Stanford University

CS229 : Machine Learning

Final Report

Autonomous Aerobatic Airplane Control with Reinforcement Learning

Authors:
Jean de Becdelievre
Kevin Poulet
Bertrand Bevillard

Instructor
Andrew Ng
John Duchi

December 15, 2016

1 Abstract and Project Overview

This project aims at designing a flight controller for an aerobatics airplane. Airplanes have non-linear aerodynamics during highly banked turns and loops, which justifies the implementation of a machine-learning based control system to embrace the complexity of the model. The particular feature of this controller is to take advantage of expert demonstrations of the expected tasks, where the pilot performs the tasks while the controller is learning – in order to achieve controller training without airplane crash. In accordance with this strategy, *learning* and *controlling* phases are distinct. The ultimate goal is to have the airplane perform a given maneuver on demand. The training and testing have all been implemented using a flight simulator.
Starting from the usual 12-dimensional state space of an aircraft, the problem has been simplified down to 4 dimensions. Reinforcement learning is applied to the control of pitch, roll and of their associated velocities. This problem is modeled as a simplified Markov decision process where a deterministic approach is employed.

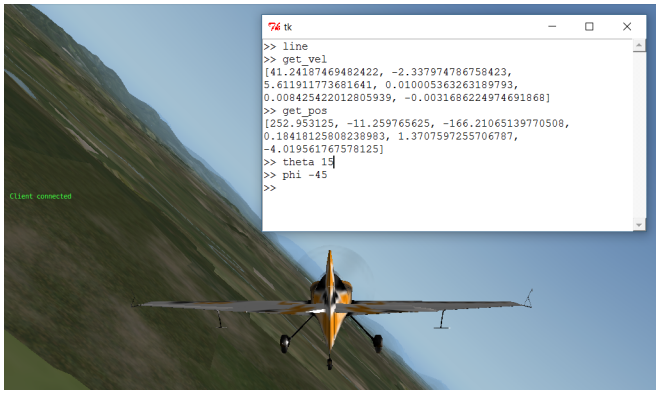
2 Flight Simulator User Interface

To ease the development process, we decided to use an off the shelf flight simulator called *X-Plane 10*[1]. The simulator offers the read/write access to most flight variables via a UDP server, which allows us to take control over the plane from our scrips. To communicate with the simulator UDP server, we used the python client of the plugin[?] [2] and wrapped our learning system around it.
The main program consists in three modules :

1. A communication wrapper class to send and receive data from X-Plane and to achieve number of basic functions such as getting and setting the plane state, pausing and resuming the physical simulation and updating the plane control surfaces

2. A custom console interface to handle user input output, useful for getting live information on the state of a controller or changing training variables in real time

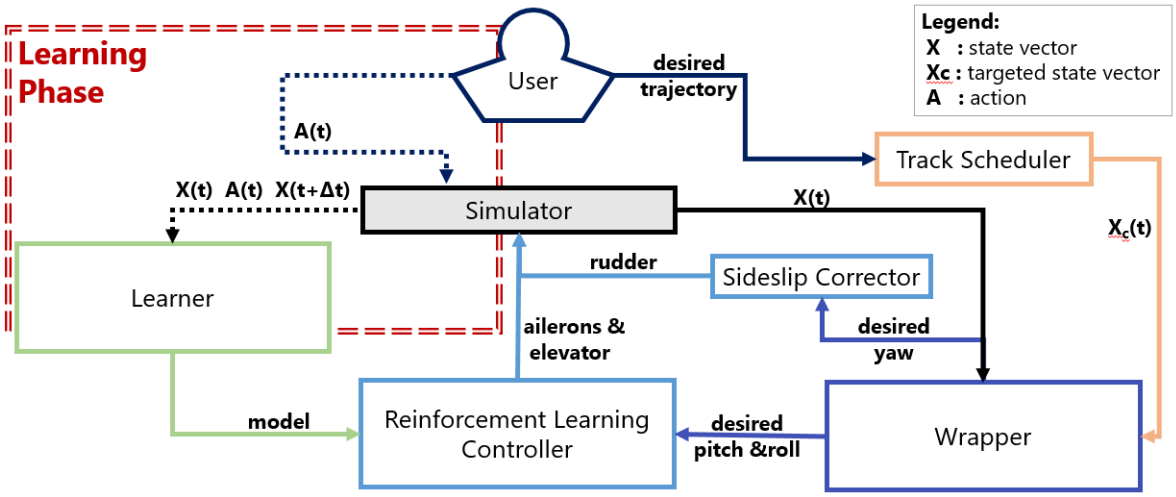
3. A controller engine capable of scheduling and managing learning and control processes.



The figure above shows an X-Plane window running a simulation along with the console, running a baseline bank/roll controller performing classical control feedback on these two angles

3 Control Strategy

Classical control techniques usually consists in determining the best commands to impose for a given state to minimize an error functional, usually taken to be a tracking measure. If the classical methods usually provide simple and robust ways to control a system, most of the time they require a good knowledge of its dynamics and perform poorly when the level of uncertainty raises.
Since the dynamics of the plane can get quite complex and difficult to model, we decided to turn toward a *reinforcement learning controller*, for its capacity to capture the complex behavior of physical system without the need to specify the laws of motion explicitly. More specifically, the dynamics are stored in a state transition matrix mapping a pair $(state, action)$ to a vector of transition probabilities on the state space.
The general control problem for a flying vehicle requires at least a 12-dimensional state representation to capture the evolution of the three Cartesian coordinates, the rotations around three axes as well as the time derivatives of these variables. The action state is usually comprised of the four usual aircraft commands : elevator, ailerons, rudder and throttle. Given the discrete nature of the transition matrix, we need to partition the state space. This quickly raises the problem of the *curse of dimensionality* : If we wanted a full state transition matrix stored in less that 1Gb, and assuming that each probability could be encoded in a float16, we would need to reduce each vector state to a ridiculous size of 4 values! To relax this constraint, we noticed that the dynamics of the plane had no dependence on its location and heading and that we could close a simple velocity feedback loop to stabilize it, and a rudder side-slip loop to prevent yawing instabilities.
These simplification led to the following controls structure



Given a trajectory, the track scheduler determines the target state given the position of the airplane. To minimize the tracking error, the wrapper determines what should be the heading, pitch and roll commands to impose. The pitch is synthesized from the heading using a simple gain feedback. From the desired *(pitch, roll, pitch rate, roll rate)* vector, a reward matrix is generated penalizing the distance away from the target, and the reinforcement controller runs a value iteration on these four states to find the optimal command doublet, *(elevator, aileron)*, then sent to the simulator. The dimensionality of the state/action space has been reduced to 6.

4 Pitch and Roll Control with Reinforcement Learning

4.1 Model

Markov Decision Processes with discrete state and action spaces were used to model this system. The state and actions are :

$$\vec{X} = \begin{bmatrix} \theta \\ \dot{\theta} \\ \phi \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} \text{pitch angle} \\ \text{pitch rate} \\ \text{roll angle} \\ \text{roll rate} \end{bmatrix} \text{ and } \vec{A} = \begin{bmatrix} d_e \\ d_a \end{bmatrix} = \begin{bmatrix} \text{elevator deflection} \\ \text{ailerons deflection} \end{bmatrix}$$

Two noticeable points can be emphasized about their application here:

- As part of our "learning while being flown by a pilot" strategy, the *learning* and the *controlling* phases are clearly distinct. During the learning phase, the probabilities of transition between two states are estimated, while this piece of information is used to pick the optimal action for a given reward during the *controlling* phase.
- The reward function is not learnt by the algorithm but provided by the *track scheduler*. In the *controlling* phase indeed the user specifies the desired trajectory, that is converted by the *track scheduler* into a time dependent reward function. The value function is then computed at each time step by running value iteration.

Practically, the discretization has been perform in the following way : From the memory requirements, we decided upon a partition of the state space, prioritizing the angular states over the rates, and the rates over the actions. We ended up with a partition of this form

$$\theta \in [-10 : 5 : 10] \quad \dot{\theta} \in [-80 : 30 : 80] \quad \phi \in [-45 : 10 : 45] \quad \dot{\phi} \in [-80 : 30 : 80]$$

With three deflections angles per control surfaces from $-20°$ to $20°$

4.2 *Learning*: capturing the Aircraft Dynamics

Theory: Training time and memory requirements quickly appeared as real challenges in this projects. Just as a driver that turns the steering wheel will get to a normally distributed new state, it seems very reasonable to make the assumption that transition probabilities are following a **Gaussian distribution** in this problem. So instead of computing and storing, for each state and action the probability to transition to every single new state, one could simply store the average new state and the associated covariance matrix.

$$P_{s,a}(s') = \frac{1}{\sqrt{2\pi|\Sigma|}} \exp\left(-\frac{1}{2}(s' - \mu_s)\Sigma^{-1}(s' - \mu_s)\right)$$

We decided to go even further in the simplification and **only store the average new state**. By doing so, it is assumed that transitions are actually deterministic. This assumption makes this problem easier to implement and apprehend. The main downside is that the uncertainty is not modeled. Take a transition density with a large spread centered around a high reward state a narrow density peaking at a lower reward state. The proposed approach will just overlook the spread and jump straight to the high reward action, when in reality it would certainly be better

to maximize the expected reward and go for the safer choice. Although it has not been observed in this project, an aircraft stall would be a perfect example of this. Note that although the state space is discrete, the average state reached from a given state by a given action is stored as a continuous state during the entire *learning* phase to maximize the accuracy

Implementation: To dynamically learn the transition model, we store for each state s and action a the average state reached $\mu_s(s, a)$ and the number of updates n. Whenever s and a occur and result in a new state s', the update rule is:

$$\mu_s(s, a) = \frac{n\mu_s(s, a) + s'}{n + 1} \text{ and } n = n + 1$$

1. Flying the airplane by hand on the simulator. This method works, although reasonably exploring the state space seems hard to be done in less than several hours.

2. Systematically exploring the state space using the simulator :

Algorithm 1 Systematic exploration

```
for a do
    for s do
        prepare the airplane in the state s
        perform action a (impose elevator deflections)
        update transition matrix
```

This strategy avoids the issues due to incomplete state space exploration and performed the best. A video of such a training can be found at the following link: `https://www.youtube.com/watch?v=WVXFOdSxL3I`.

3. Having an autopilot fly the airplane for a long time. An autopilot has been implemented with nicely tuned traditional control techniques to fly the airplane during the *training phase*. A video of this can be found at the following link : `https://www.youtube.com/watch?v=4R11n3Ob9yY`

4.3 *Controlling* : picking the best action for a given Reward Function

At each time step, the *track scheduler* provides a reward function - e.g. minus the distance to the next way point - to the controller. Value iteration is then run, and this value allows to pick the best action in the action space. Given the hypothesis taken on the probability transitions, the value iteration update rule simplifies to:

$$V_k(s) = R(s) + \gamma \max_a V_{k-1}(s'(s, a))$$

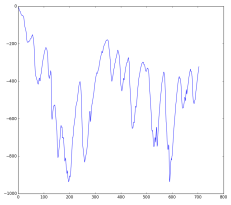
4.4 Note on *learning* and *controlling* frequencies

Learning and *controlling* are both done at given frequencies, respectively f_l and f_c. During *learning* an action is applied at t and the resulting state is recorded at $t + 1/f_l$, while during *training* an action is applied at t in order to reach a given state a $t + 1/f_c$. Two points can be noticed:

- consistency requires f_l and f_c to be in the same bulk part, with $f_l \leq f_c$ as a safety
- due to the discrete state space, there is a tight middle ground when picking f_l : sampling too fast will flag many actions as effect-less (not observe any state transition) while sampling too slowly will flag them all as over powerful (skip many states when taking and action)

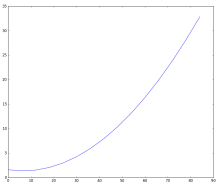
5 Results

A video of the airplane following a predefined trajectory with straight line and highly banked turns can be found at this link `https://www.youtube.com/watch?v=HxBXeiAL3Dg`.

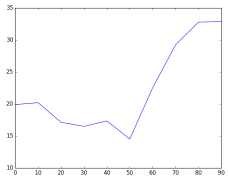


(a) Reward Function at each iteration

(b) Control Effort at each iteration



(c) Elevator Deflection as a function of pitch : theoretical function



(d) Elevator Deflection as a function of pitch : learnt heuristic

On fig.a and b respectively, control effort and reward function are plotted for a 30 seconds trajectory starting with a straight line, followed by several highly banked turns. The effect of action state discretization is noticeable and provides insight for further improvements.

Learning a control strategy : During highly banked turns, a common limitation of traditional controllers is to fail at deflecting the elevator enough, due to the usual separation of longitudinal and lateral control. Engineers often hard code an additional elevator deflection as a function of the pitch angle to minimize the altitude loss due to hi bank angles, and the commonly encoded heuristic is plotted on fig.c. The average elevator deflection on an entire run, plotted versus the roll angle on fig.d, shows that our controller learned by itself this strategy. Even with the simplified model used in this project, it is showed how reinforcement learning can provide accurate responses for cases where human insight is deeply challenged.

6 Next Steps

The three improvements could be considered in a short future : implementing a real Gaussian fit for the transition probabilities, implementing fitted value iteration to get rid of the discrete state space and include control effort in the reward to prevent sharp deflections and smooth the trajectories. In a longer term, we could try to improve the learning capabilities and include safe online learning and state exploration strategies.

References

[1] X-plane 10 simulator by laminar research, www.x-plane.com, .

[2] Xplane connect, https://github.com/nasa/xplaneconnect, .

[3] Pieter Abbeel. *Apprenticeship Learning and Reinforcement Learning with Application to Robotic Control*. PhD thesis, Stanford University, 2008.

[4] Andrew Y. Ng. *Shaping and Policy Search in Reinforcement Learning*. PhD thesis, University of California, Berkeley, 2003.