)$$

The reward function is still under construction. Its topology is searched to have a lower value (more negative value) the farther from the actual straight line joining initial and final points. The different terms make a ponderation according to the distance to the final point, the deviation angle and increasing the importance of the deviation if the current point is close to the final point.

At the end, the DDPG algorithm maximises the reward while learning.

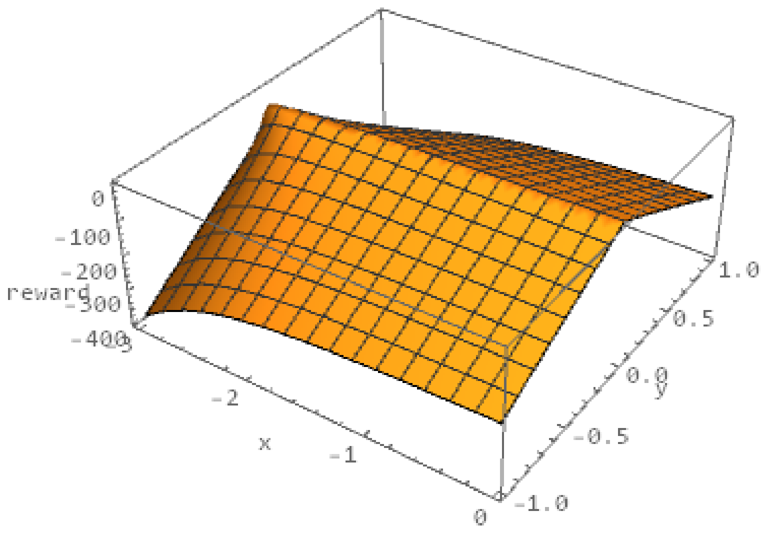


Figure 4: Example of reward topology