APPLICATION OF REINFORCEMENT LEARNING TO AUTONOMOUS AIRCRAFT CONTROL IN PARTIALLY OBSERVABLE ENVIRONMENTS

by

Dmitry Shcherbakov

Master of Arts (University of New Brunswick, 2017)

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Computer Science

Supervisor: Michael W. Fleming, PhD, Computer Science

Examining Board: Scott Bateman, PhD, Computer Science, Chair
Huajie Zhang, PhD, Computer Science
Yves Losier, PhD, Mechanical Engineering

This thesis is accepted by the Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

December, 2018

© Dmitry Shcherbakov, 2019


ABSTRACT

This thesis provides a brief survey of the mathematical background of the reinforcement learning (RL) method and sketches the current state of arguably the most developed area of RL application, to the problem of controlling autonomous vehicles (self-driven car-like vehicles). This is then compared to RL solutions in autonomous piloting tasks.

Contrasting the two shows that the latter may benefit from a common framework for RL applications. We propose a framework for autonomous piloting tasks, provide a detailed description of the toolkit available for the framework, and perform an experiment with described instruments. The experiment is designed to determine whether a small fixed memory window can mitigate the adverse influence of such unobserved factors as wind bursts and turbulence. Tests show that the memory mechanism that encapsulates control feedback is an informative input for the learning agent, as long as the unobserved factors affect control behavior significantly.
Table of contents

ABSTRACT ........................................................................................................................ ii
Table of contents ................................................................................................................ iii
List of Tables ...................................................................................................................... v
List of Figures .................................................................................................................... vi
List of Abbreviations ........................................................................................................ vii
1 Introduction ...................................................................................................................... 1
   1.1 Thesis objectives ....................................................................................................... 2
   1.2 Thesis organization ................................................................................................. 3
2 Theoretical Background ................................................................................................... 4
   2.1 Formal presentation of reinforcement learning ......................................................... 6
   2.2 Tabular approach ...................................................................................................... 8
      2.2.1 Monte Carlo ..................................................................................................... 10
      2.2.2 Temporal-difference methods .......................................................................... 11
   2.3 Approximation approach ........................................................................................ 13
      2.3.1 Value function approximation ......................................................................... 13
      2.3.2 Policy approximation ....................................................................................... 15
3 Application of reinforcement learning to aircraft piloting ............................................. 17
   3.1 Application of RL to the control of car-like vehicles ................................................. 18
   3.2 Application of RL to aircraft control problems ......................................................... 22
   3.3 Challenges of the piloting problem ......................................................................... 25
4 Autonomous aircraft control framework ....................................................................... 30
4.1 Autonomous piloting framework ................................................................. 30
4.2 Control cycle ............................................................................................... 31
4.3 Experimental toolkit .................................................................................... 34
  4.3.1 Simulated environment ......................................................................... 35
  4.3.2 Machine learning framework ............................................................... 37
  4.3.3 Benchmark platform ............................................................................ 38
5 Experiment .................................................................................................... 40
  5.1 Experiment setting ................................................................................... 40
    5.1.1 Environment settings ......................................................................... 40
    5.1.2 Performance measurement ............................................................... 44
    5.1.3 PPO agent configuration .................................................................. 45
    5.1.4 Cointegration ..................................................................................... 47
  5.2 Results ...................................................................................................... 48
6 Discussion ..................................................................................................... 58
  6.1 Future work ............................................................................................. 58
  6.2 Conclusion ................................................................................................ 58
    6.2.1 Thesis contributions ......................................................................... 59
Bibliography ..................................................................................................... 61
Curriculum Vitae
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 5.1: Test 1. Descriptive statistics for distance deviation data</td>
<td>53</td>
</tr>
<tr>
<td>Table 5.2: Test 1. Inferential statistics for distance deviation data</td>
<td>53</td>
</tr>
<tr>
<td>Table 5.3: Test 1. Descriptive statistics for heading deviation data</td>
<td>54</td>
</tr>
<tr>
<td>Table 5.4: Test 1. Inferential statistics for heading deviation data</td>
<td>54</td>
</tr>
<tr>
<td>Table 5.5: Test 2. Descriptive statistics for distance deviation data</td>
<td>56</td>
</tr>
<tr>
<td>Table 5.6: Test 2. Inferential statistics for distance deviation data</td>
<td>56</td>
</tr>
<tr>
<td>Table 5.7: Test 2. Descriptive statistics for heading deviation data</td>
<td>56</td>
</tr>
<tr>
<td>Table 5.8: Test 2. Inferential statistics for heading deviation data</td>
<td>57</td>
</tr>
<tr>
<td>Table 5.9: Test 3. Descriptive statistics for distance deviation data</td>
<td>58</td>
</tr>
<tr>
<td>Table 5.10: Test 3. Inferential statistics for distance deviation data</td>
<td>58</td>
</tr>
<tr>
<td>Table 5.11: Test 3. Descriptive statistics for heading deviation data</td>
<td>59</td>
</tr>
<tr>
<td>Table 5.12: Test 3. Inferential statistics for heading deviation data</td>
<td>59</td>
</tr>
</tbody>
</table>
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 4.1: Decision-making system for autonomous piloting</td>
<td>32</td>
</tr>
<tr>
<td>Figure 4.2: Control cycle</td>
<td>34</td>
</tr>
<tr>
<td>Figure 4.3: The types of aircraft stability</td>
<td>35</td>
</tr>
<tr>
<td>Figure 5.1: Hilo International Airport approach chart</td>
<td>43</td>
</tr>
<tr>
<td>Figure 5.2: ANN structure of the PPO agent</td>
<td>48</td>
</tr>
<tr>
<td>Figure 5.3: Test 1. Distance deviation</td>
<td>53</td>
</tr>
<tr>
<td>Figure 5.4: Test 1. Heading deviation</td>
<td>54</td>
</tr>
<tr>
<td>Figure 5.5: Test 2. Distance deviation</td>
<td>55</td>
</tr>
<tr>
<td>Figure 5.6: Test 2. Heading deviation</td>
<td>56</td>
</tr>
<tr>
<td>Figure 5.7: Test 3. Distance deviation</td>
<td>58</td>
</tr>
<tr>
<td>Figure 5.8: Test 3. Heading deviation</td>
<td>59</td>
</tr>
</tbody>
</table>
## List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADS-B</td>
<td>Automatic Dependent Surveillance-Broadcast</td>
</tr>
<tr>
<td>AFCS</td>
<td>Automatic Flight Control System</td>
</tr>
<tr>
<td>ANN</td>
<td>Artificial Neural Network</td>
</tr>
<tr>
<td>CFIT</td>
<td>Controlled Flights Into Terrain</td>
</tr>
<tr>
<td>CNN</td>
<td>Convolutional Neural Network</td>
</tr>
<tr>
<td>DL</td>
<td>Deep Learning</td>
</tr>
<tr>
<td>FDM</td>
<td>Flight Dynamics Model</td>
</tr>
<tr>
<td>GPI</td>
<td>Generalized Policy Iteration</td>
</tr>
<tr>
<td>IAS</td>
<td>Intelligent Autopilot System</td>
</tr>
<tr>
<td>IAS</td>
<td>Indicated Air Speed</td>
</tr>
<tr>
<td>IFR</td>
<td>Instrumental Flight Rules</td>
</tr>
<tr>
<td>LSTD</td>
<td>Least-Squares Time Difference</td>
</tr>
<tr>
<td>LSTM</td>
<td>Long-Short Term Memory</td>
</tr>
<tr>
<td>MC</td>
<td>Monte Carlo</td>
</tr>
<tr>
<td>MDP</td>
<td>Markov Decision Process</td>
</tr>
<tr>
<td>ML</td>
<td>Machine Learning</td>
</tr>
<tr>
<td>MSE</td>
<td>Mean Squared Error</td>
</tr>
<tr>
<td>PID</td>
<td>Proportional Integral Derivative</td>
</tr>
<tr>
<td>RL</td>
<td>Reinforcement Learning</td>
</tr>
<tr>
<td>POMDP</td>
<td>Partially Observable Markov Decision Process</td>
</tr>
<tr>
<td>PPO</td>
<td>Proximal Policy Optimization</td>
</tr>
<tr>
<td>RNN</td>
<td>Recurrent Neural Network</td>
</tr>
<tr>
<td>TD</td>
<td>Time Difference</td>
</tr>
<tr>
<td>TPU</td>
<td>Tensor Processing Units</td>
</tr>
<tr>
<td>UAV</td>
<td>Unmanned Aerial Vehicles</td>
</tr>
</tbody>
</table>
1 Introduction

The field of autonomous aircraft control is receiving considerable attention, both in the aircraft industry and in academic research [1]. These types of vehicles are capable of many different applications in fields such as law enforcement, environmental monitoring, disaster relief, and recovery operations [2]. One issue is that most current research relies on the accuracy of the model describing the prior knowledge of the environment. It is, however, very difficult to attain high accuracy in most realistic implementations, since the knowledge and data regarding the environment are normally limited or unavailable. Using reinforcement learning (RL) is a good approach to overcome this issue because it allows an aircraft to learn and navigate through the changing environment without an explicit model of the environment. The autonomous control of aircraft via reinforcement learning is the focus of this thesis.

Various RL algorithms have already been extensively researched in autonomous vehicle applications, as in many other fields of autonomous control [3]. Reinforcement learning methods enable a vehicle to autonomously discover an optimal behavior through trial-and-error interactions with the environment. This is based on the common-sense idea that if an action results in a satisfactory or better situation, then the tendency to produce that action in the starting situation is reinforced. Instead of explicitly hard-coding the behavior for the task, the designer of the control task solely provides the feedback regarding a scalar objective function that measures the one-step performance of the aircraft. Reinforcement learning is closely related to the theory of classical optimal control. Both theories address the problem of finding an optimal policy (i.e., optimal controller or optimal control policy) that optimizes an objective function. Further, both are based on
the same mathematical framework called the Markov Decision Process (MDP), which uses the notation of states for possible environment configurations that the learning agent may encounter and actions for possible decisions the agent makes in any given state. The autonomous piloting task, however, has requirements that cannot be met by many RL algorithms. Piloting involves reacting to forces that noticeably affect control behavior, but remain unrecognizable. Wind bursts and turbulence are rarely visually perceptible, but can make aircraft practically impossible to control and cause structural damage [4]. Such an environmental setting renders classical MDP and entire classes of RL algorithms not applicable.

1.1 Thesis objectives

1. The first objective of this thesis is to apply an efficient RL algorithm to the autonomous piloting problem and to conduct an experiment on the efficiency of a solution that aids to mitigate the adverse effects of unobserved factors in the environment. The specific solution that is proposed is a fixed memory window with 10 snapshots of past observations and actions from the past 10 seconds. The last 10 seconds serves as a representation of turbulence’s influence on the controllability of the aircraft that can be sufficiently representative, as the effect that turbulence imposes on an aircraft’s attitude is often momentary. At the same time, a memory window with only 10 snapshots is much less computationally intensive, compared to memory mechanisms with an arbitrarily long time lag that are more popular in RL.

2. To our knowledge there is no software framework available for ready-to-use application to the autonomous piloting task. Correspondingly, our second
objective involves the contribution of a framework that interlinks the simulation of an aircraft with the use of RL algorithms. Also, we suggest and provide justification for a specific toolset for implementation of the framework. The overall implementation should be modular, accessible and convenient in its use. In that way, we contribute more than a single solution for a specific control task, but open the possibility for benchmarking various solutions using a common platform.

1.2 Thesis organization

The remainder of the thesis is organized as follows. In Chapter 2, we provide a brief overview of the mathematical framework that serves as a foundation for the RL paradigm, and we review terminology and concepts of the theoretical material that is necessary for comprehension of the further chapters. Chapter 3 is dedicated to a short survey of the autonomous vehicle control problem, and we contrast developments in the area of controlling self-driving car-like vehicles with the task of autonomous piloting, to adopt achievements of the former and highlight challenges of the latter. From the contrast emerges a need for a conceptual framework for the autonomous piloting problem and a common implementation toolkit and benchmarking platform, which is discussed in Chapter 4. Chapter 5 suggests an autonomous RL pilot implementation with a memory mechanism that is designed to mitigate the adverse effect caused by factors imperceptible by learning agent sensors. Chapter 6 provides conclusions and discussion of possible further work in the area.
2 Theoretical Background

This chapter contains the theoretical basis used for building reinforcement learning agents applied in [5] and in the agent proposed in this thesis.

Reinforcement learning can be considered a paradigm in machine learning (ML) separate from supervised learning and unsupervised learning. Unlike supervised learning, where an agent learns from a predefined training set of labeled examples provided by a knowledgeable external supervisor, RL does not assume the presence of a guiding supervisor or prior knowledge of the environment. RL is conceptually closer to an unsupervised learning method, which is typically about finding implicit structure in collections of unlabeled data. Uncovering structure in an agent’s experience can certainly be useful in RL, but by itself does not address the core RL problem of maximizing a reward [6].

Learning with a reinforcement is about training via interactions with the environment. A learning agent trains itself on what to do in any given situation so as to maximize a numerical reward provided as feedback from the environment for the chosen action. Such a formulation puts focus on a characteristic challenge that RL has and which distinguishes it from other aforementioned ML paradigms – the exploration vs exploitation tradeoff. A RL agent seeks for maximum reward performing exploration of the environment and, doing so, it constantly faces the dilemma of choosing between the available move that appears to be the best and potentially even better unexplored ones. Another key feature of RL is that it explicitly considers the problem in its entirety of a goal-directed agent interacting with an uncertain environment without having an option to focus on one subproblem at a time, ignoring how they might fit into the big picture.
The concept can be formalized using ideas from dynamical systems theory, specifically, as the optimal control of Markov Decision Processes [7]. The next subchapter describes the MDP framework, but before that, the underlying concept needs to be presented in a formal way and its main elements need naming in order to establish a terminological apparatus. A reinforcement learning system is called an agent. As was mentioned before, an agent trains via interactions with the environment. The learning process is comprised of a number of discrete acts: at any given time \( t \), the agent makes an observation of the current state \( s_t \) of the environment and chooses an action \( a_t \); after the action is taken, the time is incremented (to \( t + 1 \)), and the environment reacts by sending a reward signal \( r_{t+1} \) to the agent. The process of learning may continue one act after another indefinitely or may be finite and last from an initial time \( t_1 \) until a terminal time \( T \); such a time period is called an episode, and a series of episodes is called an epoch. The agent decides what action to make based on its ultimate goal of maximization of the cumulative reward for a predefined time period, usually an episode. We restrict the concept to discrete time for simplicity; however, it can be extended to the continuous time-case.

The core essential elements of an agent are a policy and a value function. Roughly speaking, a policy is the way that the agent makes its decisions; it is a mapping from observed states and rewards to actions to be taken when in those states. Whereas the reward signal indicates what is good in an immediate sense, a value function specifies what is good in the long run. Both reward \( r_t \) and a state value \( v_t \) can be attributed to a single act, not a time period. However, the value represents a cumulative reward; it is an expectation of all rewards an agent can receive accumulated over the future, starting from the current state.
With the core elements of the RL paradigm identified, a more in-depth formalization of the problem can be presented.

2.1 Formal presentation of reinforcement learning

This chapter introduces the formal specification of a finite Markov Decision Process. Although general MDPs may have infinite (even uncountable) state and action spaces, the introduction will be limited to finite-state and finite-action problems for simplicity. MDPs are a classical formalization of sequential decision making, where actions define not only immediate rewards, but also subsequent situations, or states, and through those, future rewards. In order to proceed, previously described agent-environment intercommunication has to be clarified: in general, a stochastic nature of the environment is assumed; no outcome of the environment is perfectly defined by any combination of previous events. For example, when an agent picks a previously explored action, it does not know which state it is going to face next; instead, the agent builds an expectation over a set of probable states. Correspondingly, random variables will be used heavily, and capital literals will be used to indicate that a variable is random.

The MDP framework [8] is comprised of states, actions, transitions between states and a reward function definition. States are a finite set representing the conditions that can take place in the environment: \( S_t \in S \). At any given time, the agent selects an action available in the current state, \( A_t \in A(s_t) \). After the action is taken, the system transfers to the next state \( s_{t+1} \) and provides the agent with a numerical reward \( R_{t+1} \in R \subseteq \mathbb{R} \).

Since the environment is stochastic, for any state and action available in it, there is a probability distribution for the reward to be received and for the next state that the environment transfers to:
The function \( p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \rightarrow [0,1] \) is an ordinary deterministic function of four arguments.

Even though reward is essential for the framework, it cannot be a sole guide for the agent in picking actions as the agent is looking to maximize all future rewards combined. To build a more useful guide, we have to introduce two more definitions first: a function that will represent a sum of sequential returns starting from \( t \) and up to \( T \) (the return function), and the concept of discounting. Since at the moment of decision making at time \( t \), reward \( r_t \) is already received, it will not be present in the sequence; therefore, the expected return is

\[
G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T
\]

The discount rate determines the present value of future rewards; it is normally assumed that the same value is preferred at the present moment over receiving it in future. For example, some amount of money received at present can be invested and used to generate progressive interest, so the same amount of money is valued less in the future. Thus, the agent tries to select actions so that the sum of the discounted rewards it receives over the future is maximized; it chooses \( A_t \) to maximize the expected discounted return:

\[
G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T = \sum_{k=0}^{T-t-1} \gamma^k R_{t+1+k}
\]

where \( \gamma \in [0,1] \) is a discount factor. As it was already mentioned, a policy is a way the agent picks an action from the available options in a given state; in a stochastic paradigm,

\[1\] The symbol \( \doteqdot \) shows that the expression is in fact a definition.
it is a mapping from states to probabilities of selecting each possible action. If the agent is following policy $\pi$ at time $t$, then:

$$\pi(a|s) \doteq \Pr(A_t = a \mid S_t = s)$$

There are two ways to carry out exploration with regards to the policy: in on-policy methods, exploration is guided by the policy that is the subject of optimization; in the case of off-policy methods, actions are picked by a different policy during exploration.

The policy is fully dedicated to maximization of the return; however, there are conceptually different approaches to their design. Subsequent sections introduce two approaches to building a policy.

### 2.2 Tabular approach

We start with the introduction of an indicator that shows “how good” any given state is, in terms of expected return. Of course, the rewards the agent can expect to receive in the future depend on what actions it will take; that is, it depends on the policy.

The value $v_\pi(s)$ of a state $s$ under a policy $\pi$ is the expected return when starting in $s$ and following $\pi$ thereafter; that is, the state-value function for policy $\pi$ is:

$$v_\pi(s) \doteq \mathbb{E}_\pi(G_t \mid S_t = s) = \mathbb{E}_\pi \left( \sum_{k=0}^{T-t-1} \gamma^k R_{t+1+k} \mid S_{t+k} = s \right)$$

where $\mathbb{E}_\pi(\cdot)$ is the expected value under the condition of following the $\pi$ policy. A value of any action for a specific state can be defined in a similar fashion. The value of taking action $a$ in state $s$ under a policy $\pi$ is the expected return starting from $s$, taking the action $a$, and thereafter following policy $\pi$. That is, the action-value function for policy $\pi$ can be formulated in the following way:
\[ q_\pi(s, a) = \mathbb{E}_\pi(G_t \mid S_t = s, A_t = a) = \mathbb{E}_\pi\left(\sum_{k=0}^{T-t-1} \gamma^k R_{t+1+k} \mid S_{t+k} = s, A_{t+k} = a\right) \]

The value functions can be estimated from a learning process.

The Bellman equation below shows a fundamental property of value functions used by virtually every RL algorithm; it shows how a value of a state can be recursively defined only with probable states one act ahead:

\[ v_\pi(s) = \mathbb{E}_\pi(G_t \mid S_t = s) = \mathbb{E}_\pi(R_{t+1} + \gamma G_{t+1} \mid S_t = s) \]

= \sum_a \pi(a_t \mid s_t) \sum_{s_{t+1}, r_{t+1}} p(s_{t+1}, r_{t+1} \mid s_t, a_t) [r + \gamma \mathbb{E}_\pi(G_{t+1} \mid S_{t+1} = s_{t+1})]

= \sum_a \pi(a_t \mid s_t) \sum_{(s_{t+1}, r_{t+1})} p(s_{t+1}, r_{t+1} \mid s_t, a_t) [r_{t+1} + \gamma v_\pi(s_{t+1})]

The value function \( v_\pi \) is the unique solution to its Bellman equation.

Now that the value of a state is available for calculation, all that a learner needs to do at every act to maximize its expectation of cumulative reward is to pick those actions that provide maximum return, that is, choose a state with largest value. The optimal policy is:

\[ \pi_*(s_t) = \arg\max_{a_t} \mathbb{E}_\pi(G_t \mid S_t = s_t, A_t = a_t) \]

Correspondingly, the optimal value function is the one that follows the optimal policy:

\[ v_*(s_t) = \max_{a_t \in A_t} \sum_{(s_{t+1}, r_{t+1})} p(s_{t+1}, r_{t+1} \mid s_t, a_t) [r_{t+1} + \gamma v_\pi(s_{t+1})] \]

The general method of estimating \( \pi_* \) and \( v_* \) is called Generalized Policy Iteration (GPI) [9]. The GPI is a repeating sequence of two interchanging processes: one making the value function consistent with the current policy (policy evaluation), and the other
making the policy greedy with respect to the current value function (policy
improvement).

\[ \pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \ldots \xrightarrow{I} \pi_\ast \xrightarrow{E} v_\ast \]

It is easy to see that if both the evaluation process and the improvement process stabilize,
that is, no longer produce changes, then the value function and policy must be optimal.
The value function stabilizes only when it is consistent with the current policy, and the
policy stabilizes only when it is greedy with respect to the current value function.
With optimality conditions at hand, we can proceed to reviewing a few model-free
methods. Methods based on the assumption of perfect knowledge of environment
dynamics are of limited use.

2.2.1 Monte Carlo

Monte Carlo (MC) methods are ways of solving the reinforcement learning problem
based on averaging sample returns. In the learning process, MC calculates a state value as
the average returns received after visits to that state. The more returns are observed, the
closer the averages converge to the expected values.
In the absence of a perfect environment model, it is necessary to estimate action-state
values rather than state values. The policy evaluation problem for action values is to
estimate \( q_\pi(s, a) \), the expected return when starting in state \( s \), taking action \( a \), and
thereafter following policy \( \pi \). Correspondingly, in the GPI process, policy improvement
is done by making the policy greedy with respect to the current action-value function,
with the approximate action-value function approaching the true function asymptotically.

\[ \pi_0 \xrightarrow{E} q_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} q_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \ldots \xrightarrow{I} \pi_\ast \xrightarrow{E} q_\ast \]
Since we have an action-value function, no environment model is needed to construct the greedy policy. It is natural for MC to do one policy evaluation and improvement per episode. Generally, the MC algorithm can be presented as follows:

Initialize, for all \( s \in S, a \in A(s) \):
\[
Q(s, a) \leftarrow \text{arbitrary}
\]
\[
\pi(s) \leftarrow \text{arbitrary}
\]
Repeat until sufficient convergence is reached:
\[
\text{Choose } s_0 \in S \text{ and } a_0 \in A(s_0) \text{ s.t. all pairs have probability } > 0
\]
Generate an episode starting from \( s_0, a_0 \), following \( \pi \)
For each pair \( s, a \) appearing in the episode:
\[
G \leftarrow \text{the return that follows the first occurrence of } s, a
\]
Append \( G \) to \( \text{Returns}(s, a) \)
\[
q(s, a) \leftarrow \text{average(} \text{Returns}(s, a) \text{)}
\]
For each \( s \) in the episode:
\[
\pi(s) \leftarrow \text{argmax}_a q(s, a)
\]

In order for convergence to be guaranteed in infinite episodes, every action of every state must have a non-zero probability of being taken; there are both on- and off-policy solutions to this problem.

### 2.2.2 Temporal-difference methods

Like MC methods, time-difference (TD) [10] methods can learn directly from experience without having a model of the environment's dynamics, but unlike MC, TD methods update estimates based in part on learned estimates of the successors, without waiting for the terminal act of the episode; such a method is called bootstrapping. Roughly speaking, MC methods update estimated value functions upon completion of an episode, whereas TD methods need to wait only until the next act. At every time step, an estimate of the preceding state’s value is updated with the observed reward and the present estimate of the current state’s value.
\( v(s_{t-1}) \leftarrow v(s_{t-1}) + \alpha \left[ r_t + \gamma v(s_t) - v(s_{t-1}) \right] \)

where \( \alpha \in [0,1] \) is a step-size parameter. In order to avoid unwanted bias in estimation, batch updating is used. Given an approximate value function, \( \hat{v} \), the increments are computed for every time step \( t \) at which a nonterminal state is visited, but the value function is changed only once, by the sum of all the increments. Then all the available experience is processed again with the new value function to produce a new overall increment, and so on, until the value function converges.

With batch updating, TD converges deterministically to a single answer independent of the step-size parameter, \( \alpha \), as long as \( \alpha \) is chosen to be sufficiently small. As is the case for MC, convergence to an optimal policy is guaranteed only if every action-state pair has a chance to be visited, and as with MC, this condition can be satisfied via on-policy and off-policy controls.

In off-policy TD control, called Q-learning [11], the action-value function \( q(s, a) \) directly approximates \( q^* \), independently of the followed policy.

\[
q(s_t, a_t) \leftarrow q(s_t, a_t) + \alpha \left[ r_{t+1} + \gamma \max_{a_{t+1}} q(s_{t+1}, a_{t+1}) - q(s_t, a_t) \right]
\]

\( \hat{q} \) has been shown to converge to \( q^* \) under the usual stochastic approximation conditions.

Initialize \( q(s, a) \), for all \( s \in S \), \( a \in A(s) \), arbitrarily, and \( q(s_T; a_T) = 0 \)

Repeat (for each episode):

\[
\text{Initialize } s_t
\]

Repeat (for each act of episode):

Choose \( a_t \) from \( s_t \) using policy derived from \( q \) (e.g., \( \varepsilon \)-greedy)

Take action \( a_t \), observe \( r_{t+1}, s_{t+1} \)

\[
q(s_t, a_t) \leftarrow q(s_t, a_t) + \alpha \left[ r_{t+1} + \gamma \max_{a_{t+1}} q(s_{t+1}, a_{t+1}) - q(s_t, a_t) \right]
\]

\( s_t \leftarrow s_{t+1} \)

until \( t == T \)
2.3 Approximation approach

A tabular approach may not be applicable in a large set of tasks due to its intrinsic limitation – there is a need to store and traverse all explored and estimated action-state values. State spaces may be combinatorial and too large to be handled by any reasonable resources. On top of complexities with data processing, in large spaces there is a high risk of encountering a state that has not been visited during the learning process, which leaves an agent with no knowledge about how good available actions are.

One way to overcome that limitation is, instead of storing values, to draw generalized rules about actions from visiting different states discovering patterns. Function approximation techniques are widely applied in RL for that purpose [12].

2.3.1 Value function approximation

Instead of storing all explored and estimated state values, the entire value function will be approximated with a parameterized functional form, where a weight vector \( \mathbf{w} \) will be used to indicate the estimated parameters:

\[
v(s, \mathbf{w}) = v_\pi(s)
\]

An accuracy measurement has to be introduced in order to proceed with approximation techniques. An error function will indicate how precise an approximation is in its predictions and be the objective function for minimization in estimation techniques. Mean Squared Error (MSE) is one of the most popular error functions in regression because of its simplicity and convenient statistical properties:

\[
MSE(\mathbf{w}) = [v_\pi(s) - v(s, \mathbf{w})]^2
\]
An ideal goal in terms of MSE would be to find a global optimum, a weight vector $\mathbf{w}^*$ for which $\text{MSE}(\mathbf{w}^*) \leq \text{MSE}(\mathbf{w})$ for all possible $\mathbf{w}$. Reaching this goal is relatively simple for linear regression [13].

In linear methods the approximate function is linear in weights:

$$v(s, \mathbf{w}) = \mathbf{w}^\top \mathbf{x}(s) = \sum_{i=1}^{d} w_i x_i(s)$$

where $\mathbf{x}$ is a vector of variables.

It is natural to use stochastic gradient descent (SGD) [14] updates with linear function approximation. SGD methods try to minimize the error by adjusting the weight vector after each data sample by a small amount in the direction that would most reduce the error on that sample, using the gradient of the error:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{1}{2} \alpha \nabla \left[ v_\pi(s_t) - v(s_t, \mathbf{w}_t) \right]^2$$

$$= \mathbf{w}_t - \alpha \left[ v_\pi(s_t) - v(s_t, \mathbf{w}_t) \right] \nabla v(s_t, \mathbf{w}_t)$$

In the linear case the general SGD update reduces to a particularly simple form:

$$\mathbf{w}_t + \alpha \left[ v_\pi(s_t) - v(s_t, \mathbf{w}_t) \right] \mathbf{x}(s_t)$$

Thus, the per act update is:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha (r_{t+1} + \gamma \mathbf{w}_t^\top \mathbf{x}_{t+1} - \mathbf{w}_t^\top \mathbf{x}_t) \mathbf{x}_t$$

Once the system has reached a steady state, for any given $\mathbf{w}_t$, the expected next weight vector can be written:

$$\mathbb{E}[\mathbf{w}_{t+1} | \mathbf{w}_t] = \mathbf{w}_t + \alpha (\mathbf{b} - \mathbf{A} \mathbf{w}_t)$$

where $\mathbf{b} = \mathbb{E}[r_{t+1}\mathbf{x}_t]$ and $\mathbf{A} = \mathbb{E}[\mathbf{x}_t (\mathbf{x}_t - \gamma \mathbf{x}_{t+1})^\top]$.

However, computation per act proportional to the number of parameters is not necessary in the case of linear regression, as a TD fixed point can be reached using matrix algebra.
In the Least-Squares TD (LSTD) algorithm [15], \( A \) and \( b \) estimates are computed to find fixed point estimated weights:

\[
\mathbf{w}_{TD} = A^{-1}b
\]

LSTD is arguably the best that can be done in the case of linear approximation [6].

**2.3.2 Policy approximation**

Even if a value function is approximated and an arbitrarily large state space can be processed, as long as a policy is built on choosing the best action from the set available for a given state, the agent is vulnerable to the same limitations in relation to action spaces as tabular methods are in relation to state spaces. For example, if an action space is continuous, greedy search through all available actions is not practically feasible. In order to overcome any space limitations, the policy itself can be parametrized and estimated.

\[
\pi(a | s, \theta) = \text{Pr}(A_t = a | S_t = s, \theta_t = \theta)
\]

where \( \theta \in \mathbb{R}^d \) is the policy’s parameter vector. A value function may still be used to learn the policy parameter, but is not required for action selection.

In general, estimation is based on the gradient of some performance measure \( J(\theta) \) with respect to the policy parameters. The goal is to maximize performance; correspondingly, parameter updates approximate gradient ascent in \( J \):

\[
\theta_{t+1} = \theta_t + \alpha \nabla J(\theta_t)
\]

where \( \nabla J(\theta_t) \) is a stochastic estimate whose expectation approximates the gradient of the performance measure with respect to its argument \( \theta_t \). One way to define the performance is

\[
J(\theta) = v_{\pi_\theta}(s_0)
\]
in which case gradient ascent is possible, as it has been shown [16; 17] that:

$$\nabla J(\theta) \propto \sum_s \mu(s) \sum_a q_\pi(s, a) \nabla_\theta \pi(a | s, \theta)$$

Artificial neural networks (ANN) [18] are widely used for nonlinear function approximation. An ANN is a network of interconnected nodes that mimic the properties of neurons, the main component of nervous systems. The nodes are typically semi-linear units; that is, they compute a weighted sum of their input signals and then apply to the result a nonlinear function, called the activation function, to produce the node's output. The functions are parameterized by the network's connection weights. Approximation is achieved by iteratively adjusting the parameters via backpropagation.
3 Application of reinforcement learning to aircraft piloting

One major application scenario of RL is the embedded computing environment, such as in unmanned aerial vehicles and autonomous driving. The information these applications utilize to make their decisions normally consists of data provided by multiple sensors. These types of sensors often generate data in large quantities. A popular approach to solving problems with high dimensional data as input is ML. One of the drawbacks of some approaches in ML, such as supervised learning, is that they require large amounts of data. Because of this, the use of synthetic data for training ML algorithms has been an active research field. For the case of autonomous vehicle control, acquiring sufficient training data is difficult since the dataset will very likely be biased towards containing samples from close to ideal control conditions. A remedy to this problem is to train the models on synthetic data since non-ideal control scenarios can be simulated within a synthetic environment safely. Training on synthetic data is particularly useful for RL since these algorithms require the agent to interact with the environment in an iterative learning-by-doing scheme, thus incapacitating the process of training data generation. Since the RL-models learn by exploring the environment and different actions, training using a real vehicle to conduct the exploration is currently highly improbable, as accidents will likely occur in this setting.

In this chapter, we provide a brief survey of RL applications to vehicle control, specifically to driving car-like vehicles (as arguably the most developed direction in the area of autonomous vehicle control) and piloting aircrafts. Then we compare the two, in order to highlight challenges in the automated piloting task and possible directions for future developments in the area.
3.1. Application of RL to the control of car-like vehicles

Driving a vehicle requires a high level of skill, experience and constant focus from a human driver. Although computers might perform much better in terms of sustained attention and focus than humans, the current state of AI system development has not yet achieved complete autonomy in driving. However, automated (assisted) driving systems are advancing at a rapid pace. One may say that the area of automated car-driving systems is the most developed application of ML in vehicle control, and recent achievements in the area are getting very close to the state of actual autonomous control [19].

Research efforts, both in academia and in the car industry, towards developing driverless vehicle technology have been steadily increasing in the last four decades, fueled by recent advances in sensing and computing technology. Since as early as the 1980s, numerous projects and competitions have taken place in pursuit of driverless technology. Up to now, many prototypes for autonomous or highly automated cars have been developed and demonstrated [20].

The extent to which a car is automated can vary from fully human-operated to completely autonomous. The US-based Society of Automotive Engineers has introduced and maintains the standard J3016 [21] with a scale from 0 to 5 for grading the level of vehicle automation, which can be summarized as follows:

- Level 0 (No Automation): the full-time performance by the human driver of all aspects of the dynamic driving task, even when enhanced by warning or intervention systems.
• Level 1 (Driver Assistance): the driving mode-specific execution by a driver assistance system of either steering or acceleration/deceleration using information about the driving environment and with the expectation that the human driver perform all remaining aspects of the dynamic driving task.

• Level 2 (Partial Automation): the driving mode-specific execution by one or more driver assistance systems of both steering and acceleration/deceleration using information about the driving environment and with the expectation that the human driver perform all remaining aspects of the dynamic driving task.

• Level 3 (Conditional Automation): the driving mode-specific performance by an automated driving system of all aspects of the dynamic driving task with the expectation that the human driver will respond appropriately to a request to intervene.

• Level 4 (High Automation): the driving mode-specific performance by an automated driving system of all aspects of the dynamic driving task, even if a human driver does not respond appropriately to a request to intervene.

• Level 5 (Full Automation): the full-time performance by an automated driving system of all aspects of the dynamic driving task under all roadway and environmental conditions that can be managed by a human driver.

The tasks involved in creating an autonomous driving agent can be divided into three categories:

• Recognition: Identifying components of the surrounding environment.

• Prediction: Building internal models that predict the future states of the environment.
- Planning: Generating of an efficient model that incorporates recognition and prediction to plan the future sequence of driving actions that will enable the vehicle to navigate successfully.

The partitioning of these levels is, however, rather blurred, with different variations of this scheme occurring in the literature [20].

We will narrow down our focus to aspects of decision-making levels that involve RL implementation for self-driving cars, for systems falling into the automation level of 3 and above.

**Recognition:** Although far from trivial, recognition is now a relatively easy task thanks to advances in Deep Learning (DL) algorithms. DL is a class of machine learning algorithms that use a cascade of multiple layers of nonlinear processing units for feature extraction and transformation, where each successive layer uses the output from the previous layer as input [22]. In this regard, Convolutional Neural Networks (CNNs) are probably the most successful deep learning model, and have formed the basis of every winning entry on the ImageNet challenge since AlexNet in 2012 [23]. DL models are able to learn complex feature representations from raw input data, avoiding the need for handcrafted features. DL algorithms have reached human level recognition and even surpassed it at several object detection and classification problems [24; 25; 26].

**Prediction:** To be able to predict the future, it is important to integrate past information. As such, Recurrent Neural Networks (RNNs) are used as a convenient solution to this problem. A RNN is a class of ANN, which has an architecture that allows it to exhibit temporal dynamic behavior for a time sequence. Layers of RNN are organized in such a way that information from a previous time step input data layer is passed to the
successive layer, so the network can extract useful information not only from input samples but from their order too. Unlike regular ANNs, RNNs can use their internal state (memory) to process sequences of inputs. Long-Short Term Memory (LSTM) networks [27] are one such category of RNN that have been used in end-to-end scene labeling systems [28].

**Planning:** Planning is the hardest task of the three. The difficulty lies in integrating the ability of the model to understand the environment (recognition) and its dynamics (prediction) in a way that enables it to plan the future actions such that it avoids unwanted situations (penalties) and drives safely to its destination (rewards). The RL framework has been used for a long time in control tasks. The mixture of RL with DL was pointed out to be one of the most promising approaches to achieve human-level control [29]. In various works [30] [31] [32] human level or above control performance was demonstrated on Atari games using the different RL models, in which RL is responsible for the planning part while DL is responsible for the representation learning part. A conceptually similar approach was taken to introduce intelligent agents that would be able to control a car-like vehicle in realistic environments. In addition to various proposed schemes of mapping raw sensors’ input to driving actions and different RL models for decision making [33] [34] [35], there is a proposed framework [36] for conducting autonomous driving experiments on a pipeline starting from simulated driving raw video footage and ending with control decisions that comprises an end-to-end training scheme of a deep ANN for autonomous driving. Such a framework can serve as a common platform for conducting experiments and comparing the results in the specific task of autonomous driving.
3.2 Application of RL to aircraft control problems

Human factors, including pilot error, are another potential set of factors, and currently the factor most commonly found in aviation accidents. An Automatic Flight Control System (AFCS/autopilot) is a system used to control the trajectory of an aircraft without constant 'hands-on' control by a human operator being required. Autopilots in their current state of development do not replace human operators, but instead they assist them in controlling the aircraft; it is not a substitute for the pilot, but is an aid to help workload and situational awareness [37]. Even though advancements in autopilot technology help reduce human error in aviation, and the number of controlled flights into terrain (CFIT) has diminished since 2008, in 2017 CFIT was the largest contributing fatal risk factor in scheduled commercial flights on airplanes above 5.7t [38]. A CFIT is an accident in which an airworthy aircraft, under pilot control, is unintentionally flown into the ground, a mountain, a body of water or an obstacle. There are grounds for believing that the replacement of human aircraft operators with automated controlling systems may increase civil aviation safety.

Unmanned aerial vehicles (UAV) represent another approach in the aviation industry that improves safety, has a vast field of efficient applications and can be enhanced with new advances in autonomous piloting research. While UAVs originated mostly in military applications, their use is rapidly expanding to commercial, scientific, recreational, agricultural, and other applications, such as policing, peacekeeping, and surveillance, product deliveries, aerial photography, agriculture, smuggling, and drone racing [2]. Autonomous piloting is different from driving in two major aspects:

- Piloting involves fewer (if any) interactions with other traffic participants.
• Visual input is often of limited use or completely uninformative.

A well-planned flight route does not involve intersections with other routes, and normally mid-air operations do not include collision avoidance maneuvers. Any risks of potential collisions are normally resolved in advance by ground air traffic services [39]. Almost all modern aircraft are fitted with traffic collision avoidance systems, which are designed to try to prevent mid-air collisions. The system, based on the signals from aircraft transponders, alerts pilots if a potential collision with another aircraft is imminent. Despite its limitations, it is believed to have greatly reduced the chance of a mid-air collision [40].

Flights are often carried out in conditions that sufficiently complicate or make impossible establishing visual references, for example during heavy precipitation, in cloud layers at cruising altitude, or on moonless nights. Instrumental flight rules (IFR) are established to govern flight under conditions in which flight by outside visual reference is not safe. IFR flight depends upon flying by reference to instruments in the flight deck, and navigation is accomplished by reference to electronic signals [41]. Use of instrument flight rules is also required when flying in "Class A" airspace regardless of weather conditions. Class A is the most common on commercial flight paths.

Such distinctions of the autonomous piloting task imply that hypothetically there may be fully functional autonomous aircraft control systems that operate air vehicles under IFR without or with limited aid of visual sensor input. Following the terminology used in a previous section, recognition (perception) of the environment can be rendered fully by regular aircraft avionics.
There are suggestions of both kinds of intelligent learning autopilots in the literature: with visual input and with instrument input only.

The authors of [42] created an Intelligent Autopilot System (IAS) that can learn from human pilots by applying the Learning by Imitation concept with ANN. By using this approach, they aim to extend the capabilities of modern autopilots and enable them to autonomously adapt their piloting to suit multiple scenarios ranging from normal to emergency situations. Learning by Imitation [43] is split into two main parts, each with its own objectives: 1. learning a policy or a low-level task which could represent a direct mapping between states and relative actions, and 2. learning a reward function or a high-level task which could represent a specific goal to be achieved. The identical approach was taken [44] to extend the capabilities of the IAS to be able to learn to handle emergency situations. In both works, the method was comprised of a human pilot performing a demonstration in a high-fidelity aircraft simulator “X-Plane”, while flight instrument data and control inputs are collected every 0.1 second and stored in a database. As soon as a sufficient number of data samples are collected, an IAS is trained.

In [45], the authors used Yamaha R-50 remote-controlled helicopter to apply The PEGASUS RL algorithm [46] for learning to perform a sustained autonomous hover. The helicopter carries an Inertial Navigation System consisting of 3 accelerometers and 3 rate gyroscopes installed in exactly orthogonal x,y,z directions, and a differential GPS system, which with the assistance of a ground station, gives position estimates with a resolution of 2cm. An onboard navigation computer runs a Kalman filter which integrates the sensor information from the GPS, INS, and a digital compass, and reports (at 50Hz) 12 numbers corresponding to the estimates of the helicopter’s position, orientation, velocity and
angular velocities, which were used as learning agent inputs. In similar work [47] a remote-controlled helicopter equipped with triaxial accelerometers, rate gyros, and magnetometers was trained to perform aerobatic maneuvers, with no visual input used. A vision-based approach was introduced in [48], where authors use a CNN to interpret the scene by extracting information from a raw image, and mapping it to a robust representation in the form of a waypoint and desired speed. The perception system uses a 300 × 200 RGB image from the onboard camera, which is fed to the CNN, and a two-dimensional vector that encodes the direction to the new goal in normalized image coordinates and a normalized desired speed to approach it. The perception system was trained with imitation learning, using automatically generated globally optimal trajectories as a source of supervision.

Another paper [5] shows an application of three RL algorithms for autonomous aircraft control using a specific collision avoidance technology in a simulated environment. The applied algorithms were Monte Carlo, Least-Squares Temporal Difference and Q-Learning. In addition to regular avionics, the perception system received input from an Automatic Dependent Surveillance-Broadcast system (ADS-B). ADS-B is a surveillance technology that is used to determine an aircraft’s position by means of satellite navigation. This position is then broadcast at regular time intervals to inform other agents of the aircraft’s position [49]. No visual input was used.

3.3 Challenges of the piloting problem

Human pilots are trained to handle flight uncertainties and emergency situations such as severe weather conditions and aircraft malfunctions. In contrast, autopilots operate normally only in a predictable standard environment. Strong turbulence or severe cross
wind, for example, can cause the autopilot to disengage or even attempt an undesired action which could jeopardize flight safety. The limitations of autopilots require constant monitoring of the system and the flight status by the flight crew to react quickly to any undesired situation or emergencies; in no case should an aircraft be controlled by an autopilot unattended. On the other hand, trying to anticipate everything that could go wrong with a flight, and incorporating that into the set of rules or control models “hardcoded” in an autopilot, is practically infeasible. There is research where authors study limitations of current autopilots [50] [51] (such as the inability to handle severe weather conditions) and consider a number of accidents in aviation as result of inappropriate behavior of corresponding autopilot systems.

Classic and modern autopilots rely on controllers such as Proportional Integral Derivative controllers (PID controllers), and Finite-State automata [52]. Many recent research efforts focus on enhancing flight controllers, through the introduction of various approaches [53; 54; 55; 56; 57]. However, manual design and developing of all the necessary controllers to handle the complete spectrum of flight scenarios and uncertainties ranging from normal to emergency situations might not be practically feasible due to the vast number of possible situations.

Although the RL approach allows autonomous discoveries of an optimal behavior through trial-and-error interactions with the environment and represents a natural potential solution to the problem, learning algorithms require augmentation in order to perform optimally in the environments with the presence of uncertainties. The mathematical framework that RL is based upon assumes perfect knowledge of observed states, which is rarely (if ever) the case for any actor in the real world.
Specifically, in aviation, information about such important factors as wind speed and direction, as well as turbulence, is often not available mid-flight. Ground services and other pilots may report the information about certain areas, but that can only cover a part of a commercial airline. Turbulence of some types is predictable by the source of its origin: convective weather patterns, i.e., thunderstorms, produce strong updrafts and downdrafts; air flowing over the top of a mountain produces turbulence in the form of waves when it reaches the other side. Pilots can anticipate turbulence in thunderstorms or in mountainous areas as a natural occurrence. However, so called clear-air turbulence caused when bodies of air moving at widely different speeds meet, is much more difficult to predict; at typical heights where it occurs, the intensity and location cannot be determined precisely. Clear-air turbulence is usually impossible to detect with the naked eye and very difficult to detect with a conventional radar [58].

There are optical instrument solutions that help to detect turbulence imperceptible for a pilot’s vision and for regular aircraft sensors; however, most modern aircrafts are not equipped with such devices and not every aircraft is able to be equipped with them. In order to widen applicability of the topic, we will consider information about wind and turbulence unobservable.

Reinforcement learning for environments in which state information is available incompletely for the agent is handled by the partially observed Markov decision process (POMDP) framework [59]. What distinguishes a POMDP from a fully observable MDP is that the agent perceives an observation $o \in \Omega$, instead of observing the state directly. Formally, the discrete set of observations $\Omega = \{o^1, \ldots, o^M\}$ represent all possible sensor readings the agent can receive. Which observation the agent receives depends on the next
state and may also be conditional on its action, and is drawn according to the probabilistic observation function $O : S \times A \times \Omega \rightarrow [0,1]$.

In order for an agent to choose its actions successfully in partially observable environments, memory is needed. A straightforward implementation of memory would be to simply store the sequence of actions executed and observations received; however, because such a form of memory can grow indefinitely, it is not a practical option. Instead, a history window of a-priori defined length can be maintained to keep lagged actions and observations [60].

Finite history windows cannot, however, capture arbitrary long-term dependencies or dependencies on arbitrary late lag. Various solutions have been introduced to alleviate that problem [61; 62; 63]. One of the approaches was used in previously mentioned research [36] dedicated to autonomous driving, where RNNs based on the Long Short-Term Memory architecture [27] have been used as an internal state representation.

Another difficulty that the autonomous piloting task faces is the need for continuous observation and action spaces.

Presently, the problem of unobserved factors in the environment state has not been addressed directly in the area of RL application to autonomous aircraft control. However, turbulence can cause large abrupt changes to an aircraft attitude or even cause an emergency situation that may lead to a fatal accident [64]. Some RL algorithms perform better in the presence of uncertainties than others; for example, value-function approximation algorithms, unlike policy-gradient methods, are not guaranteed not to diverge or oscillate infinitely in the learning process. Specific modifications to the algorithms may alleviate the adverse effect of the unobserved factors on the learning
performance. In this thesis, we address the problem of mitigating the corrupting influence of turbulence on the learning performance of a RL algorithm to see if a policy-gradient algorithm performance can be improved with a fixed memory window augmentation. Inputs from avionics have real values; although it is possible to map any value into a discrete domain, the coarser the domain the bigger the source of inaccuracy introduced into the algorithm, as any transformation of input value increases the imperfection of the observation. Deterministic discretization causes a strong bias in policy evaluation and policy improvement [65].

A similar issue occurs with the action space. Sometimes a very small adjustment of a control actuator is required for the best result, and so if a discrete action space is not fine enough, any initial action will be an overshoot followed a potentially infinite series of consecutive stabilizing control actions. Jerky controls adversely affect stability of the aircraft and its durability. One pilot performance criterion is the stability and smoothness of the flight [66].
4 Autonomous aircraft control framework

In order for an experiment to be representable and repeatable, it needs to be conducted in well-defined conditions. There is a need for a testbed to compare various ML solutions and to make objective conclusions about their performance. In this chapter, we introduce a proposed theoretical (terminological) framework for simulated experiments on an autonomous aircraft control task, present a suggested implementation, and provide justification for the chosen toolkit. We then describe an experiment run on it to compare the efficiency of different RL algorithms in a specified learning task.

4.1 Autonomous piloting framework

The framework outlines the context of various components of an autonomous piloting software system and sketches their interaction; it will help to give a clearer picture of how the entire system can function and where the application of RL takes place. We start by depicting a decision-making system for autonomous piloting (Figure 4.1) on an abstract level and then we dive into each of its components to describe it in detail.

Even though the system is meant to be autonomous, it has to receive a user-defined assignment to carry out. In the first stage, an operator inputs a mission – an explicit goal to be reached without direct control by a human operator. Typical examples would be: to fly a specific route, to fly from a specific origin to a specific destination, or to carry out surveillance of a specific area. As soon as the mission is defined, the system generates a plan for performing it – a structured list of key objectives to accomplish that comprises the goal. In the case of building a flight plan from one point to another, for example, an airway network can be represented as a directed graph with edge weights corresponding to the cost of traversing any given airway, so the plan building can be formulated as the
problem of finding a minimum-cost path on an airway network graph. A mission can potentially be indefinite; the plan is cyclical in this case. The control cycle is the most difficult stage; it is a process of successive achievement of key objectives via an iterative procedure of assessing the current state of the environment and taking corresponding control actions. An example would be following a flight route. During the control cycle, whenever necessary, the plan may be altered as appropriate. For instance, a flight route may be changed to avoid a storm cell. The control cycle stage is described in more detail in the next section. Both the control cycle and plan-building stages can be configured and assisted by a human operator. Once the goal is accomplished, the mission is considered complete and the system transitions back to the mission input stage.

A schematic description of the control cycle is provided in the next section, and a specific proposed implementation is presented in Section 4.3.

4.2 Control cycle

Essentially, the control process is about mapping the understanding of the current situation to the manipulation of actuators (Figure 4.2) in order to keep as close as possible to the plan execution.

Typical aircraft instruments report the following essential data: airspeed, vertical speed, altitude, GPS coordinates, heading, pitch, roll. Most of the data available have the form of real values and can be used in learning algorithms directly. Data from systems that
report more complex information can be converted into the required form. Alternately, features from raw data can be extracted via manual feature design or an automatic feature extraction approach can be applied, such as the method with CNN [67].

As mentioned in the previous chapter, the RL agent requires a memory to have a more accurate state observation; therefore, observation is comprised of the representation of previously taken actions and the corresponding perceived observations, along with data from sensors available for an aircraft. One of two different memory mechanisms are available for a RL agent: either a fixed length history window or a memory with arbitrary lag length. The former approach is less computationally intensive but can capture only actions and observations before a specific lag. The latter does not have to represent (possibly long) entire histories, but can in principle extract and represent just the relevant information for an arbitrary amount of time. However, learning to do that has proven difficult [68]. The difficulty lies in discovering the correlation between a piece of information and the moment at which this information becomes relevant at a later time, given the distracting observations and actions between them. This difficulty can be viewed as an instance of the general problem of learning long-term dependencies in timeseries data. LSTM [27] is one of the promising implementations of such an approach. Both approaches can be argued as advantageous in the autonomous piloting application; in general, the choice depends on the need for capturing historical events from the arbitrarily distant past.

Even though the view outside of the aircraft is often limited (in clouds, in adverse weather conditions, at nighttime) and not sufficiently informative for the learner, a pilot has to use visual aids at least for separation (collision avoidance), and the IFR requires
visual aids to be a primary source of information. Besides, human vision limitations may be compensated with enhanced flight vision systems and synthetic vision systems that incorporate information from various aircraft-based sensors to provide vision in limited visibility environments. Informative visual data can be used as part of the state observation in control decision-making processes. Such data is not given to the controlling algorithm in its raw form, but rather processed through the CNN for feature extraction and then the feature vector is used as part of the observation. The observation data is used as input for the system to zero in on such an operation with controllers that lead to the finest plan execution possible.

Figure 4.2: Control cycle

The flight controls are the devices and systems that govern the flight path followed by the aircraft. In the case of many conventional fixed-wing airplanes, the primary flight controls utilize hinged, trailing edge surfaces called elevators for pitch, ailerons for roll, and the rudder for yaw (Figure 4.3). These surfaces are operated by the pilot in the flight deck or by an automatic pilot. Aircraft engine controls are also considered as flight controls as they affect thrust - a force that moves an aircraft in the direction of the
motion. There are many more other controls that manipulate the aircraft’s systems depending on its configuration.

Figure 4.3: The types of aircraft stability².

4.3 Experimental toolkit

In this section, we present a toolkit that allows simulated experiments on RL algorithms being used for the implementation of the autonomous piloting system. We provide a brief description of each tool used, along with its advantages over similar solutions and instruments used for integration of individual tools.

4.3.1 Simulated environment

The following requirements for the simulated environment software were considered:

- High accuracy of flight dynamics simulation: the flight dynamics model has to be as realistic as possible so results of the training would be applicable in real life.

- I/O interface: the software has to be equipped with a communication interface so the simulation could be controlled and current parameters of the agent could be monitored.

- Weather settings: there has to be access to control over weather settings so training could be performed under a wide spectrum of meteorological effects.

- Visually-realistic rendering: photo-realistic graphics are preferred for training results to be usable with real-life visual inputs.

There are several flight simulators that meet the criteria: “AeroFly Flight Simulator”\(^3\), “FlightGear”\(^4\), “Microsoft Flight Simulator X”\(^5\), “Prepar3D”\(^6\), and “X-Plane”\(^7\). Even though it is difficult to draw a conclusive comparison between them, considering all available configurations and third-party modifications, “FlightGear” stands out as the

---

\(^3\) http://www.aerofly.com/

\(^4\) http://home.flightgear.org/

\(^5\) https://en.wikipedia.org/wiki/Microsoft_Flight_Simulator_X

\(^6\) https://www.prepar3d.com/

\(^7\) https://www.x-plane.com/
only free, open source multi-platform flight simulator that has been used for academic research, professional training and entertainment [69].

FlightGear supports several different types of flight dynamics models (FDM).

- **LaRCsim** was originally developed by NASA, and was the first FlightGear model, now replaced by JSBSim and YASim.
- **JSBSim** was originally developed specifically for FlightGear, and has been the default FlightGear flight dynamics model since 2000. In addition to airplanes, JSBSim also supports rockets, helicopters and lighter-than-air aircrafts.
- **UIUC** was developed by the UIUC Applied Aerodynamics Group at the University of Illinois, based on LaRCsim, and includes additional features, such as the ability to simulate aircraft icing. UIUC lacks ground interaction.
- **YASim** is yet another FDM using different calculation methods, which supports airplanes and helicopters.
- **SpaceFDM** was developed for simulations in conditions of low to non-existent gravitational force and aerodynamic lift and drag.
- **Other custom FDMs** have been written for specific aircraft types, such as for hot air balloons. FlightGear can also be set up to render using inputs from an external FDM source, such as from MATLAB™.

Currently, FlightGear is the only graphical flight simulator to support all of these flight dynamics models.

FlightGear supports multiple concurrent input-output connections, including Telnet, UDP and HTTP. It allows users to read and modify most of the properties of the flight
dynamics model, including aircraft properties such as flight instrument readings and actuator positions, as well as environment properties such as weather conditions. YASim and JSBSim, being arguably the most advanced flight dynamics models, support virtually all relevant atmospheric phenomena that affect piloting, such as: temperature, dew point, pressure, density, wind (three dimensional), visibility, turbulence, etc.

4.3.2 Machine learning framework

There is no straightforward way to rank the vast number of various machine learning frameworks: their performance is test-dependent, and ease of use is subjective. However, the size of the support community is relatively easy to measure, and TensorFlow is the clear winner in that category. TensorFlow has the most GitHub activity, Google searches, Medium articles, books on Amazon and ArXiv articles [70]. This is relevant because a strong positive correlation was found between open-source project activity and its quality [71].

TensorFlow is perceived to be one of the best open-source software libraries for dataflow programming. It is a machine learning system that operates at a large scale and in heterogeneous environments. TensorFlow uses dataflow graphs to represent computation, shared state, and the operations that mutate that state. It maps the nodes of a dataflow graph across many machines in a cluster, and within a machine across multiple computational devices, including multicore CPUs, general purpose GPUs, and custom-designed ASICs known as Tensor Processing Units (TPUs). This architecture gives flexibility to the application developer. TensorFlow supports a variety of applications, with a focus on training and inference on deep neural networks.
4.3.3 Benchmark platform

Although the previous two components of the toolkit are sufficient for conducting experiments on RL autonomous pilot solutions, the overall experiment framework can benefit from the introduction of a benchmark platform that serves as a common interface for a simulated environment and a RL algorithm that operates in it. Python-based OpenAI Gym is known as a substantial tool that provides a standard baseline to compare and benchmark RL algorithms. Such a tool has a great use in standardization and reproducibility of deep reinforcement learning projects, the importance of which is essential for maintenance of rapid progress in RL [72].

The authors of OpenAI Gym aim to combine the best elements of previously developed benchmark collections, in a software package that is maximally convenient and accessible. It focuses on the episodic setting of RL and is designed around two core concepts of agent and environment. The agent describes the method of step-wise running a RL algorithm in order to solve an OpenAI Gym environment’s task. Since the environments share a common agent interface, the users can write their agents with their needs and research aim. The different agents can then be applied to any Gym environment and vice versa.

The unified environment interface class, \( Env \), includes the following methods:

- \( reset(self) \): Resets the environment’s state (after terminal state) and returns initial observations.
- \( step(self, action) \): Performs one time-step and returns \( observation, reward, done, info \).
The environment’s step function returns the following substantial information for reinforcement learning algorithms:

- **observation (object)**: an environment-specific object representing the state that the agent is currently in
- **reward (float)**: amount of reward achieved by the previous action
- **done (boolean)**: indicates if the episode has terminated
- **info (dict)**: information useful for debugging or logging

At each time-step, the agent chooses an action, and the environment returns an observation and a reward. The process gets started by calling `reset`, which returns an initial observation.

The performance of a RL algorithm can be measured using metrics provided by the OpenAI Gym interface, namely the average score over 100 episodes of a particular environment, or by any other custom means.
5 Experiment

This chapter describes the setup of an experiment on RL algorithms applied to the autonomous piloting task, conducted with the toolkit presented in Chapter 4. The results obtained are also presented. We use the Proximal Policy Optimization (PPO) [73] algorithm on a specific control task to find how a fixed history window memory augmentation solution mitigates distortion caused by unobserved turbulence and winds, and improves speed of learning convergence to human-level control performance.

Following the terminology from Section 4.1, the mission is to fly a prebuilt trajectory, and the plan is the trajectory comprised of two lines.

Section 5.1 describes the experiment setting in detail, and findings of the experiment are presented in Section 5.2

5.1 Experiment setting

In this section, we give a detailed presentation of the environment settings, and we describe how FlightGear and a PPO implementation on TensorFlow are incorporated into OpenAI Gym. We also present the configuration of the PPO agents and the mechanism of communication between them.

5.1.1 Environment settings

Most of the choices in environment settings were made arbitrarily as they do not affect the outcome, as long as settings are same on every run.

In order to have a human performance reference point we used a trajectory used by human pilots – the landing approach to Hilo (Hawaii, USA) International Airport (Figure 5.1).
The task starts ahead of the final approach point, 7 nautical miles away from the runway threshold, heading on 259 degrees, on an altitude of 100 feet above the final approach point - 1900 feet, with knots-indicated air speed of 100 knots; the engine rpm is 2500, the
throttle lever is at 80%; aileron, elevator and rudder at 0 degrees. The operated aircraft is a Piper PA-28A-161 “Warrior II”.

The following weather conditions are set on every test:

- Clouds: Few clouds at 7,200 feet (2,200 meters). Few clouds at 35,000 feet.
- Temperatures: Temperature 25°C (77°F), dew point 7°C (45°F).
- Pressure: QNH 1028 hPa (30.36 inHg).
- Expectations: No significant changes expected.

We ran three tests in three different turbulence settings. For reporting purposes, turbulence intensity is characterized into the following three categories [74]:

- Light: Indicated air speed (IAS) fluctuates 5 - 15 kt; turbulence that momentarily causes slight erratic changes in attitude and/or altitude.
- Moderate Turbulence: IAS fluctuates 15 - 25 kt; turbulence that is similar to light turbulence but of greater intensity. Changes in altitude and/or attitude can occur but the aircraft remains in positive control at all times.
- Severe Turbulence: IAS fluctuates more than 25 kt; turbulence that causes large, abrupt changes in altitude and/or attitude. The aircraft may be momentarily out of control.

Due to the fact that tests are time-consuming (one test takes about one day to execute), we had to focus on contrasting learning performance in non-turbulent conditions with performance in the presence of easily noticeable turbulence; we omitted tests of the effect of light turbulence and replaced it with tests in a non-turbulent environment.

The following winds and turbulence settings are used in three tests.

Test 1 (insignificant influence from unobservable factors):
- Winds: Wind from 150° (southeast) at 3 knots.
- Turbulence: 0 - No turbulence

Test 2 (moderate influence from unobservable factors):
- Winds: Wind from 150° (southeast) at 9 knots.
- Turbulence: 0.45 - Moderate turbulence.

Test 3 (severe influence from unobservable factors):
- Winds: Wind from 150° (southeast) at 15 knots.
- Turbulence: 0.9 - Severe turbulence

The desired trajectory is to linearly descend to 1800 feet altitude for 2 nautical miles, then upon reaching the final approach point, descend with 3 degrees approach glide slope and perform takedown on the runway; maintain an approach heading all the time. The run resets if plane deviates 500 meters (horizontally or vertically) or more from the trajectory or collides with terrain.

The observation space is continuous and comprised of 6 dimensions; state is represented by the following instrument readings: heading, roll, pitch, altitude, latitude, longitude. Memory is represented in the form of 10 past pairs of action and observation perceived at next act (time step), the pairs are taken from approximately the last 10 seconds of the current observation, with about 1 second between two consecutive pairs. Such a memory representation is designed to help the learner to get a clearer picture of the actual environment state. The past 10 seconds should be sufficient to obtain inference of present (or absent) turbulence through the aircraft’s attitude reaction on control actions, as the effect that turbulence imposes on an aircraft’s attitude is often momentary. At the same time, only 10 snapshots of states and actions do not increase the algorithm’s
computational complexity in the way that a long memory window would, let alone a
technique such as LSTM [27]. Each pair from 10 memorized acts depicts the direction
and magnitude of the control bias caused by unobserved factors. The last 10 seconds of
control give a relatively informative representation of control feedback for the agent to
deduce on mitigating adjustment to the control action.

All three test configurations are run with and without the memory mechanism to illustrate
the difference.

Similar to the observation space, the action space is also continuous. It is comprised of
aileron, elevator and rudder positions, with each control having values in a linear range [-
1, 1], where -1 and 1 are the extreme positions of the controls.

5.1.2 Performance measurement

In each test, the agent starts with no prior knowledge of how the controls affect the
plane’s attitude. The agent trains through 5000 episodes, then executes 10 evaluation
runs, in which no exploration or modification of agent knowledge is made, with average
performance of the 10 runs reported as final. The agent also executes 1 evaluation run
after every set of 50 training episodes to record training progress; the average across
measurements at every distance is reported.

Performance is measured as a sum of horizontal and vertical deviations from the desired
trajectory and heading. Every time data is received from the environment, a check is
made to see whether the plane is 0.1 nautical mile closer to the runway threshold from the
last point of deviation measurement. If it is, then deviations from the desired trajectory
(in meters) and deviation from the required heading (in degrees) are measured. As soon
as the episode ends, all measurements are saved to a file.
Performance criteria from the Pilot Proficiency Check [66] were taken as nominal human performance level, namely:

- prior to the final approach course, maintain declared altitudes (±100 feet) without descending below applicable minimum altitudes, and maintain headings (±10 degrees);
- on the final segment, maintain within 5 degrees of the desired track

The sum of horizontal and vertical deviation from the desired trajectory is used as the punishment of the agent in the training process. In the case of a collision or a deviation that leads to episode reset, the reward is set to -1000 to motivate the learner against such decisions, as that reward is the lowest that the agent can receive.

5.1.3 PPO agent configuration

The idea in policy approximation was to take gradient ascent steps on the objective function, guiding the agent towards more efficient actions and higher rewards. However, choosing the step size was challenging: too small - the training process is too slow, too high - there is too much variability in the training. PPO improves the stability of the training by limiting the policy update at each training step. Experimental tests of PPO on a collection of benchmark tasks, including simulated robotic locomotion and playing a Atari game, showed that PPO outperforms other online policy gradient methods, and overall strikes a favorable balance between ease of implementation, sample complexity, and ease of tuning [73].

We used a TensorForce [75] implementation of PPO that runs on the TensorFlow framework. The ANN structure (Figure 5.2) has 90 input nodes for memory and 6 for observation, the memory node layer is followed by layers of 64, 32, 16 and 8 nodes,
observation nodes are followed by an 8-node layer directly, then both 8-node layers are connected to a single 8-node layer followed by another 7 layers of 8 nodes each, and finally the last layer consists of 3 nodes of control action. Each layer is fully connected to its successor, and the activation function is hyperbolic tangent (tanh).

Figure 5.2: ANN structure of the PPO agent.

We used experience replay memory [76] with a capacity of 5000 samples. The idea behind experience replay is to speed up convergence, not only by using observed state transitions (the experience) once, but replaying them repeatedly to the agent as if they were new observations collected while interacting with the system. The transitions collected when using experience replay resemble the connection between individual states and make more efficient use of this information, by spreading it along these connections, and, ideally, speeding up convergence.

As in [73], we used the Adam optimizer [77]. The Adam optimization algorithm is an extension to stochastic gradient descent. Stochastic gradient descent maintains a constant learning rate for all weight updates and the learning rate does not change during training.
Adam computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients; the name Adam is derived from adaptive moment estimation.

An exploration parameter $\epsilon$ linearly decays from 1 to 0.1 during the 5000 training episodes.

### 5.1.4 Cointegration

OpenAI Gym was used as a common platform for running the experiment. We made a custom environment class for the Gym that runs the FlightGear application with the settings described in Section 5.1.1. During construction of the class, an instance of FlightGear is launched with a command line command that takes specifics of the environment as a set of arguments\(^9\). In addition to a window that the flight simulator opens, it also reports its state into the standard out and standard error streams, which are captured by the process instance variable to monitor the state. As soon as the simulator is fully loaded, it automatically sets on pause, as specified by one of the command line arguments. A UDP connection is then established with the FlightGear instance, to read environment parameters and send control actions. After that, a TensorForce implementation of a PPO agent\(^10\) is launched with settings specific to the test. The implementation is readily applicable to the OpenAI Gym environment interface and can

\(^9\) FlightGear command line options: http://wiki.flightgear.org/Command_line_options

be run in a training session by instantiating a Runner\(^{11}\) object of TensorForce, the constructor of which takes an environment and an agent as parameters. Approximately every 0.1 seconds, an observation is received from the environment, and the corresponding action is generated by the agent and sent to the environment. As soon as training is complete, the next observation is read and the reward is calculated.

Communication is implemented via the UDP protocol. FlightGear is launched with two open UDP ports: one for sending observations and another for receiving control signals, which is done by reading and setting properties of the FlightGear property tree\(^{12}\); the application connects to the port and converts inbound and outgoing packets into a format appropriate for the environment. FlightGear conveniently allows using custom communication protocols\(^{13}\) (defined in XML format) that allow sending and receiving specified data only.

### 5.2 Results

Tests were executed on a single computer equipped with an AMD FX-6300 processor and 8Gb DDR3 RAM. The JSBSim FDM model was used, and the flight dynamics simulation was accelerated by 16 times for time economy. After running three tests with different wind and turbulence settings, we have evidence to believe that a 10-second history window improves the convergence speed of learning.

\(^{11}\) TensorForce Runner: [https://github.com/reinforceio/tensorforce/blob/master/docs/runner.md](https://github.com/reinforceio/tensorforce/blob/master/docs/runner.md)

\(^{12}\) FlightGear property tree: [http://wiki.flightgear.org/Property_tree](http://wiki.flightgear.org/Property_tree)

\(^{13}\) FlightGear generic protocol: [http://wiki.flightgear.org/Generic_protocol](http://wiki.flightgear.org/Generic_protocol)
We measured learning performance as the average deviations of evaluation runs that were executed after every 50 training episodes; sums of horizontal and vertical deviations larger than 1000 are not displayed as the episode resets in case such a deviation is reached. Similarly, heading deviations larger than 50 degrees are also omitted.

The data are summarized in the charts below. Tables with descriptive statistics provide an overview of collected data. Inferential statistics tables provide an assessment of memory augmentation effectiveness. To contrast learning performance with and without memory augmentation and to assess statistical significance of the performance difference, we applied the following statistical technique. We combined all performance data points (of learners with and without memory) into a single data sample and introduced an additional binary variable to distinguish different learners’ data points. The variable takes a value of 1 for datapoints produced by the learner with memory and 0 for the learner without; it essentially represents the effect of the memory augmentation on learning performance. We shall call it the memory factor (mf). We then applied linear regression on the data according to the following model:

\[
\text{performance} = \beta_0 * \text{intercept} + \beta_1 * \text{mf} + (\beta_2 + \beta_3 * \text{mf}) * \text{test\_run}
\]

where performance is the dependent variable, mf and test\_run are independent variables: memory factor and number of a test run correspondingly; intercept is the expected mean value of the dependent variable when all independent variables are zeros. \(\beta\)’s are estimated coefficients. \(\beta_1\) is an intercept shift relative to the omitted category (no memory), \(\beta_2\) is the slope of the “no memory” performance linear trend and \(\beta_3\) is the slope difference between trends of “no memory” and “with memory” performances. The
coefficient of the intercept is of no particular interest. A traditional level of significance $\alpha = 0.05$ is used in the analysis. In test 1 (insignificant influence of unobservable factors), the convergence speed of learning is almost identical for PPO without memory and with memory. The distance deviation chart and the heading deviation chart (Figures 5.3, 5.4) show that the memory window can slightly improve or worsen the performance of learning. The linear trend of the memory-augmented learner performance has 75 meters of negative intercept shift, but a less steep slope for 1 meter per test. Such an ambiguous effect of memory on convergence speed in a test with insignificant unobservable factors, probably, can be explained by the fact that the representation of control feedback provides very little useful information to the agent and that little help may be outweighed by the adverse effect of an enlarged observation space. Training may be less efficient on larger input networks, as additional training iterations are spent on finding which inputs are more correlated with the outcome. However, both $\beta_1$ and $\beta_3$ in both position and heading deviation test data did not cross the significance level cut-off value of 0.05, and thus are not statistically significant, even though $\beta_1$ in position deviation just very slightly missed the significance level (Tables 5.2, 5.4). This leads to a conclusion that the memory window does not impose a statistically significant effect on the learning performance in conditions when unobservable factors are insignificant.
Figure 5.3: Test 1. Distance deviation.

![Graph showing the sum of horizontal and vertical deviation with trend lines for no memory, with memory, and human level performance.]

Table 5.1: Test 1. Descriptive statistics for distance deviation data

<table>
<thead>
<tr>
<th></th>
<th>Median</th>
<th>Minimum</th>
<th>Mean</th>
<th>Maximum</th>
<th>Std. Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>no memory</td>
<td>68.13038106</td>
<td>2.199318855</td>
<td>226.7061181</td>
<td>1000</td>
<td>296.7060899</td>
</tr>
<tr>
<td>with memory</td>
<td>106.4383451</td>
<td>2.535240979</td>
<td>207.1590157</td>
<td>1000</td>
<td>257.3734493</td>
</tr>
</tbody>
</table>

Table 5.2: Test 1. Inferential statistics for distance deviation data

<table>
<thead>
<tr>
<th></th>
<th>Coefficients</th>
<th>Standard Error</th>
<th>t Stat</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>beta_0</td>
<td>673.5909115</td>
<td>27.83628632</td>
<td>24.19830374</td>
<td>8.87409E-61</td>
</tr>
<tr>
<td>beta_1</td>
<td>-74.92442307</td>
<td>39.36645364</td>
<td>-1.90325593</td>
<td>0.058472663</td>
</tr>
<tr>
<td>beta_2</td>
<td>-8.849203829</td>
<td>0.478550974</td>
<td>-18.49166402</td>
<td>7.61115E-45</td>
</tr>
<tr>
<td>beta_3</td>
<td>1.096580607</td>
<td>0.676773278</td>
<td>1.620307189</td>
<td>0.106774055</td>
</tr>
</tbody>
</table>
Test 2 (moderate influence of unobservable factors) indicates that memory augmentation helps to mitigate the distortion caused by notable unobservable factors. In spite of the lack of any perception of unobserved factors, the memoryless agent successfully converges to human level performance, but memory-augmented learning convergence is faster. PPO performs well in the partially-observable Markov Decision Process.
(POMDP) setting because it is essentially a policy-gradient approach, which in contrast to value-function approximation methods, cannot diverge or oscillate, as this approach does not require the agent to estimate an observation.

However, as the test 2 charts (Figures 5.5, 5.6) show, the PPO agent augmented with memory converges faster than its memoryless counterpart. In both position deviation and heading deviation data, $\beta_1$ has a noticeable negative value of -77 and statistically significant characteristics (Tables 5.6, 5.8), which indicates that the control feedback is a somewhat accurate representation of the unobserved factors and that the memory augmentation improves learning performance. Contradictory to this conclusion, $\beta_3$ has a positive value; however, the value is very small and far from being statistically significant.

Figure 5.5: Test 2. Distance deviation.
Table 5.5: Test 2. Descriptive statistics for distance deviation data

<table>
<thead>
<tr>
<th></th>
<th>Median</th>
<th>Minimum</th>
<th>Mean</th>
<th>Maximum</th>
<th>Std. Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>no memory</td>
<td>108.7412029</td>
<td>9.547243853</td>
<td>283.5834407</td>
<td>1000</td>
<td>310.5216368</td>
</tr>
<tr>
<td>with memory</td>
<td>107.5393528</td>
<td>2.010498087</td>
<td>228.4643475</td>
<td>1000</td>
<td>255.6576226</td>
</tr>
</tbody>
</table>

Table 5.6: Test 2. Inferential statistics for distance deviation data

<table>
<thead>
<tr>
<th></th>
<th>Coefficients</th>
<th>Standard Error</th>
<th>t Stat</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>beta_0</td>
<td>777.3463553</td>
<td>26.28908529</td>
<td>29.56916708</td>
<td>3.52229E-74</td>
</tr>
<tr>
<td>beta_1</td>
<td>-77.01472462</td>
<td>37.17838096</td>
<td>-2.071492158</td>
<td>0.039622278</td>
</tr>
<tr>
<td>beta_2</td>
<td>-9.183150875</td>
<td>0.451952075</td>
<td>-20.31885984</td>
<td>3.95586E-50</td>
</tr>
<tr>
<td>beta_3</td>
<td>0.487684397</td>
<td>0.639156754</td>
<td>0.763012193</td>
<td>0.446373491</td>
</tr>
</tbody>
</table>

Figure 5.6: Test 2. Heading deviation.

![Heading deviation graph](image)

Table 5.7: Test 2. Descriptive statistics for heading deviation data

<table>
<thead>
<tr>
<th></th>
<th>Median</th>
<th>Minimum</th>
<th>Mean</th>
<th>Maximum</th>
<th>Std. Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>no memory</td>
<td>27.27638616</td>
<td>2.501330804</td>
<td>26.39149336</td>
<td>50</td>
<td>16.28765868</td>
</tr>
<tr>
<td>with memory</td>
<td>21.25155221</td>
<td>2.050178018</td>
<td>21.96027574</td>
<td>50</td>
<td>16.22701084</td>
</tr>
</tbody>
</table>
Similarly to test 2, test 3 (severe influence of unobservable factors) shows that the PPO agent can be readily applied to POMDP settings, as the memory window boosts the convergence speed. In addition to that, the results charts (Figures 5.7, 5.8) indicate that the more influence unobservable factors have, the more efficiently the memory window affects learning. The difference in convergence speed between the agent with memory and the agent without memory in test 3 is greater than in test 2. In both position deviation and heading deviation data, $\beta_1$ and $\beta_3$ are negative, which means that not only does the memory-augmented agent show better performance in test runs on average but also that it learns faster as deviation decreases more per test run. The only insignificant coefficient is $\beta_3$ in position deviation data.

It is evidence that a small memory window effectively alleviates the harmful influence of the unobservable factors on the aircraft control. It also shows that the beneficial effect of the memory window is commensurate with the magnitude of the adverse unobservable factors, which is expected as the control feedback part of the observation is a representation of the unobserved factors.
Table 5.9: Test 3. Descriptive statistics for distance deviation data

<table>
<thead>
<tr>
<th></th>
<th>Median</th>
<th>Minimum</th>
<th>Mean</th>
<th>Maximum</th>
<th>Std. Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>no memory</td>
<td>447.9886449</td>
<td>35.58054906</td>
<td>511.6406519</td>
<td>1000</td>
<td>328.6378174</td>
</tr>
<tr>
<td>with memory</td>
<td>365.1040672</td>
<td>9.309182533</td>
<td>414.1807346</td>
<td>1000</td>
<td>336.2754124</td>
</tr>
</tbody>
</table>

Table 5.10: Test 3. Inferential statistics for distance deviation data

<table>
<thead>
<tr>
<th></th>
<th>Coefficients</th>
<th>Standard Error</th>
<th>t Stat</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>beta_0</td>
<td>1072.189162</td>
<td>13.79049495</td>
<td>77.74841776</td>
<td>2.9482E-149</td>
</tr>
<tr>
<td>beta_1</td>
<td>-85.66711644</td>
<td>19.50270498</td>
<td>-4.392576133</td>
<td>1.83187E-05</td>
</tr>
<tr>
<td>beta_2</td>
<td>-11.0999705</td>
<td>0.237081007</td>
<td>-46.81931566</td>
<td>2.3256E-108</td>
</tr>
<tr>
<td>beta_3</td>
<td>-0.23352081</td>
<td>0.335283175</td>
<td>-0.696488302</td>
<td>0.486948235</td>
</tr>
</tbody>
</table>

Figure 5.7: Test 3. Distance deviation.
Figure 5.8: Test 3. Heading deviation.

Table 5.11: Test 3. Descriptive statistics for heading deviation data

<table>
<thead>
<tr>
<th></th>
<th>Median</th>
<th>Minimum</th>
<th>Mean</th>
<th>Maximum</th>
<th>Std. Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>no memory</td>
<td>43.79468227</td>
<td>5.097541287</td>
<td>38.76309035</td>
<td>50</td>
<td>13.2727416</td>
</tr>
<tr>
<td>with memory</td>
<td>35.77742356</td>
<td>1.897183197</td>
<td>30.16737183</td>
<td>50</td>
<td>15.6166413</td>
</tr>
</tbody>
</table>

Table 5.12: Test 3. Inferential statistics for heading deviation data

<table>
<thead>
<tr>
<th></th>
<th>Coefficients</th>
<th>Standard Error</th>
<th>t Stat</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>beta_0</td>
<td>59.72437709</td>
<td>0.906239632</td>
<td>65.90351492</td>
<td>1.05E-135</td>
</tr>
<tr>
<td>beta_1</td>
<td>-2.873041563</td>
<td>1.281616378</td>
<td>-2.241732871</td>
<td>0.026099617</td>
</tr>
<tr>
<td>beta_2</td>
<td>-0.415074985</td>
<td>0.015579731</td>
<td>-26.64198629</td>
<td>4.5682E-67</td>
</tr>
<tr>
<td>beta_3</td>
<td>-0.113320336</td>
<td>0.022033067</td>
<td>-5.143193856</td>
<td>6.5242E-07</td>
</tr>
</tbody>
</table>
6 Discussion

6.1 Future work

Several aspects of this work can be improved. An immediate continuation of the current state of the work could be similar tests of long and complex flight routes, so that the learner would train to perform various maneuvers under different weather conditions. Another obvious addition to the work could be an introduction of visual input so that a learner could use the view outside of the cockpit or synthetic vision system data as input for a more comprehensive representation of current state.

Different memory techniques can be tested for improvement of the convergence boost in the presence of unobserved factors. Specifically, it would be interesting to see if a small fixed history window gives a more or less informative representation of wind bursts and turbulence than the LSTM technique. It is also of particular interest whether an arbitrarily long memory mechanism could pick up significantly lagged control irregularity patterns and use them to improve overall flight performance.

Additionally, various RL algorithms’ performance could be compared under difficult conditions, so that systematic analysis could be made in order to make objective rankings of recent solutions in RL for autonomous piloting.

6.2 Conclusion

In this thesis, we provided a short overview of the mathematical background of reinforcement learning concepts. We briefly described the current state of arguably most developed area of RL application, to the problem of controlling autonomous vehicles. We
contrasted this with RL solutions in autonomous piloting tasks, which led to proposal of a framework for autonomous piloting tasks.

Development in the area of autonomous aircraft control systems has the potential to equip the aircraft industry with higher economical efficiency and safety; the application of reinforcement learning to the area is one of the promising approaches towards more efficient solutions. However, the control system design process can be resource-intensive, as the efficiency of a particular ML algorithm on a new task is not known in advance. Therefore, there is a need for a testbed to compare various ML solutions and to make objective conclusions about their performances.

6.2.1 Thesis contributions

We contributed a theoretical (terminological) framework and an example of its practical implementation that can serve as a benchmark platform for RL solutions. The framework uses open source components and is available for experiments to any researcher. It is important to have a common experimental toolkit for two main reasons:

- For reproducibility of attained results, as reproducibility is the mechanism that fixes obtained findings as facts and make them available for analyses.
- For benchmarking. It is very difficult to draw a general conclusion from results of a RL solution’s performance in a specific benchmark; however, there is no way to compare qualities of various solutions without a common testbed.

We also provided an example of such an experiment by testing a solution to a problem specific to the application of RL in a real-life scenario.

One of the problems that automated and autonomous piloting systems face is the partial observability of the environment; specifically, severe turbulence and gusty winds are
often unobservable with regular means of visual and instrumental reception but are able to complicate aircraft control dramatically. Partial observability is a known problem in the RL area and multiple solutions exist in various RL algorithms. A fixed time memory window is a possible approach that is one of the simplest, which makes it one of the least computationally complex. It is significantly less computationally intensive than LSTM, a more common approach in the area of RL autonomous control. However, unlike LSTM, fixed memory window cannot capture patterns in events that occurred later than length of the window. We conducted an experiment to test a fixed history window approach as a solution to unobserved turbulence that compromises control efficiency. An experiment was performed, to determine whether a small fixed memory window can mitigate the adverse influence of such unobserved factors as wind bursts and turbulence. Tests show that the memory mechanism that encapsulates control feedback is an informative input for the learning agent, as long as the unobserved factors affect control behavior significantly. Specifically, 10 snapshots of states and actions from the past 10 seconds is a sufficiently accurate representation of how unobserved factors affect the aircraft’s reaction of controls on its attitude, sufficiently accurate for the learner to infer from it the changes in control actions required to alleviate the adverse effects caused by the unobservables.
Bibliography


19. *Autonomous Driving in the Real World: Experiences with Tesla Autopilot and Summon*. Dikmen, Murat and Burns, Catherine M. s.l. : 8th International Conference on
Automotive User Interfaces and Interactive Vehicular Applications, 2016. 978-1-4503-4533-0.


74. JEPPESEN. *JAA ATPL Meteorology*. s.l. : Oxford Aviation Academy, 2011. 9781906202583.


Curriculum Vitae

Candidate's full name: Dmitry Shcherbakov.

Universities attended:

- Far Eastern Federal University, Manager, 2011.
- University of New Brunswick, Master of Arts, 2017.

Publications: None.

Conference Presentations: None.