FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

ASSIGNMENT OF MASTER'S THESIS

Title:	Learning to land
Student:	Bc. Matúš Tóth
Supervisor:	MSc. Juan Pablo Maldonado Lopez, Ph.D.
Study Programme:	Informatics
Study Branch:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	Until the end of summer semester 2019/20

Instructions

A crucial challenge for self-flying tripulated aircraft is handling the necessary maneuvers for landing. In this work we explore the feasibility of teaching an airplane to land. The method is trained and tested using professional flight simulation software.

1) Provide a general context of the landing problem and survey of the existing methods for automatic landing.
2) Explore in particular the approaches around model-free methods (reinforcement learning).
3) Consider possible optimizations and improvements.
4) Implement the improved method and analyze the performance of the achieved results.

References

Will be provided by the supervisor.

Ing. Karel Klouda, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague October 12, 2018

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Learning to land

Bc. Matúš Tóth

Department of Applied Mathematics
Supervisor: MSc. Juan Pablo Maldonado Lopez, Ph.D.

February 14, 2019

Acknowledgements

I would like to thank my supervisor MSc. Juan Pablo Maldonado Lopez, Ph.D. for guiding my work and for essential advice, Ing. Marek Batelka for valuable information about aviation and flying in general. Further I would like to thank my girlfriend, close friends and friends for support during writing this thesis.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46(6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity. However, all persons that makes use of the above license shall be obliged to grant a license at least in the same scope as defined above with respect to each and every work that is created (wholly or in part) based on the Work, by modifying the Work, by combining the Work with another work, by including the Work in a collection of works or by adapting the Work (including translation), and at the same time make available the source code of such work at least in a way and scope that are comparable to the way and scope in which the source code of the Work is made available.

In Prague on February 14, 2019 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Tóth, Matúš. *Learning to land.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2019.

Abstrakt

Táto práca sa zaoberá problematikou umelej inteligencie zameranej na pristávanie dopravného lietadla Boeing 737-800. Ako simulačné prostredie využíva X-Plane 11 a na základe dát extrahovaných z tohoto simulátora natrénujeme rôzne modely zachytávajúce dynamiku letu počas pristávania. Jednotlivé modely optimalizujeme a porovnávame, ako presne dokážu spomínanú dynamiku reprezentovať. Tieto modely využívame na efektívne natrénovanie agentov pomocou učenia so spätnou väzbou pomocou metódy krížnej entropie využitím troch rôznych odmeňovacích funkcií. Cieľom tohoto trénovania je naučiť agentov s lietadlom pristáť. Agentov optimalizujeme vzhľadom na odmeňovacie funkcie a na záver vyberieme troch elitných agentov, ktorých otestujeme naspäť v prostredí X-Plane. Výstupom je zhodnotenie ako úspešní elitní agenti sú vzhľadom na podmienky bezpečného a stabilizovaného pristátia a vyvodenie záveru, či by umelá inteligencia mohla nahradiť pilota.

Klíčová slova Autonómne lietadlo, dynamický model, letectvo, metóda krížovej entropie, stabilizované priblíženie, učenie so spätnou väzbou, umelá inteligencia, X-Plane

Abstract

This work is concerned with the issue of autonomous landing of airliner Boeing 737-800. As the simulation environment, X-Plane 11 is used, and based on the data obtained from this simulator, we develop various models capturing flight dynamics during landing. We optimize each model and compare how accurately they represent the dynamics. These models are utilized for training policy effectively using reinforcement learning algorithm with cross entropy method and three different reward functions. The aim of this training is to teach the aircraft to land. The agents are optimized according to reward functions and in the end we select three elite agents that we test back in the X-Plane environment. The outcome of this evaluation is how successful the elite agents are with regard to the conditions of safe and stabilized approach and conclusion, whether the pilot could be replaced by the artificial intelligence.

Keywords Artificial intelligence, autonomous aircraft, aviation, cross-entropy method, dynamics model, reinforcement learning, stabilized approach, X-Plane

Contents

List of Figures

List of Tables

Introduction

Motivation

Money. Nowadays, everything revolves around money. Aviation industry is no exception - quite the opposite. Aviation industry is dependent on money circulation a lot more than other industries, because every single part of the aircraft undergoes strict inspections and has to fit various regulations, which prolongs and overcharges the process of constructing an airplane and its introduction into real use.

How much though?

Let us get the idea of how much money is in the aviation industry. Airlines need to own or lease airplanes in the first place. How many airplanes do airlines own? As the table 0.1 shows, the differences are great even between the largest airlines. [1]

Rank	**Airline**	**Country**	**Fleet**
1	American Airlines	United States	952
2	Delta Air Lines	United States	850
3	United Airlines	United States	745
4	Southwest Airlines	United States	697
5	China Southern Airlines	China	545
6	China Eastern Airlines	China	486
7	Ryanair	Ireland	413
8	Air China	China	392
9	FedEx Express	United States	371
10	Turkish Airlines	Turkey	329

Table 0.1: Major airlines by fleet size

Aircraft	In service	Orders	Price (mil. USD)
Boeing 737-800	304	—	102.2
Airbus A321-200	219	—	118.3
Airbus A319-100	126	7	92.3
Boeing 737 MAX 8	16	84	117.1
Airbus A321neo	—	100	129.5

Table 0.2: American airlines fleet

However, this still does not say how much it costs. Let us have a look at the fleet of the American Airlines. As they have many different types of aircrafts, we only pick the most common (or to be common) aircrafts out of the fleet.

Getting deeper into analyzing the expenses of an airline, we see how many airplanes does the American airlines own and will own in following years in the table 0.2 [2]. As we can see, also the prices of the airplanes are included and they are sky-high [3, 4]. Furthermore, the expenses do not end with purchasing the fleet - quite the opposite. Now that we have the fleet, we need to have the aircrafts insured, maintained and also populated by the crew.

To get the idea of how to measure the operating cost of an aircraft, we need to understand two terms first:

1. **Block Hour** means the period of time (in minutes) beginning when an Aircraft first moves from the ramp blocks in connection with a Scheduled Flight, a Non-Scheduled Flight or a Charter Flight and ending when the Aircraft next comes to a stop at the ramp at any station or other point of termination as recorded by ACARS or another mutually agreed system, divided by sixty (60). [5]

2. **Cost per available seat mile** (CASM) is a commonly used measure of unit cost in the airline industry. CASM is expressed in cents to operate each seat mile offered, and is determined by dividing operating costs by ASMs. This number is frequently used to allow a cost comparison between different airlines or for the same airline across different time periods (say for one year vs the preceding year). A lower CASM means that it is easier for the airline to make a profit, as they have to charge less to break even. A low CASM, however, is by no means a guarantee of profitability. Further, CASM should only be compared across airlines with care. For instance, all other things being equal, an airline with a longer average stage length will have a lower CASM, because fixed costs will account for a lower portion of its total costs. For this reason, to be meaningful, CASM comparisons across different airlines generally require, at a minimum, that CASMs for all airlines be adjusted to a

> common stage length, or that the CASMs be graphed versus the stage length of all the airlines being compared. [6]

Understanding these two terms, we can measure how much does a flight cost. Let us take a 1000 mile flight with a Boeing 737-800. Having 172 seats [7], we generate 172 000 ASMs. According to [8], cost per available seat mile (CASM) for year 2017 for the American airlines was $0.138 and so this example flight would cost $23736. Considering that this flight would take approximately 3 hours, we get cost per block hour of $7912. As [9] says, total cockpit cost per block hour of the American airlines for year 2017 was $1236 which consists of paying two pilots. We see that the cost of pilots is 15.62% out of the total cost per block hour.

Getting rid of the aircrew

Knowing how much of operating cost of an airplane do pilots consume, we can see the logic in downsizing the aircrew. In past (before 1980), there were three people in the cockpit. As the navigation systems were not advanced enough for the pilots to rely on them, apart from the captain and the first officer, there was also a navigator [10]. However, this downsizing was quite a long time ago.

Lately, there were statements from Boeing, that there are and will be efforts to reduce the size of aircrew even more. The single pilot operated jets are at hand [11] and autonomous jets are the vision of the future [12], which means that the aircraft will have to handle even the most difficult tasks performed by pilots.

Looking back at the money, this would save great amount of money as the American airlines employs 14000 pilots [13] with average yearly salary of $152,461 [14].

What does the airplane need to handle?

Considering that we want an aircraft to fly the whole flight by itself, we need to know whether this task is feasible. To see how difficult each phase of the flight is, we should look at the number off accidents that happen in each phase.

As we see from the graphs in the picture 0.1 [15], 49% of fatal accidents and 44% of onboard fatalities happen after the final approach fix even though this phase takes only 4% of the 1.5 hour flight (during longer flights this ratio lowers, as mostly only the cuise is prolonged).

There are two phases after final approach fix and that are final approach and landing. Let us consider this part of the flight as the part that requires the most skill to perform and if the aircraft can perform it by itself, we can assume that it will be able to handle the whole flight as well.

The question now stands - can the aircraft land?

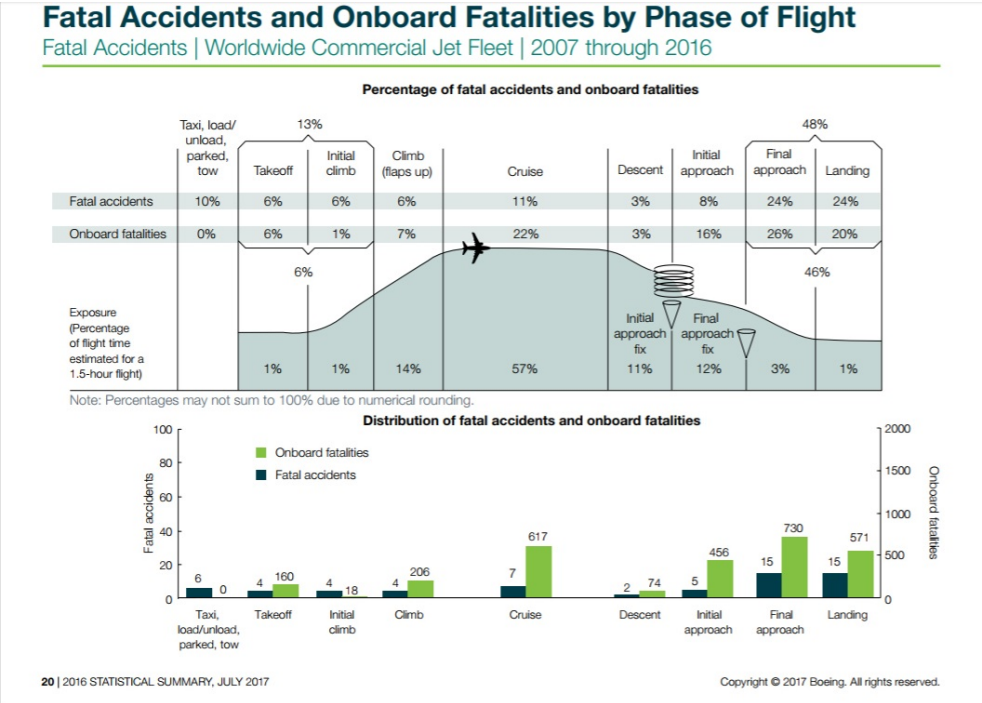


Figure 0.1: Fatal Accidents and Onboard Fatalities by Phase of Flight

Chapter 1

Theoretical background

To answer the question whether the aircraft can land autonomously, we need certain theoretical background.

1.1 Aviation

Demonstrating autonomous landing requires us to know what it even means to land an aircraft. As we showed in previous section, the most demanding part of flight is after final approach fix. Now we need to know what these phases are and why are they so dangerous.

1.1.1 Glossary

In this section we describe the terminology of angles and speeds.

1.1.1.1 Angles

All the further described angles are shown in the picture 1.1 [16]. According to [16], angle of attack (AOA) is the angle between the oncoming air or relative wind and a reference line on the airplane or wing. Sometimes, the reference line is a line connecting the leading edge and trailing edge at some average point on the wing. Most commercial jet airplanes use the fuselage centerline or longitudinal axis as the reference line. It makes no difference what the reference line is, as long as it is used consistently.

AOA is sometimes confused with pitch angle or flight path angle. Pitch angle (attitude) is the angle between the longitudinal axis (where the airplane is pointed) and the horizon. This angle is displayed on the attitude indicator or artificial horizon.

Flight path angle is defined in two different ways. To the aerodynamics, it is the angle between the flight path vector (where the airplane is going) and the local atmosphere. To the flight crew, it is normally known as the

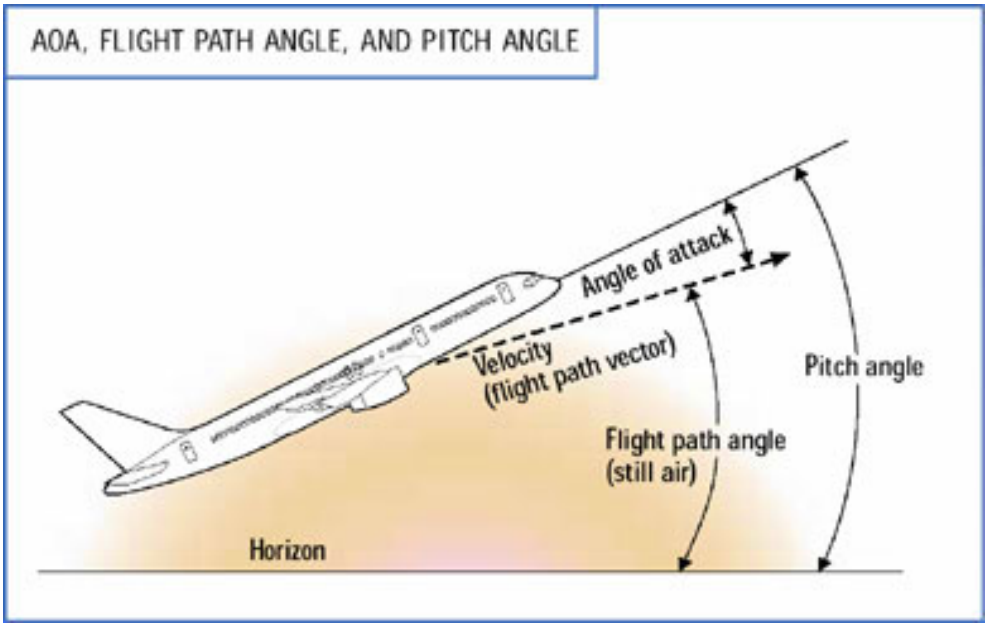


Figure 1.1: Angle terminology

angle between the flight path vector and the horizon, also known as the climb (or descent) angle. Air-mass-referenced and inertial-referenced flight path angles are the same only in still air (i.e., when there is no wind or vertical air movement). For example, in a headwind or sinking air mass, the flight path angle relative to the ground will be less than that referenced to the air. On the newest commercial jet airplanes, this angle can be displayed on the primary flight display and is calculated referenced to the ground (the inertial flight path angle).

AOA is the difference between pitch angle and flight path angle when the flight path angle is referenced to the atmosphere. Because of the relationship of pitch angle, AOA, and flight path angle, an airplane can reach a very high AOA even with the nose below the horizon, if the flight path angle is a steep descent.

1.1.1.2 Speeds terminology

All the further described speeds are described in the picture 1.2 [17]. Indicated airspeed (IAS or KIAS) means the speed of an aircraft as shown on its pitot static airspeed indicator, calibrated to reflect standard atmosphere adiabatic compressible flow at sea level, uncorrected for airspeed system errors.

Calibrated airspeed (CAS) is indicated airspeed corrected for instrument errors, position errors (due to incorrect pressure at the static port) and installation errors. Indicated airspeed will be a few knots lower. This is because the pitot tube is not picking up as many air molecules as it should, because of

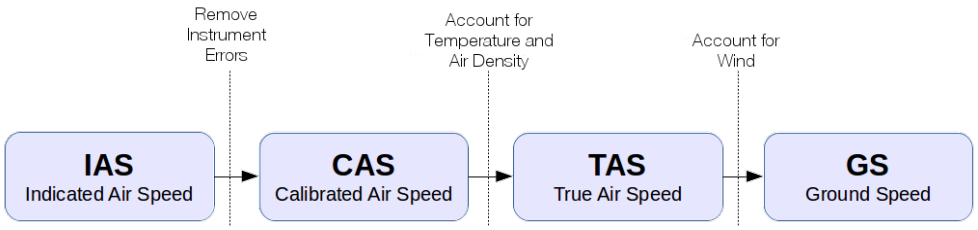


Figure 1.2: Speeds terminology

its angle of attack. Your real speed did not change, but the airspeed indicator thinks it did.

Equivalent airspeed (EAS) is the airspeed at sea level in the International Standard Atmosphere at which the dynamic pressure is the same as the dynamic pressure at the true airspeed (TAS) and altitude at which the aircraft is flying.

True airspeed (TAS or KTAS) is the speed of the aircraft relative to the atmosphere. The true airspeed and heading of an aircraft constitute its velocity relative to the atmosphere.

Ground speed (GS) is the speed of the aircraft relative to the ground. This speed is the combination of the true airspeed vector of the aircraft and the speed vector of wind at aircraft altitude.

1.1.2 Instrument approach procedure

According to [18] an Instrument approach procedure (IAP) is a series of predetermined manoeuvres by reference to flight instruments with specified protection from obstacles from the initial approach fix, or where applicable, from the beginning of a defined arrival route to a point from which a landing can be completed and thereafter, if a landing is not completed, to a position at which holding or en-route obstacle clearance criteria apply.
These procedures are divided into three categories:

1. Non-precision approach (NPA) procedures
2. Approach procedures with vertical guidance (APV) - Performance-based navigation
3. Precision approach (PA) procedures - Procedures based on navigation systems

Each of the procedures may have up to 5 separate segments:

1. Arrival segment
2. Initial approach segment
3. Intermediate approach segment
4. Final approach segment
5. Missed approach segment

1.1.2.1 Arrival segment

An arrival segment permits transition from the en-route phase to the approach phase. The arrival segment starts at the latest en-route point and ends at the initial approach fix.

It can be either a defined standard instrument arrival (STAR) route published on charts or omnidirectional or sector arrival to an initial approach fix (IAF).

1.1.2.2 Initial approach

The initial approach segment is the segment of an instrument approach procedure between the initial approach fix (IAF) and the intermediate fix (IF). The initial approach segment begins at the initial approach fix (IAF) and ends at the intermediate fix (IF).

In the initial approach segment, the aircraft has left the en-route structure and it is entering the Instrument approach procedure.

1.1.2.3 Intermediate approach

That segment of an instrument approach procedure between the intermediate fix and the final approach fix. This is the segment during which the aircraft speed and configuration should be adjusted to prepare the aircraft for final approach. For this reason the descent gradient is kept as shallow as possible.

1.1.2.4 Final approach

Final approach is the last leg in an aircraft's approach to landing, when the aircraft is lined up with the runway and descending for landing. In aviation radio terminology, it is often shortened to "final". [19]

This is the segment in which alignment and descent for landing are made. Normally, final approach may be made to a runway for a straight-in landing, or to an aerodrome for a visual manoeuvre (circling). Considering our case where we look onto the American airlines and their fleet, we focus only on large

airports (not aerodromes) and so the final approach will be straight (without circling or steering).

There are various types of final approach (based on procedure category):

- Non precision approach (NPA) - this type of final approach is performed by pilot either by utilizing autopilot or simply flying by hand
- Approach with vertical guidance (APV) - here we rely on either ground based navigation aid or computer-generated navigation data
- Precision approach (PA) - these approaches u

Instrumental landing system According to [20] an instrument landing system operates as a ground-based instrument approach system that provides precision lateral and vertical guidance to an aircraft approaching and landing on a runway, using a combination of radio signals and, in many cases, high-intensity lighting arrays to enable a safe landing during instrument meteorological conditions (IMC), such as low ceilings or reduced visibility due to fog, rain, or blowing snow.

An instrument approach procedure chart (or 'approach plate') is published for each ILS approach to provide the information needed to fly an ILS approach during instrument flight rules (IFR) operations. A chart includes the radio frequencies used by the ILS components or navaids and the prescribed minimum visibility requirements.

Radio-navigation aids must provide a certain accuracy (set by international standards of CAST/ICAO); to ensure this is the case, flight inspection organizations periodically check critical parameters with properly equipped aircraft to calibrate and certify ILS precision.

An aircraft approaching a runway is guided by the ILS receivers in the aircraft by performing modulation depth comparisons. Many aircraft can route signals into the autopilot to fly the approach automatically. An ILS consists of two independent sub-systems. The localizer provides lateral guidance; the glide slope provides vertical guidance.

During this phase the pilot should get the aircraft to a state with landing configuration of control surfaces. For a safe approach and landing he has to monitor the state of the aircraft constantly - keep the approach stabilized.

ILS aids with landing to a great extent, but not every runway is equipped with such a system.

Stabilized approach Focusing on establishing and maintaining a stabilized approach and landing is a great way to avoid experiencing a loss of control. A stabilized approach is one in which the pilot establishes and maintains a constant angle glide-path towards a predetermined point on the landing runway. It is based on the pilot's judgment of certain visual clues, and depends on the maintenance of a constant final descent airspeed and configuration. [21]

All flights must be stabilized by 1000 feet above airport elevation in Instrument Meteorological Conditions (IMC - foggy or cloudy conditions, the pilot has to focus solely on the gauges and devices inside the cockpit) and 500 feet above airport elevation in Visual Meteorological Conditions (VMC - the pilot is able to clearly see the runway).
Factors of a Stabilized Approach [21]

- The aircraft is on the correct flight path
- Only small changes in heading/pitch are necessary to maintain the correct flight path
- The airspeed is not more than $V_{REF} + 20kts$ indicated speed and not less than V_{REF} (Landing reference speed or threshold crossing speed.)
- The aircraft is in the correct landing configuration
- Sink rate is no greater than 1000 feet/minute. If an approach requires a sink rate greater than 1000 feet/minute a special briefing should be conducted
- Power setting is appropriate for the aircraft configuration and is not below the minimum power for the approach as defined by the operating manual
- All briefings and checklists have been conducted
- Specific types of approach are stabilized if they also fulfil the following:
  - Instrument Landing System (ILS) approaches must be flown within one dot of the glide-slope and localizer
  - during a circling approach wings should be level on final when the aircraft reaches 300 feet above airport elevation
- Unique approach conditions or abnormal conditions requiring a deviation from the above elements of a stabilized approach require a special briefing.

An approach that becomes unstabilized below 1000 feet above airport elevation in IMC or 500 feet above airport elevation in VMC requires an immediate go-around.

A **go-around** is an aborted landing of an aircraft that is on final approach. The cause of a go-around could be many things, such as a plane on the runway or a gust of wind which blows the plane off course. If the go-around necessity is ignored, various accidents may occur. Unstabilized approaches are responsible for 14% of all accidents in approach and landing phases. [22]

1.1.2.5 Missed approach

A missed approach segment must be followed if the approach cannot be continued. In this segment, the pilot is faced with the demanding task of changing the aircraft configuration, attitude and altitude.

1.1.2.6 Landing

Landing - Landing is the process of getting an aircraft from short final to a full stop on the runway. From the beginning of the landing flare until aircraft exits the landing runway, comes to a stop on the runway, or when power is applied for takeoff in the case of a touch-and-go landing. [23]

To land, the airspeed and the rate of descent are reduced such that the object descends at a low enough rate to allow for a gentle touch down. Landing is accomplished by slowing down and descending to the runway.

Autoland Autoland systems were designed to make landing possible in visibility too poor to permit any form of visual landing, although they can be used at any level of visibility. They may also include automatic braking to a full stop once the aircraft is on the ground, in conjunction with the autobrake system, and sometimes auto deployment of spoilers and thrust reversers.

Autoland may be used for any suitably approved instrument landing system (ILS) or microwave landing system (MLS) approach, and is sometimes used to maintain currency of the aircraft and crew, as well as for its main purpose of assisting an aircraft landing in low visibility and/or bad weather.

As mentioned in section 1.1.2.4, not every runway is equipped with such system. Also, there have been reported cases where utilization of autoland resulted in touchdown outside of runway, mainly caused by poor signal of ILS. [24]

Another point against using autoland is that pilots claim it does not provide safer nor better (smoother and more precise) landing than any pilot. [25]

1.2 Survey of approaches

There already are works that focus on autonomous control of various vehicles.

Focusing only on aircrafts, there are two main branches of how to look at this problem. First one is vision-based, which requires more computational power and the second one is dynamics-based approach.

There are many papers that describe autonomous control of helicopters. First approach is vision-based. Here notable mentions are [26] where they use GPS and vision to locate the landing target.

From the dynamics-based approaches there is for example [27] introducing training of a model that captures helicopter dynamics and then using rein-

forcement learning algorithm to train a policy that flies inverted helicopter flight.

Similarly, other papers, such as [28], [29] first train dynamics model and then apply reinforcement learning to it.

Another approach to this, introduced in [30], is utilization of inverse reinforcement learning, where we learn the trajectory first and then learn the reward function.

As to the fixed-wing aircrafts, many of the approaches from helicopters are applicable here as well. In [31], the author focuses on landing a small aircraft (Cessna 172 Skyhawk). He introduces a dynamics model for landing the aircraft and uses apprenticeship learning where there is human (or expert) input, which then is imitated by reinforcement learning policy. The same approach is used in [32].

Lastly to mention, [33] looks onto autonomous landing of reusable rockets (connected to Space-X). The author explores various approaches to the problem using control algorithms such as optimal control, linear quadratic regulator and model predictive control.

None of mentioned papers performed the learning on actual hardware (helicopters or other aircrafts), but using various simulation environments.

1.3 Machine learning

1.3.1 Reinforcement learning

According to [34] reinforcement learning (RL) is an area of machine learning concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward. The problem, due to its generality, is studied in many other disciplines, such as game theory, control theory, operations research, information theory, simulation-based optimization, multi-agent systems, swarm intelligence, statistics and genetic algorithms. In the operations research and control literature, reinforcement learning is called approximate dynamic programming, or neuro-dynamic programming. The problems of interest in reinforcement learning have also been studied in the theory of optimal control, which is concerned mostly with the existence and characterization of optimal solutions, and algorithms for their exact computation, and less with learning or approximation, particularly in the absence of a mathematical model of the environment. In economics and game theory, reinforcement learning may be used to explain how equilibrium may arise under bounded rationality. In machine learning, the environment is typically formulated as a Markov Decision Process (MDP), as many reinforcement learning algorithms for this context utilize dynamic programming techniques. The main difference between the classical dynamic programming methods and reinforcement learning algorithms is that the latter do not as-

sume knowledge of an exact mathematical model of the MDP and they target large MDPs where exact methods become infeasible.

Less formally, we can identify four main sub-elements of a reinforcement learning system: a policy, a reward signal, a value function, and, optionally, a model of the environment.

A policy defines the learning agent's way of behaving at a given time. Roughly speaking, a policy is a mapping from perceived states of the environment to actions to be taken when in those states. It corresponds to what in psychology would be called a set of stimulus–response rules or associations (provided that stimuli include those that can come from within the animal). In some cases the policy may be a simple function or lookup table, whereas in others it may involve extensive computation such as a search process. The policy is the core of a reinforcement learning agent in the sense that it alone is sufficient to determine behavior. In general, policies may be stochastic.

A reward signal defines the goal in a reinforcement learning problem. On each time step, the environment sends to the reinforcement learning agent a single number, a reward. The agent's sole objective is to maximize the total reward it receives over the long run. The reward signal thus defines what are the good and bad events for the agent. In a biological system, we might think of rewards as analogous to the experiences of pleasure or pain. They are the immediate and defining features of the problem faced by the agent. The reward sent to the agent at any time depends on the agent's current action and the current state of the agent's environment. The agent cannot alter the process that does this. The only way the agent can influence the reward signal is through its actions, which can have a direct effect on reward, or an indirect effect through changing the environment's state. In our example above of Phil eating breakfast, the reinforcement learning agent directing his behavior might receive different reward signals when he eats his breakfast depending on how hungry he is, what mood he is in, and other features of his of his body, which is part of his internal reinforcement learning agent's environment. The reward signal is the primary basis for altering the policy. If an action selected by the policy is followed by low reward, then the policy may be changed to select some other action in that situation in the future. In general, reward signals may be stochastic functions of the state of the environment and the actions taken. [34]

Whereas the reward signal indicates what is good in an immediate sense, a value function specifies what is good in the long run. Roughly speaking, the value of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state. Whereas rewards determine the immediate, intrinsic desirability of environmental states, values indicate the long-term desirability of states after taking into account the states that are likely to follow, and the rewards available in those states. For example, a state might always yield a low immediate reward but still have a high value because it is regularly followed by other states that yield high rewards. Or

the reverse could be true. To make a human analogy, rewards are somewhat like pleasure (if high) and pain (if low), whereas values correspond to a more refined and farsighted judgment of how pleased or displeased we are that our environment is in a particular state. Expressed this way, we hope it is clear that value functions formalize a basic and familiar idea.

Rewards are in a sense primary, whereas values, as predictions of rewards, are secondary. Without rewards there could be no values, and the only purpose of estimating values is to achieve more reward. Nevertheless, it is values with which we are most concerned when making and evaluating decisions. Action choices are made based on value judgments. We seek actions that bring about states of highest value, not highest reward, because these actions obtain the greatest amount of reward for us over the long run. In decision-making and planning, the derived quantity called value is the one with which we are most concerned. Unfortunately, it is much harder to determine values than it is to determine rewards. Rewards are basically given directly by the environment, but values must be estimated and re-estimated from the sequences of observations an agent makes over its entire lifetime. In fact, the most important component of almost all reinforcement learning algorithms we consider is a method for efficiently estimating values. The central role of value estimation is arguably the most important thing we have learned about reinforcement learning over the last few decades. [35]

The fourth and final element of some reinforcement learning systems is a model of the environment. This is something that mimics the behavior of the environment, or more generally, that allows inferences to be made about how the environment will behave. For example, given a state and action, the model might predict the resultant next state and next reward. Models are used for planning, by which we mean any way of deciding on a course of action by considering possible future situations before they are actually experienced. Methods for solving reinforcement learning problems that use models and planning are called model-based methods, as opposed to simpler model-free methods that are explicitly trial-and-error learners—viewed as almost the opposite of planning. [36]

1.3.1.1 Markov Decision Processes

A stochastic process has the Markov property if the conditional probability distribution of future states of the process (conditional on both past and present states) depends only upon the present state, not on the sequence of events that preceded it. A reinforcement learning task that satisfies the Markov property is called a Markov decision process, or MDP. [34]

The Agent–Environment Interface The reinforcement learning problem is meant to be a straightforward framing of the problem of learning from interaction to achieve a goal. The learner and decision-maker is called the

agent. The thing it interacts with, comprising everything outside the agent, is called the environment. These interact continually, the agent selecting actions and the environment responding to those actions and presenting new situations to the agent. The environment also gives rise to rewards, special numerical values that the agent tries to maximize over time. A complete specification of an environment defines a task, one instance of the reinforcement learning problem. More specifically, the agent and environment interact at each of a sequence of discrete time steps, $t = 0, 1, 2, 3, ...$ At each time step t, the agent receives some representation of the environment's state, s_t belongs to S, where S is the set of possible states, and on that basis selects an action, a_t belongs to $A(s_t)$, where $A(s_t)$ is the set of actions available in state s_t . One time step later, in part as a consequence of its action, the agent receives a numerical reward, $r_{t+1} \in R$, and finds itself in a new state, s_{t+1}. At each time step, the agent implements a mapping from states to probabilities of selecting each possible action. This mapping is called the agent's policy and is denoted π_t , where $\pi_t(a|s)$ is the probability that $a_t = a$ if $s_t = s$. Reinforcement learning methods specify how the agent changes its policy as a result of its experience. The agent's goal, roughly speaking, is to maximize the total amount of reward it receives over the long run. [35]

Goals and Rewards In reinforcement learning, the purpose or goal of the agent is formalized in terms of a special reward signal passing from the environment to the agent. At each time step, the reward is a simple number, Rt belongs to R. Informally, the agent's goal is to maximize the total amount of reward it receives. This means maximizing not immediate reward, but cumulative reward in the long run. We can clearly state this informal idea as the reward hypothesis:

> That all of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward).

The use of a reward signal to formalize the idea of a goal is one of the most distinctive features of reinforcement learning.

Exploration Reinforcement learning requires clever exploration mechanisms. Randomly selecting actions, without reference to an estimated probability distribution, shows poor performance. The case of (small) finite Markov decision processes is relatively well understood. However, due to the lack of algorithms that provably scale well with the number of states (or scale to problems with infinite state spaces), simple exploration methods are the most practical.

One such method is ϵ-greedy, when the agent chooses the action that it believes has the best long-term effect with probability $1 - \epsilon$. If no action which satisfies this condition is found, the agent chooses an action uniformly at

random. Here, $0 < \epsilon < 1$ is a tuning parameter, which is sometimes changed, either according to a fixed schedule (making the agent explore progressively less), or adaptively based on heuristics. [36]

Cross entropy method The cross-entropy (CE) method is a Monte Carlo method for importance sampling and optimization. It is applicable to both combinatorial and continuous problems, with either a static or noisy objective. [37]

The method approximates the optimal importance sampling estimator by repeating two phases:

1. Draw a sample from a probability distribution.
2. Minimize the cross-entropy between this distribution and a target distribution to produce a better sample in the next iteration.

Reuven Rubinstein developed the method in the context of rare event simulation, where tiny probabilities must be estimated, for example in network reliability analysis, queueing models, or performance analysis of telecommunication systems. The method has also been applied to the traveling salesman, quadratic assignment, DNA sequence alignment, max-cut and buffer allocation problems.

Apprenticeship learning As the title says, this type of reinforcement learning uses an expert to train (or speed up the training) of a policy. For example, we may have a human pilot give us an initial demonstration of helicopter flight. Given this initial training data with which to learn the dynamics, we show that it suffices to only execute exploitation policies (ones that try to do as well as possible, given the current model of the MDP). More specifically, we propose the following algorithm:

1. Have a teacher demonstrate the task to be learned, and record the state-action trajectories of the teacher's demonstration.
2. Use all state-action trajectories seen so far to learn a dynamics model for the system. For this model, find a (near) optimal policy using any reinforcement learning (RL) algorithm.
3. Test that policy by running it on the real system. If the performance is as good as the teacher's performance, stop. Otherwise, add the state-action trajectories from the (unsuccessful) test to the training set, and go back to step 2.

From this example, we see the utilization of greedy policy, as we never perform an exploration step. [38]

Chapter 2

Environment

To prove the concept of autonomous landing, we needed to set up our environment for experiments.

2.1 Machine learning

Choosing a machine learning approach to autonomous landing, we need to choose a machine learning framework. As I have the most experience with scikit-learn, I chose this framework the thesis.

Scikit-learn is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy. (scikit-learn.org)

This also sets our programming language to Python.

2.2 Flight simulator

As mentioned in 1.2, none of the papers used real hardware for training autonomous control. Similarly, we needed a flight simulator that will have a suitable API for gathering data from the environment and also sending commands to control the aircraft.

After a brief survey we chose X-Plane 11 to be the flight simulator used for this thesis. X-Plane is used by multiple organizations and industries such as NASA, Boeing, Cirrus, Cessna, Piper, Precession Flight Controls Incorporated, Japan Airlines, and the American Federal Aviation Administration. [32] X-Plane can communicate with external applications by sending and receiving flight status and control commands data over a network through User Datagram Protocol (UDP) packets.

Reading the data of X-Plane can be done in various ways:

- Output selected data to a file - with certain frequency appending a line and when the flight is reset, rewrite the file
- Send selected data via UDP
- Read datarefs (pointer-like variables) externally

On the other hand, sending data to X-Plane can be done via:

- Write datarefs externally
- Send pilot like commands externally (for example set flaps handle one down)

The datarefs can be of various types, such as integer, float or arrays and some of them are not writable. Complete list of datarefs can be found at [39]. Commands are just signals that are called, they correspond to certain pilot actions and do not take any parameters. Complete list of commands can be found at [40].

2.2.1 X-Plane interface

As mentioned, we are using Python. There are several plugins for X-Plane that utilize the UDP communication in order to gather data and send commands.

Sandy Barbour's XPlugin SDK is a project that allows python scripts to be run from Xplane - more exactly the Python interface plugin. The plugin has a control panel that allows scripts to be reloaded and also data can be sent to the two list boxes. It also has an option for enabling and disabling any script and another screen for script info. [41]

Python interface plugin is written in Python 2.7 and runs all the scripts with this version of python with no option to modify the source codes of the plugin. This is because it is already precompiled in a way that X-Plane is able to run the plugin.

The plugin is able to read and write X-Plane datarefs as well as send commands just the way pilot pushes buttons and moves handles in the cockpit.

X-Plane Connect The X-Plane Communications Toolbox (XPC) is an open source research tool used to interact with the commercial flight simulator software X-Plane. XPC allows users to control aircraft and receive state information from aircraft simulated in X-Plane using functions written in C, C++, Java, Python or MATLAB in real time over the network. This research tool has been used to visualize flight paths, test control algorithms,

simulate an active airspace, or generate out-the-window visuals for in-house flight simulation software. [42]

This connector is a stand-alone script that is not launched through X-Plane. This gives us the option to modify the source codes and to launch it with the version of python we need. The version of

However, the XPC does not support sending commands in a way the pilot does it (pushing buttons and moving handles), but provides reading and writing datarefs.

Chapter 3

Methodology

Using the environment defined in previous chapter, we describe the methodology of learning and evaluation of the experiments.

3.1 X-Plane

In this section, we describe the process of approach and landing in the X-Plane flight simulator.

Taking into consideration the example case of the American airlines described in the motivation section, we decided to perform our experiments on the most common aircraft in the fleet, Boeing 737-800. This aircraft is one of the basic aircrafts available in X-Plane. Performing landing is also more difficult on large jets, as they react to modification in control configuration with certain delay (their weight, size and kinetic energy forbid them to make a right angle turn in place).

3.1.1 Initial state

Having the Boeing 737-800, we now need to plan the destination airport and the landing itself. As the destination airport we chose Phoenix Sky Harbor International Airport (KPHX ICAO code) in Phoenix, Arizona - specifically the runway 07L, which has 78° magnetic and 90° true heading. This runway has length of 3139 meters and is equipped with ILS, which allows the autopilot to utilize it when descending and landing. ILS will be further used as the expert control of the aircraft (lateral and vertical guidance).

As we want to simulate the most difficult parts after final approach fix, we need to start before it. That is why we we placed the aircraft into a position that is before final approach fix, in the intermediate approach (a level flight segment before final approach), 10 nautical miles from the beginning of the runway 07L, parallel with the heading of the runway (as the final approach is straight-in at airports).

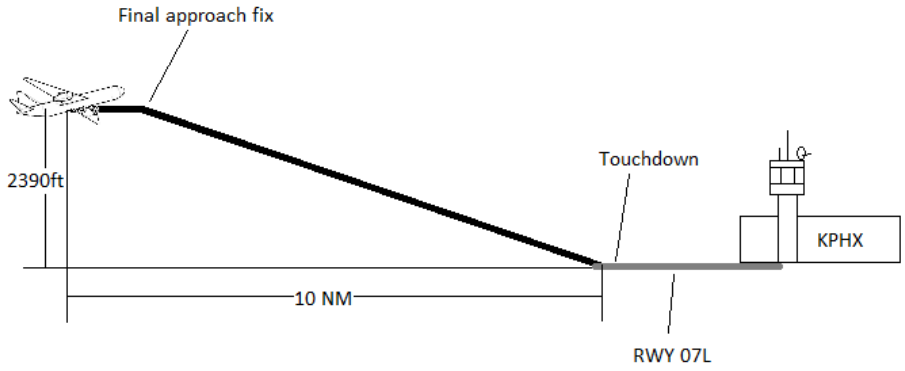


Figure 3.1: Vertical path of the aircraft

The aircraft is placed 3500ft above the sea level (which is about 2390ft above the runway) and its indicated air speed is 235kts. The position and the vertical path can be seen on the picture 3.1. The lateral path is a straight line, so no picture is necessary here.

The aircraft in the initial state does not have the landing gear out, is heading straight with no air brakes applied, nor flaps extended.

3.1.2 Approach and landing

From the initial state described, we want the aircraft to descend and land onto the runway 07L. As we want to land in the most convenient way, we also take into consideration the conditions of stabilized approach described in 1.1.2.4. As we utilize instrumental landing system during the approach for more precise control of the aircraft, the lateral and vertical guidance is solved. What we have to focus on while performing the landing as the expert (flying in the simulator), is the correct timing of extension of flaps, correct timing of pulling out the gear and maintaining the correct speed.

As the meteorological conditions set in X-Plane correspond to Visual Meteorological Conditions, we need to be stabilized 500ft above the runway. According to [43] the landing configuration is:

- Landing gear down
- Flaps set to 30 or 40 (full flaps)
- V_{REF} is 140kts for our case

As the stabilized condition says, the airspeed is not more than $V_{REF} + 20kts$ indicated speed and not less than V_{REF}. Thus, we have to keep the indicated airspeed between 140 and 160kts when below 500ft. [44, 45]

While stabilized, we land on the runway and immediately after touchdown, we apply air brakes, speed brakes and full reverse throttle to bring the aircraft to a full stop as fast as possible.

3.2 Setting the pipeline

Now that we know what our scenario is, we can incorporate it into a pipeline. As teaching the policy in real time from X-Plane would take too much time and my hardware configuration is limited, we decided to train a dynamics model that would represent the dynamics of the environment in X-Plane.

3.2.1 Dynamics model

To capture the dynamics during the approach and landing, the scenario was flown 10 times. Each of these scenarios was recorded using Python Interface plugin in X-Plane (described in 2.2.1) with certain frequency of recording. As we can see the data flow in the picture 3.2, aircraft state and corresponding action to the state are stored into a database. We do not have a single action for controlling the whole aircraft, but rather a set of sub-actions (each sub-action belongs to one handle or other aircraft control) that form a complete action.

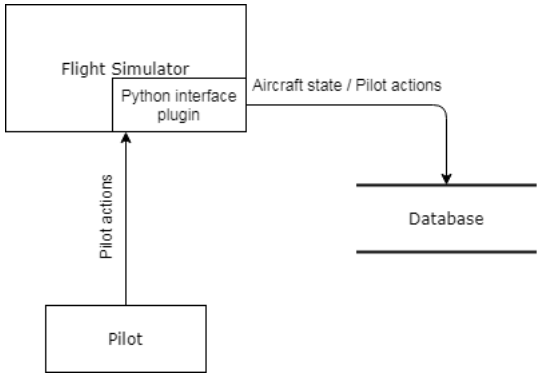


Figure 3.2: Collecting data from X-Plane

After all the flights were stored into the database, we focused on discretizing actions as training a policy with continuous actions is not as straightforward. With discrete actions we train a dynamics model that takes tuple $[state_t, action_t]$ as input and $state_{t+1}$ as output. The model is trained with whole database of flights. This process is depicted in the picture 3.3.

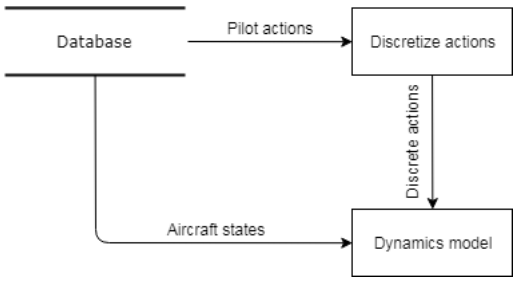


Figure 3.3: Training the dynamics model

3.2.2 Train policy

Now that we have the dynamics model trained, we need to use it to train a policy. We create an environment for reinforcement learning, similar to [46] environments, where we incorporate the dynamics model for generating a new state according to previous state and selected action when performing a step. Our custom environment includes a reward function for being in the current state.

The policy is initialized as a set of agents - for each sub-action there is one agent that evaluates what sub-action should be taken as part of the action for the current state.

The training itself is done via a script utilizing a cross entropy method, where we first execute number of sessions. Each session starts at the state defined in 3.1.1 and utilizing the trained dynamics model we generate a flight that terminates when the aircraft lands (touches the ground). In every of the states of mentioned sessions we take an action according to current policy and after we reach certain number of sessions (a batch), we select a certain percentile of best performing flights (the rewards were the highest) and use these flights to train the policy. Described process is shown in algorithm 1.

Data: X-Plane environment
Result: trained policy
initialize policy evenly;
while *criteria not met* **do**
    generate sessions;
    select best performing;
    feed the best performing to policy;
end

Algorithm 1: Training policy algorithm

The schema of training the policy can be seen in the picture 3.4.

Figure 3.4: Training the policy

3.2.3 Evaluate policy

In previous section we described training the policy and now we need to know how can we assess whether the policy is successful. As we are using the dynamics model trained with X-Plane data and not X-Plane itself, even when the average (and also maximal) reward per session grows, it does not mean that the aircraft will fly better in X-Plane. This is why we need a backward connector, that will send actions to X-Plane according to extracted current state. Schema of this process can be seen in the picture 3.5.

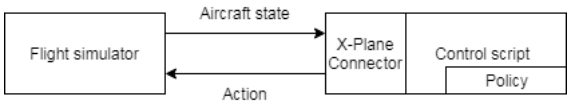


Figure 3.5: Evaluating the policy

As we see, we now utilize the X-Plane Connector. Python interface plugin was not suitable for this task as it explicitly calls python version 2 (and scikit-learn used for reinforcement learning described in previous section requires python version 3) and also serialization and deserialization lead to unexpected crashes of X-Plane. This is why I modified the code of X-Plane Connector, so that we could run it with python 3 and used it as a communication tool between the control script and X-Plane.

The algorithm 2 of evaluation of the policy is straightforward:

Data: X-Plane, policy
load trained policy;
connect to X-Plane;
set aircraft to initial state;
while *aircraft not landed* **do**
    get state;
    select action for the state;
end

Algorithm 2: Evaluating policy algorithm

After the algorithm is finished, we have the aircraft on the ground. The assessment has to be done by an expert once again.We focus on the smoothness of landing, respecting conditions of stabilized approach and accuracy of the touchdown (we do not want to land far from runway).

Chapter 4

Implementation

Describing the process does not necessarily describe the way we implement the steps. This chapter focuses on implementation details.

4.1 Gathering data

For data collection we use the Python interface plugin, which is developed by Sandy Barbour. This plugin offers registering various callback functions to certain events. One of registrable functions is flight loop callback, which is periodically called while the script in plugin is running. Utilizing this function we can collect data with a set frequency.

As we fly our scenarios with a large jet (Boeing 737-800), the speed of changes in the state of the aircraft is not rapid. That is why we chose the frequency of data collection to once per second. Knowing the frequency, we also need to know what to collect. We chose a subset of datarefs to represent the state of the aircraft (including handle and control positions) shown in the table 4.1.

We see (in the table 4.1) that there is a lot of attributes and so the reinforcement learning algorithm would be time demanding. Because of this, we chose to select only a subset of these attributes. The subset was chosen according to two facts:

1. Even though the datarefs are different, some of them have duplicate values.

2. The variance of certain datarefs is in some cases 0 or close to 0.

After taking this into consideration, we were able to limit and divide the selection into two categories shown in tables 4.2 and 4.3.

The table 4.2 contains datarefs that represent the aircraft state - the position and heading, speeds (lateral and angle as well), control surfaces position

flightmodel/position/latitude	flightmodel/controls/lail1def
flightmodel/position/longitude	flightmodel/controls/ldruddef
flightmodel/position/elevation	joystick/FC_hdng
flightmodel/position/indicated_airspeed	joystick/FC_ptch
flightmodel/position/alpha	joystick/FC_roll
flightmodel/position/beta	flightmodel2/controls/pitch_ratio
flightmodel/position/vpath	flightmodel2/controls/roll_ratio
flightmodel/position/hpath	flightmodel2/controls/heading_ratio
flightmodel/position/P	flightmodel/controls/ail_trim
flightmodel/position/Q	flightmodel/controls/rud_trim
flightmodel/position/R	flightmodel/controls/elv_trim
flightmodel/position/P_dot	cockpit2/controls/rudder_trim
flightmodel/position/Q_dot	cockpit2/controls/yoke_pitch_ratio
flightmodel/position/R_dot	cockpit2/controls/yoke_roll_ratio
flightmodel/position/theta	cockpit2/controls/yoke_heading_ratio
flightmodel/position/true_theta	cockpit2/engine/actuators/throttle_beta_rev_ratio_all
flightmodel/position/phi	cockpit2/controls/flap_ratio
flightmodel/position/true_phi	cockpit2/controls/speedbrake_ratio
flightmodel/position/psi	flightmodel/failures/onground_any
flightmodel/position/true_psi	cockpit2/controls/gear_handle_down
flightmodel/position/mag_psi	flightmodel2/gear/deploy_ratio
flightmodel/controls/sbrkrat	flightmodel2/engines/throttle_used_ratio
flightmodel/controls/flaprat	cockpit2/engine/indicators/N1_percent

Table 4.1: Datarefs extracted

flightmodel/position/latitude	flightmodel/position/Q
flightmodel/position/longitude	flightmodel/position/R
flightmodel/position/elevation	flightmodel/position/theta
flightmodel/position/indicated_airspeed	flightmodel/position/phi
flightmodel/position/alpha	flightmodel/position/mag_psi
flightmodel/position/beta	flightmodel/controls/sbrkrat
flightmodel/position/vpath	flightmodel/controls/flaprat
flightmodel/position/hpath	flightmodel2/engines/throttle_used_ratio
flightmodel/position/P	flightmodel/failures/onground_any

Table 4.2: State datarefs

flightmodel/controls/elv_trim	cockpit2/controls/flap_ratio
joystick/FC_ptch	cockpit2/controls/speedbrake_ratio
cockpit2/engine/actuators/throttle_beta_rev_ratio_all	cockpit2/controls/gear_handle_down

Table 4.3: Action datarefs

and throttle used. The table 4.3 represents the positions of handles that influence the state datarefs. Not all state datarefs are writable, but all of the action datarefs are.

Every single of the 10 approaches performed was different as we tried to capture as many possibilities of landing styles as possible. This resulted in having flights with different timespans, having 2321 samples (one sample every second) altogether. One sample is a pair of state datarefs tuple and action datarefs tuple.

4.1.1 Discretization of actions

Having continuous actions, we need to transform them into discrete. Desired output of this discretization is to have actions that contain only three values: 1, 0 and -1, that correspond to move the handle up, keep the handle as it is and move the handle down.

To achieve this, we first binned the actions to certain number of bins. The number of bins was set empirically. First we tried 10 bins, which was not sufficient for some of the datarefs as the change between the samples was more delicate and vast majority of samples had value of 0 (keep the handle as it is).

We then settled for 100 bins and tried various binning techniques:

1. Bins with equal width. Each bin has width of 1/100 of the range of the dataref. KBinsDiscretizer class from scikit-learn was used for this transformation.

2. Binning comparing two consecutive samples. If the absolute value of change between them is greater than 1/100 of the range of corresponding dataref, we consider them to belong to different bins.

To transform the bins into the three values described, we traverse each flight and set the first action to a tuple of zeros. If the next action belongs to another bin, we set the action to 1 if it is a higher bin, to -1 if it is a lower bin, else 0.

According to the results of dynamics model and reinforcement learning algorithms, the second option was used.

4.2 Dynamics model

The dynamics model should represent how a certain action changes the state we are in. The input of the model should be a tuple of $[state_t, action_t]$ and output $state_{t+1}$.

4.2.1 The model

There were various model which we tested as a dynamics model of X-Plane:

- Linear regression
- K-Nearest neighbors regression
- Decision tree regression / Random forest regression
- Neural network regression

As we see, we focus on regression because the predicted variables are not discrete and thus classifiers are useless.

Linear regression The first model utilized as a dynamics model was linear regression. This simple model was used as the baseline to asses whether other models perform reasonably. The parameters of the scikit-learn class LinearRegression, were set as follows:

- $fit\_intercept$ to True, which means that we do not assume that data we feed to the model are centered.
- $normalize$ to False, as we do not want to normalize the data before feeding them to the model. That way we may lose valuable information.

K-Nearest neighbors regression This model and its implementation in scikit-learn (KNeighborsRegressor) provides an opportunity to fine-tune the parameters. Here we focus on tuning two parameters:

- $n\_neighbors$ which represents the number of neighbors (k) which we take into consideration while predicting a new y based on input x. We chose to demonstrate how well dynamics model performs when scaling $n\_neighbors$ from 1 until 50.
- $weights$, which defines how the neighbors influence the prediction. The two basic options are 'uniform' which does not consider the distance of the neighbors and each of them influences the result equally, and 'distance' which weights the points by the inverse of their distance. In this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.

Decision tree regression The third model used as a dynamics model was decision tree regression and corresponding class DecisionTreeRegressor in scikit-learn. As well as K-Nearest neighbors, it offers various parameters for tuning:

- *criterion* is the function to measure the quality of a split. Supported criteria are 'mse' for the mean squared error, which is equal to variance reduction as feature selection criterion and minimizes the L2 loss using the mean of each terminal node, 'friedman_mse', which uses mean squared error with Friedman's improvement score for potential splits, and 'mae' for the mean absolute error, which minimizes the L1 loss using the median of each terminal node.

- *max_depth* The maximum depth of the tree. We tune this parameter from 5 until 50 while comparing the criterions between them.

Other parameters remain fixed at their default values.

Random forest regression Another model, closely connected to the decision tree model is random forest regression and its scikit class RandomForestRegressor. As all tree related parameters were tuned in the decision tree regression experiments, here we focus on one parameter:

- *n_estimators* - is the number of trees in the forest. We begin testing from 2 (as we will already have the results of one tree from decision tree regression) until 100 trees.

Other parameters will remain fixed at their defaults.

Neural network regression The last model tested for a dynamics model were neural networks, specifically a multilayer perceptron regressor (MLPRegressor class in scikit-learn). The parameters we tune here are following:

- *hidden_layer_sizes* - this parameters sets the number and size of hidden layers of multilayer perceptron. In our case, we fixed the number of layers to two and scaled the number of neurons in each of them from 10 to 50.

- *activation* represents the activation function of each neuron in hidden layer. There are four activation functions scikit-learn offers:

  1. 'identity', no-op activation, useful to implement linear bottleneck, returns $f(x) = x$.
  2. 'logistic', the logistic sigmoid function, returns $f(x) = 1/(1 + exp(-x))$.
  3. 'tanh', the hyperbolic tan function, returns $f(x) = tanh(x)$.
  4. 'relu', the rectified linear unit function, returns $f(x) = max(0, x)$

  In this thesis, we compare all the mentioned activation functions and see how they perform further.

- max_iter Maximum number of iterations. The solver iterates until convergence or this number of iterations. As some of the model configurations were not able to converge during the 200 default maximum number of iterations while performing our experiments, we grew this number to 20000.
- 'shuffle' - whether to shuffle samples in each iteration. Only used when *solver* ='sgd' or 'adam'. We set this parameter to False as we did not want to shuffle the samples.

Other parameters of MLPRegressor were left at their defaults. This also includes setting the solver to 'adam', which refers to a stochastic gradient-based optimizer proposed by Kingma, Diederik, and Jimmy Ba.

4.2.2 Training

The training itself is done in two steps:

1. Go through the first 8 flights sample by sample
   - Set x to be the pair of state with corresponding discrete action
   - Set y to be the following state
2. Feed the set of x-es and corresponding y-s to the model

This way we got a model that should represent the dynamics that are captured in the approaches we flew. We only used the first 8 approaches because we wanted the other two approaches for model evaluation.

4.2.3 Evaluation

There are two ways we need to evaluate the trained model.

1. The first is to check whether the model performs well if we try to predict only the next state. We went through all the samples of the remaining two approaches and tried to predict the next state based on the expert flown pair $[state_t, action_t]$ and then compared it to $state_{t+1}$. The distance metric here is average squared error per attribute.
2. The second way we needed to test our model, was to simulate whole flight starting at the initial state and see whether it corresponds to the actual flight. During each step, we take an action we actually took while flying and compare the state extracted from X-Plane to the one our dynamics model predicted. Each of the results contains one line which represents the average of the two test approach errors. The distance metric used is squared difference between each of the attributes of a state added up together.

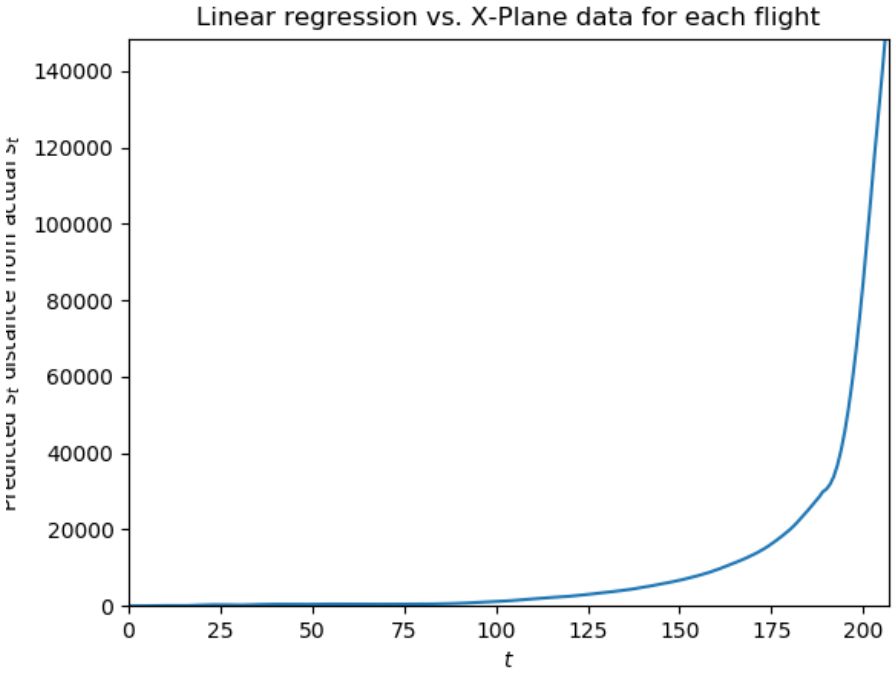


Figure 4.1: Linear regression dynamics model evaluation

Linear regression There were no parameter tunings performed for linear regression model.
As to the two methods of evaluation:

1. The average squared error per attribute was 0.0118.

2. As we see from the picture 4.1, the quality of linear regression model is acceptable for the first 100 steps, where it reaches a maximum error of 1173.59. However, after this period the model becomes unreliable. As our expert flown descents contain approximately 200 steps, this model being unreliable after 100 does not prove to be useable any further. The error after 200 steps grew up to 140000.

K-Nearest neighbors regression As for the L-Nearest neighbors regression model, we tuned K and the method of taking neighbors into consideration. The results of this experiment can be seen in the picture 4.2.

As we see from the picture 4.2, the average squared error per attribute rises as the parameter K rises. This was expected and thus we set our K to 7 not wanting to have model that is over-fitted nor under-fitted (the minimum of the graph). As for the weights of neighbors, 'distance' (weighting closer

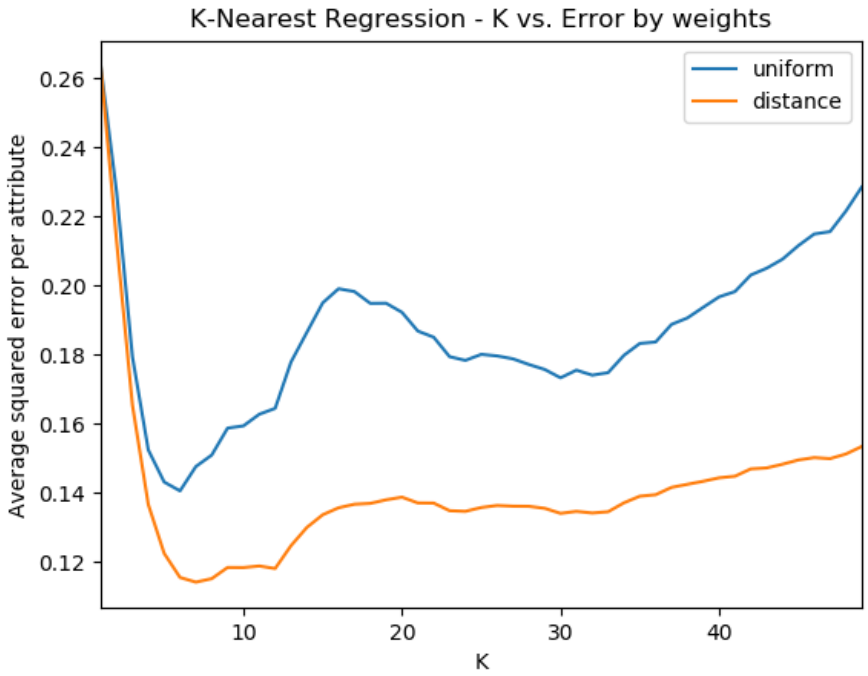


Figure 4.2: Tuning k for K-NN Regressor

neighbors more) performs better than 'uniform' and so we choose this weight method.

According to described experiment and utilizing described configuration of the model, we perform the two methods of evaluation:

1. The average squared error per attribute was 0.1141. This result is worse than linear regression.

2. As we see from the picture 4.3, the quality of K-NN regression model is visibly better than linear regression model and until the step 75 it appears as nearly ideal dynamics model, as the error is always below 30. The final error is 5366.5.

As seen from the errors described, K-NN Regressor may have worse average error per attribute, but in general it depicts the dynamics of X-Plane better than linear regression.

Decision tree regression Third model used as the dynamics model was Decision tree regression. Here we focused on tuning two parameters and according to this the experiment was developed. As we see from the picture 4.4,

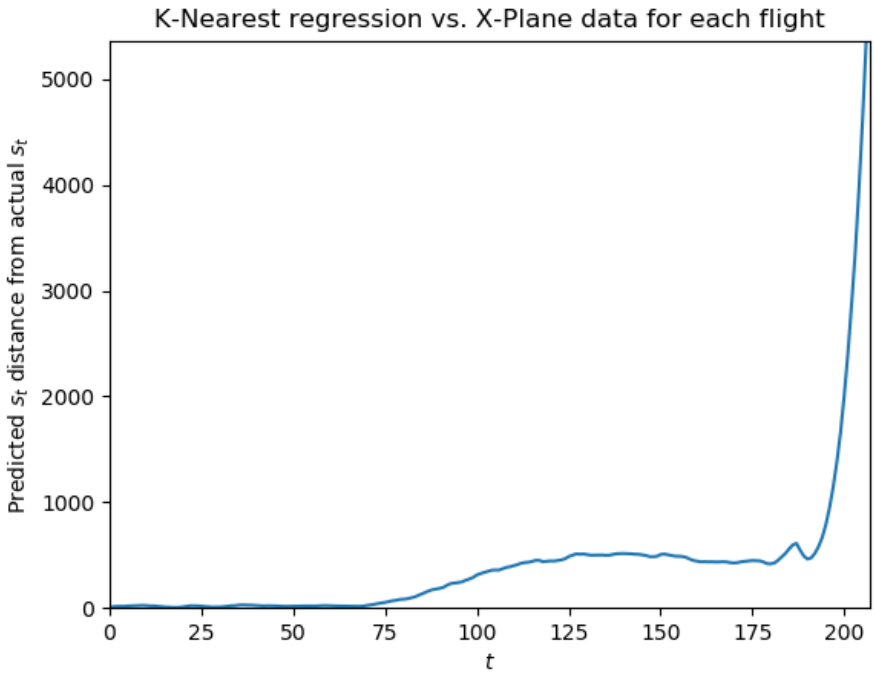


Figure 4.3: K-NN regression dynamics model evaluation

the criterion for splitting the node influences the average squared error per attribute greatly. We see that in some cases, the worst criterion result has more than three times the error of the best criterion (MSE). The criterion MAE shows results similar to MSE, but the training time of MAE criterion regressor was more than 20 times slower than training MSE criterion regressor.

We also see, that the parameter of maximum depth of tree does not influence the error significantly if it is greater than 10. According to this experiment we set the criterion to MSE and maximum depth to 15.

Considering our two defined metrics:

1. The average squared error per attribute was 1.0785. This was the worst result amongst the tested dynamic models.

2. As we see from the picture 4.5, it behaves similarly to K-NN regressor. However, here we see that the model is highly reliable until 100 steps where it reached an error of 519.2 and then the error rapidly rises. It reaches a maximum error of 21380.9.

We see, that even though the decision tree regressor has the worst average squared error per attribute, it performs exceptionally well until a threshold

Figure 4.4: Tuning maximum depth for decision tree regression

of 100 steps which does not satisfy our needs, because our flights last around 200 steps.

Random forest regression Taking into consideration that one tree performs well, we decided to experiment with an ensemble method incorporating more trees - random forest regression. As we already know what are the optimal parameters of a decision tree, we just have to tune the number of estimators.

As we see from the picture 4.6, the average squared error decreases rapidly until 10 trees and after 20 trees it stays constant at value of approximately 0.25. We set the parameter to 20, not wanting to overfit the model.

1. The average squared error per attribute was 0.2622, which was a significant improvement compared to decision tree itself. This error was not the lowest out of all regressors, but it does not disqualify random forest regression as the dynamics model.

2. As we can see from the picture 4.7, the random forest regression performs extraordinarily. The error always stays below 400 until the step 190 and then it rapidly rises. As we said, the flights take around 200 steps, which shows the model is very likely to predict whole flight very precisely.

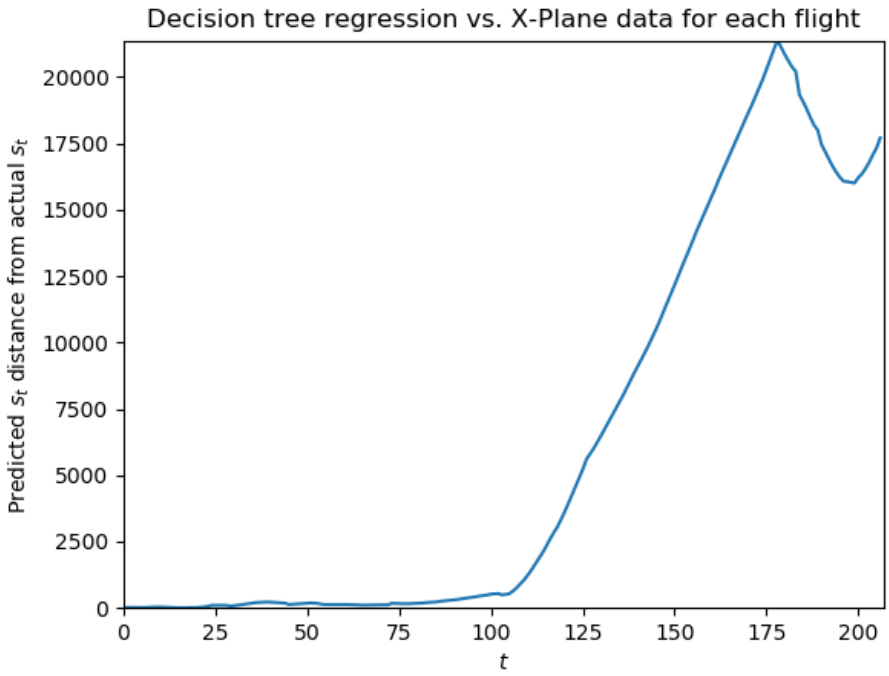


Figure 4.5: Decision tree regression dynamics model evaluation

The random forest regression model takes the decision tree regression to a whole new level as can be seen from our error metrics. This is why we chose to replace the decision tree in further experiments with random forest.

Neural network regression The last model tested as a dynamics model of X-Plane were neural networks. Here we tested how various activation functions and the size of hidden layers affect the error.

The results of the experiment can be seen in the picture 4.8. We see that activation function 'relu' and 'identity' perform much better than 'logistic' and 'tanh' activation functions. This is why we chose to show the performance of the best two activation functions in detail.

As can be seen from the picture 4.9, the two activation functions perform similarly when it comes to average squared error per attribute. This is why we chose the size of hidden layer to 25 neurons and tried both of them in our two step evaluation method of dynamics model:

1. The average squared error per attribute for 'relu' activation function neural network regressor was 0.2589 and for 'identity' activation function 0.129. We see that the 'identity' activation function performs better here, but from experience with previous regression models we know that

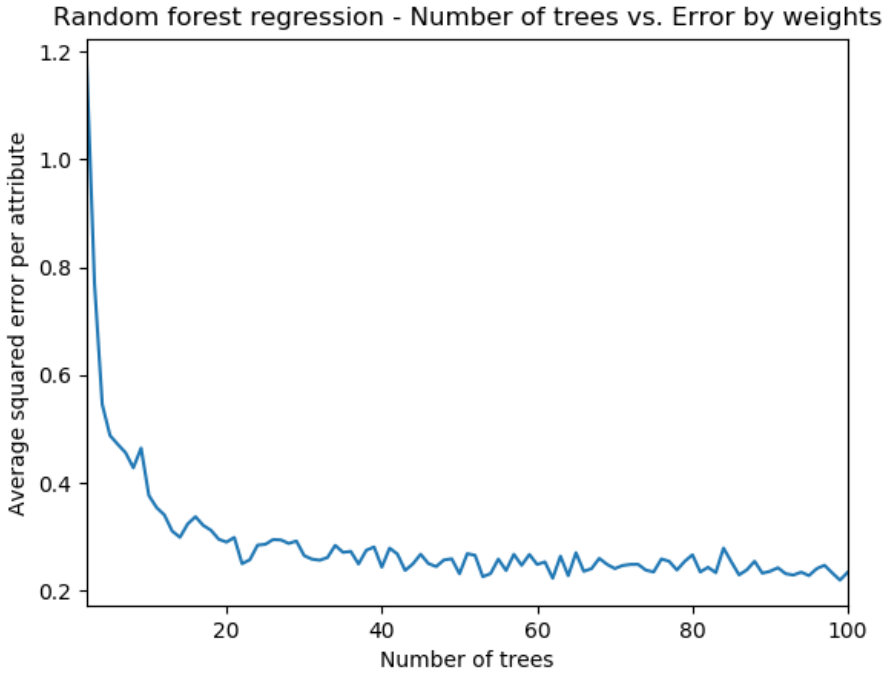


Figure 4.6: Tuning number of trees in random forest

even if this error is higher, the second error measure can show that the dynamics model is of high quality.

2. However, when we look at the results for whole flights shown in the picture 4.10, we see that the 'relu' neural network performs better. The error of 'relu' is higher at first (50 steps), but then the accuracy of 'identity' activation function regressor rises rapidly and at 175 steps the 'relu' neural network error is 602.87 and 'identity' error 2536.77. The error then rises to 3805.72 for 'relu' and to 10559.5 for 'identity'. During training experiments we experienced, that 'identity' activation function dynamics model performed much worse, resulting in errors as high as 10^{50}. The 'relu' dynamics model on the other hand always had errors of reasonable range.

In general, we see that the best performing dynamics model according to our metrics is ambiguous. For the first 75 steps, the best performing model is K-NN regressor alongside with Random forest regressor, but after that 'relu' activation function neural network performs better than the mentioned K-NN regressor. The random tree regressor appears to be the best for our task, as it has the smallest overall error.

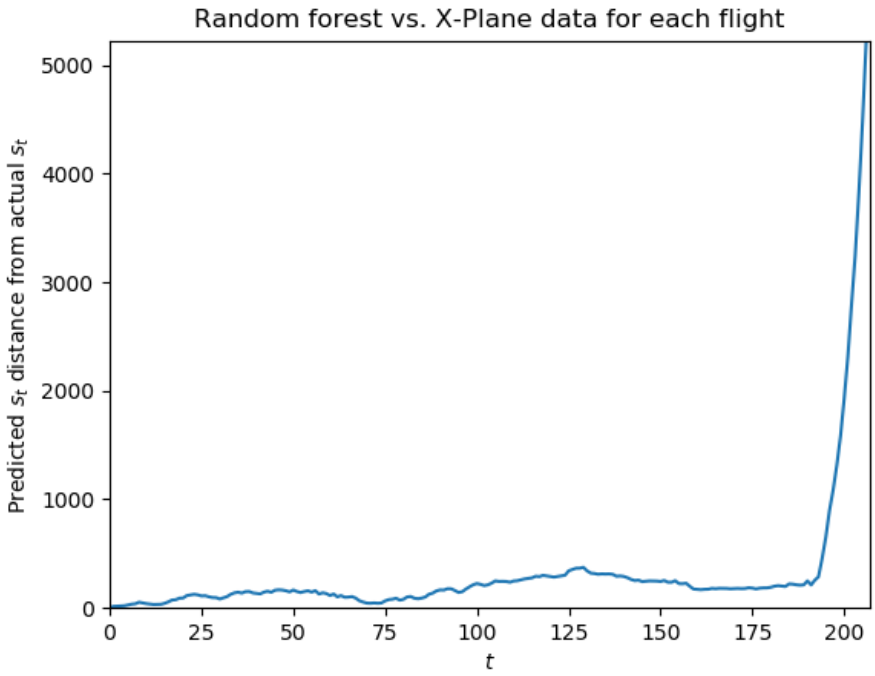


Figure 4.7: Random forest regression dynamics model evaluation

We will further test these models by training policies based on them, excluding linear regression model as it performed poorly and any policies trained on this model resulting in acceptable behavior could be considered random.

4.3 Policy

This section will describe the process of learning and evaluation of policy based on the dynamics models trained.

4.3.1 Environment

The environment is inspired by [46]. We provide a similar interface that openai environments provide:

- Constructor sets up the environment - the initial state of the environment to the initial state defined in 3.1.1, the model according to which we generate the next state to the models trained in previous section. This means we have 4 different environments corresponding to:
  1. K-NN regression model
  2. Random forest model

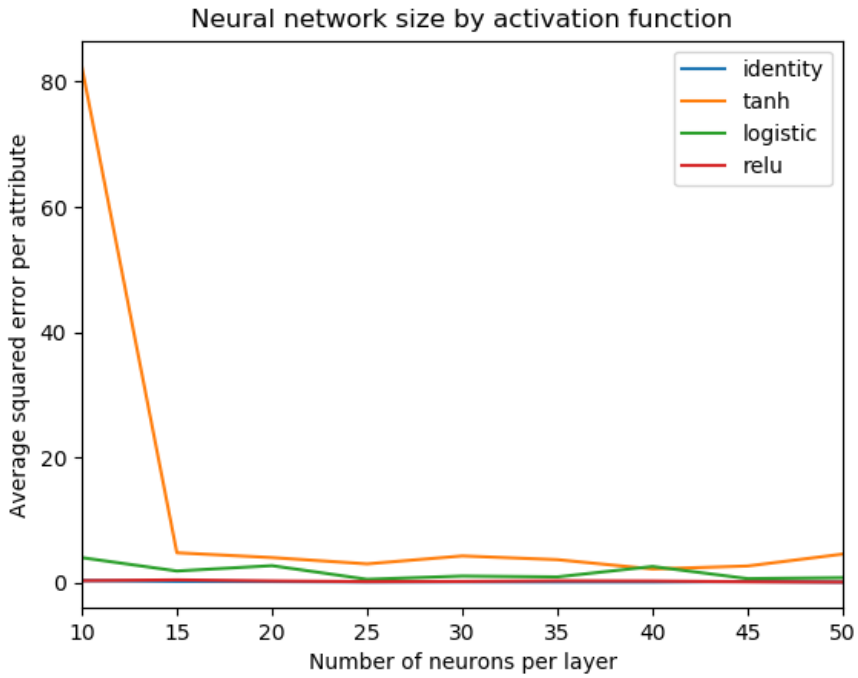


Figure 4.8: Tuning hidden layer size for neural network regression

3. Neural net regression model with 'relu' activation function
4. Neural net regression model with 'identity' activation function

- Reset resets the environment to the initial state and does not alter the model in any way.
- Step is a method that provides new generated state based on the previous state which the environment keeps and action taken. It also computes reward for being in the new state and information whether the current episode has ended or not. We consider the episode to be over when the aircraft has landed.

Reward function All the improvement of the policy we want to train is driven by the reward function. We tried two different state reward functions in this thesis and compared them:

1. Reward function based on pilot good practices. It rewards the state if:
   - The aircraft is slowing down or keeps constant speed compared to the previous state.
   - The aircraft is descending.

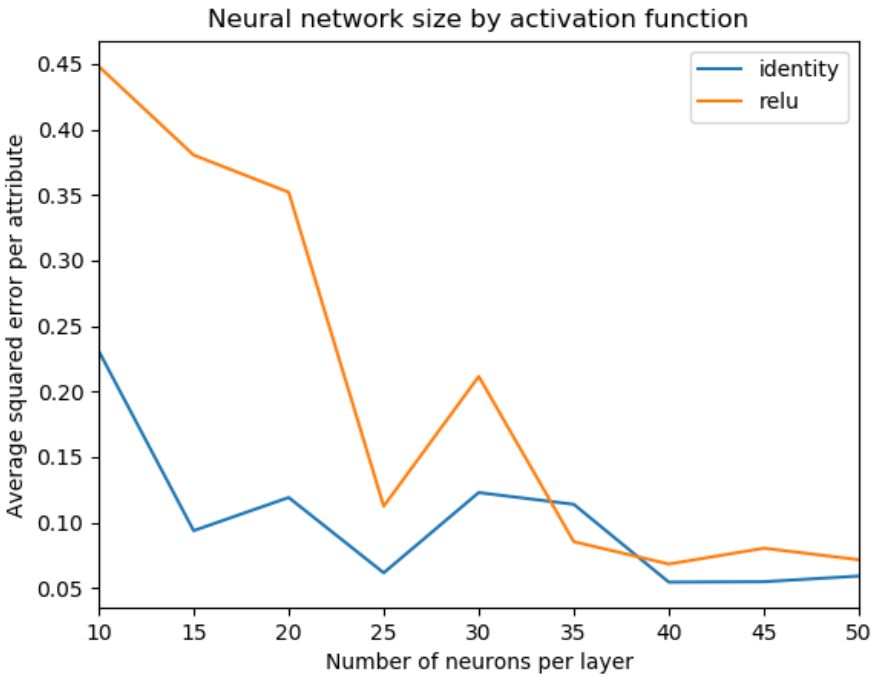


Figure 4.9: Tuning hidden layer size for 'relu' and 'identity'

- The flight path angle is greater than -4 degrees. As the optimal angle for approach and landing is 3 degrees and anything less steep satisfies the conditions of stabilized approach, we wanted to limit the angle with certain margin.

- The landing is smooth and accurate. We reward it if the indicated airspeed is below 160kts, the heading is parallel to the heading of the runway, the position of touchdown is the beginning of the runway and the angle of attack indicates that the nose is up (aircraft should land on the rear wheels).

2. Reward function aiming to learn to fly the trajectory we flew during example flights. The reward is based on the distance between the current state and the closest state from the sample states from the database. Every step we take, we compare the generated state against the states representing last two flights (9 and 10, those which we did not use to train the dynamics model) from the database using accumulated squared error per attribute as distance metric. The negative value of this distance is considered the reward (as we try to maximize the reward).

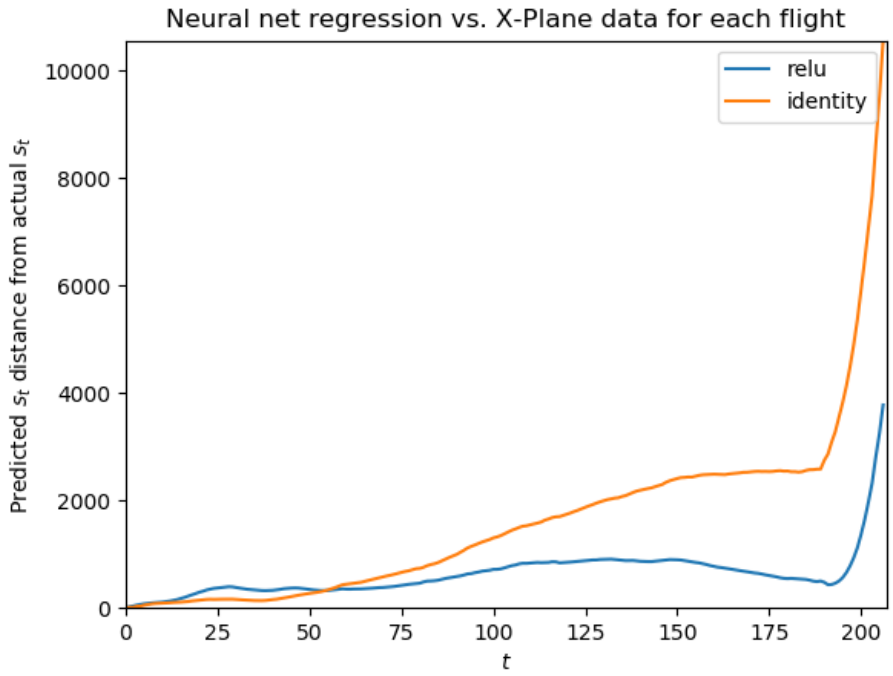


Figure 4.10: Neural network regression dynamics model evaluation

4.3.2 Training

Using the environment described, we perform the training. The training itself is done utilizing the cross entropy method of reinforcement learning.

First, we generate flight sessions. We generate 100 sessions, which results in states batch, actions batch and total reward per session. To generate a session, it means to reset the environment to the initial state and perform actions according to the agent each step until the aircraft has landed (or until we reach maximum of 250 steps which). According to total rewards, we select a certain percentile of sessions (we settled for a percentile of 80) and set a threshold that will distinguish whether the session will be used for training or not. Next, we go through all the sessions and according to the mentioned threshold either feed the states batch and corresponding action batch to the agent, or not. This is one episode of training.

Agent First, we have to define what our agent is. As we have 6 elements to control (shown in table 4.3), we decided to introduce set of 6 sub-action agents, where every agent is responsible for a single dataref. The input of an agent is always a whole state (not a subset of it) and output is $\{-1, 0, 1\}$ or $\{0, 1\}$, as it does not make sense to move certain handles back (for example

once gear is out, it does not make sense to retract it again).

Every single of these agents had to be a classifier as we have discrete output variables. As our learning will be iterative and we expect our agent to get better every run, we had to focus on models that are able to have a warm start (they reuse the solution of the previous call to fit as initialization) to ensure that we do not delete our progress each time we run the algorithm. As we plan to utilize cross entropy method for reinforcement learning, we did a research of suitable classifiers. Every of the implementations researched used the same classifier - MLPClassifier with following configuration: They had two hidden layers with various number of neurons per layer. The 'tanh' activation function was used (the hyperbolic tan function, returns $f(x) = tanh(x)$) and solver 'adam'. As we perform batch training and we do not want to start from the beginning each fitting, the parameter *warm_start* was set to true, which keeps the progress between the calls of method fit. Also, the *max_iter* parameter was set to 1. All the other parameters were left to default values.

We took the exact same approach, but tuned the size of the hidden layers and watched how successful are these agents using trained dynamics models as their environment.

4.3.3 Experiments

Tuning the size of the hidden layers, we took every of the dynamics models previously trained and looked onto how the average reward of the policy behaves each training episode. The training is done as described in 4.3.2. As some of the agents in this section did not perform well because of limited data and increasing size of hidden layers, the results did not seem reliable and thus we omitted the corresponding lines in following graphs.

Original best practices reward function First, we utilized the best practices reward function and trained three sets of agents on every dynamics model. The sizes of hidden layer were (per layer): 5, 10, 20.

Our goal was to see whether we are able to see the policy performing better every training episode until the 100th episode of training.

All the training mentioned below was time consuming and we tried to make it more efficient using multi-threading implemented in Python.

As we see from the picture 4.11, we can see progress while training the policy based on the K-NN Regression dynamics model. During these 100 episodes, the agent consisting of sub-agents with hidden layers of size 5 neurons performed the most successfully, reaching average reward per episode of approximately 510 points. However, we can see that the trend is more constant than rising rapidly (although there is a relatively big jump in the beginning of the training). The neural network with 10 neurons in each hidden layer performed worse than the 5 one, but we see a clear rising trend in the reward - the policy is able to learn. Lastly, the neural network with 20 neurons in

Figure 4.11: K-NN dynamics model - Policy trained with the best practices reward function

each hidden layer performed with a great variance and thus the conclusion after 100 training episodes is that we either have an incorrect reward function, the dynamics model is not accurate and so it does not make sense to evaluate the reward function as it requires the state to be represented as accurately as possible.

Looking onto the picture 4.12, we see that all the policies have a similar rising trend. As the random forest dynamics model was considered the most reliable according to our metrics, even though the average reward is smaller compared to the K-NN dynamics model trained policy, we can say that this model is able to teach a policy.

Although there is a relatively big variance in the performance of the 10 and 20 neurons agent policy and the results seemed unreliable and thus omitted, the 5 neurons agent policy rises clearly and rapidly, without big variance (seen in the picture 4.13). We can train a policy using this combination of dynamics model and reward function as well.

The last dynamics model tested was identity activation function neural network. The result of this experiment can be seen in the picture 4.14. There is clearly no rising trend in the first 100 training episodes and so we can say that this combination of dynamics model and reward function does not lead

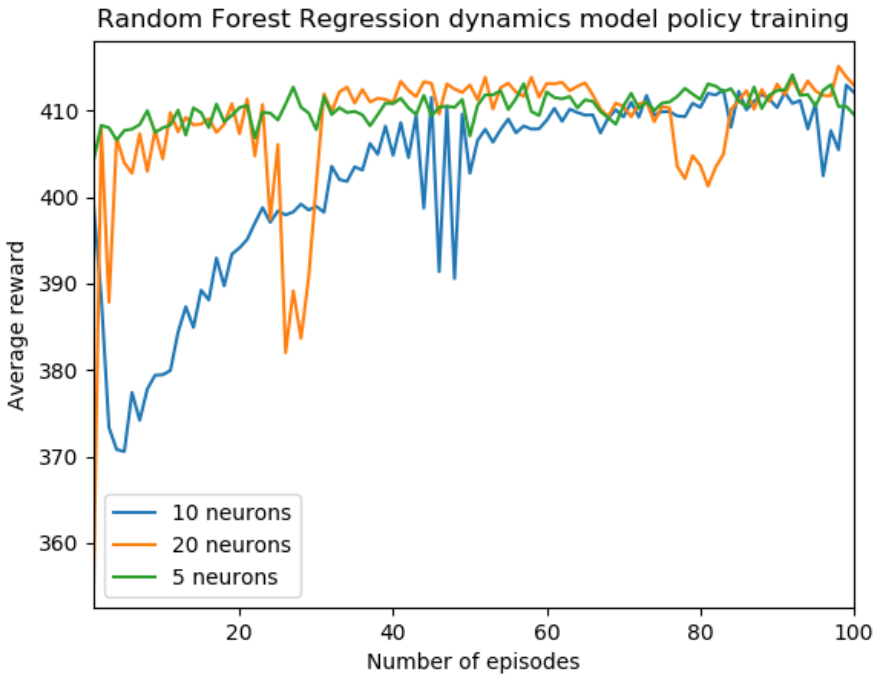


Figure 4.12: Random forest dynamics model - Policy trained with the best practices reward function

to any valid results (the 20 neurons agent policy appeared invalid because of too great peaks).

Comparing all the trained policies, we see various results and that only relu dynamics model trained policy achieved results above 700 points of reward. This is caused by the fact that only this policy and dynamics model combination was able to land during the 250 steps (expert flown flights were approximately 200 steps long, so this should be an acceptable margin). The spikes in average reward are caused by the variable ratio of landed and not landed sessions.

Upgraded best practices reward function To increase the number of landed cases, our first idea was to enhance the reward function. In the original reward function, the aircraft could just simply maintain a very low descent angle and slow down imperceptibly and it would still be rewarded even though it has already passed the runway or it is not even going in the direction of runway. The second case where this policy limps is not penalizing if the aircraft speeds up or climbs.

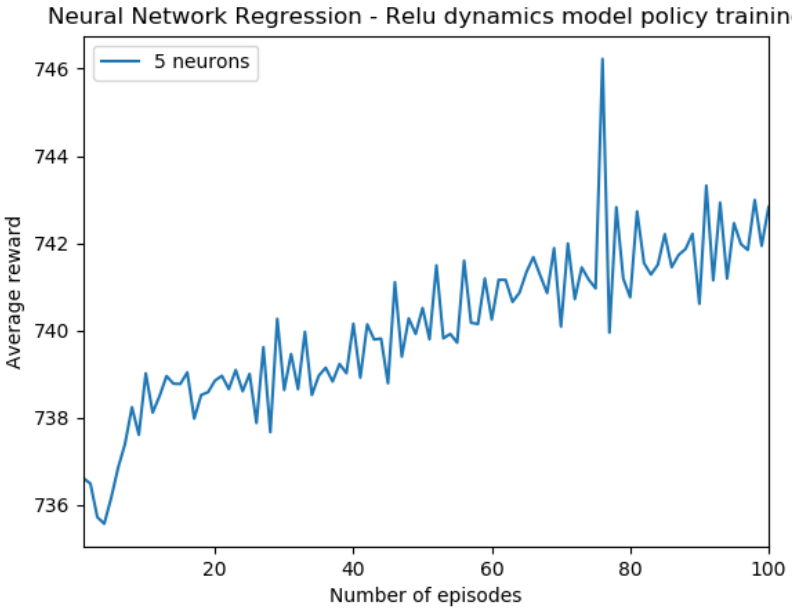


Figure 4.13: NN Relu dynamics model - Policy trained with the best practices reward function

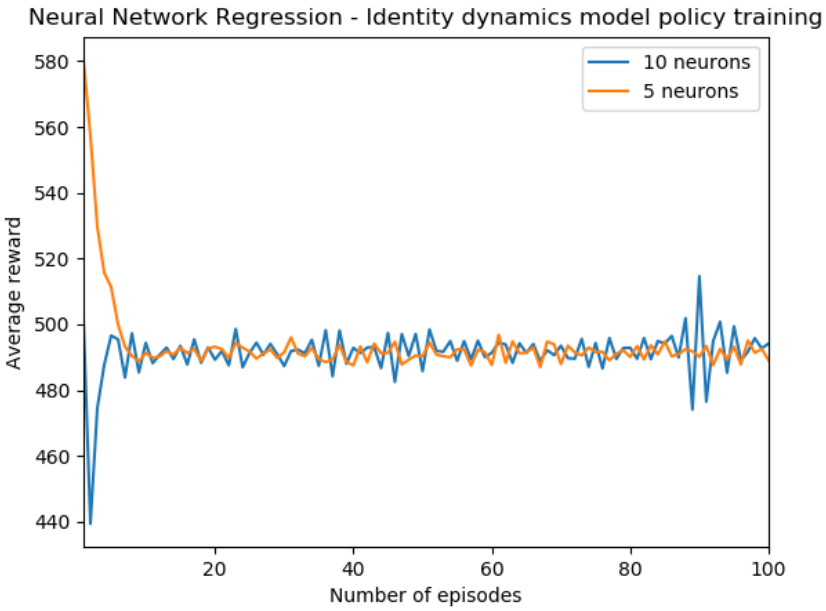


Figure 4.14: NN Identity dynamics model - Policy trained with the best practices reward function

This is why we introduced following changes:

- For the slowing down we introduce a lower boundary of 140kts (according to stabilized approach conditions).
- If we do not slow down or descend, we penalize the current state.
- If we are getting closer to runway (compared to the previous state), we reward the state, else penalize it.

We performed the same experiment for new reward function, but also introducing a greater variety of hidden layer sizes (now 5, 10, 20, 40, 60, 80, 100 neurons per hidden layer) to see whether the neural networks were not just too small to capture the behavior.

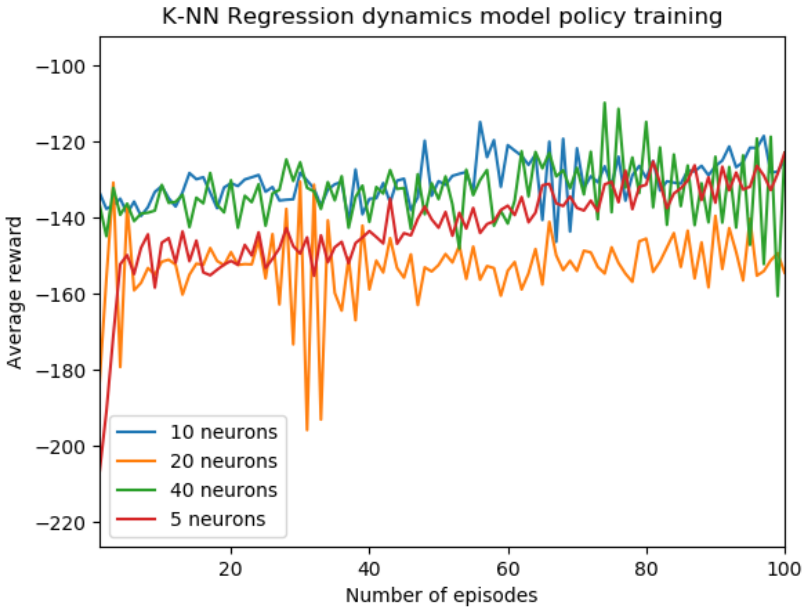
From the pictures 4.15, we see a rising trend in almost all cases of network sizes. Larger networks tend to have bigger spikes, which is most probably caused by increased bias with increasing the network size. This could be solved with more training episodes. Compared to the original best practices reward function, we see a bigger rising trend using the enhanced reward function.

As we see from the pictures 4.16, policies trained on random forest dynamics model using our enhanced reward function have larger peaks than policy trained with the original and also the rising trend is not that obvious. Also, we omitted the 80 neurons policy because of not showing reliable results. However, the average reward of the 10 neurons policy appears to be rising even though there are certain spikes.

The results of neural network dynamics model with relu activation function are shown in the pictures 4.17. There is no improvement when introducing the new reward function in the first 100 training episodes. However, there is no visible rising trend, which could also mean that the policy is not able to learn using the combination of neural networks dynamics model with relu activation function with our enhanced reward function. The bigger the neural network was, the bigger spikes it achieved, thus making the result unreliable, which caused all of the policies with agents of size above 10 neurons to appear unreliable.

The last dynamics model was neural networks with identity activation function. From the pictures 4.18, we see that enhancing the reward function changed nothing as well - there is no rising trend (which could of course change in the further training episodes). All the omitted lines (corresponding to policies with agents of 40, 60 and 80 neurons per layer performed poorly).

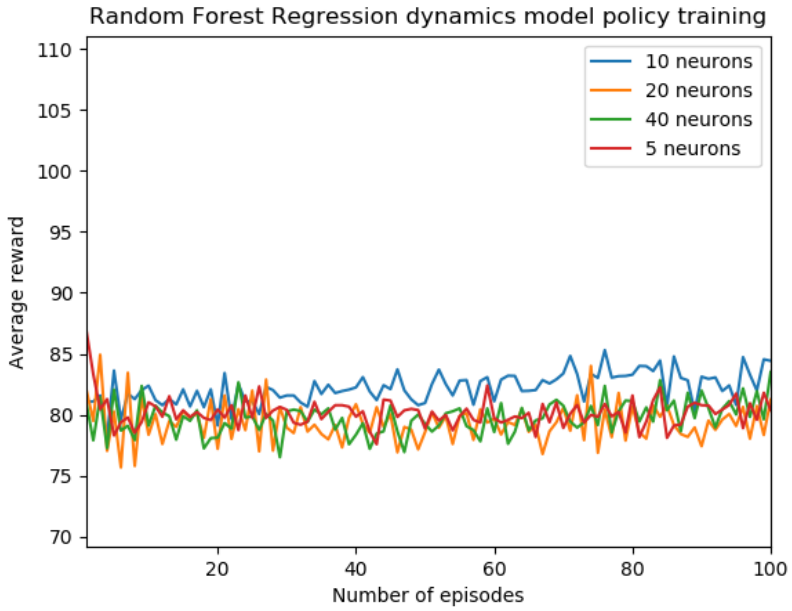
Overall, adding more neurons in hidden layer proved useless as it only prolonged the training and the results were worse than those when using smaller networks. On the other hand, modifying the reward function seems like it could lead to better policy trained. Experiments with best practices reward functions using various dynamics models and agent sizes were highly time demanding. Performing 100 training episodes for a single dynamics model



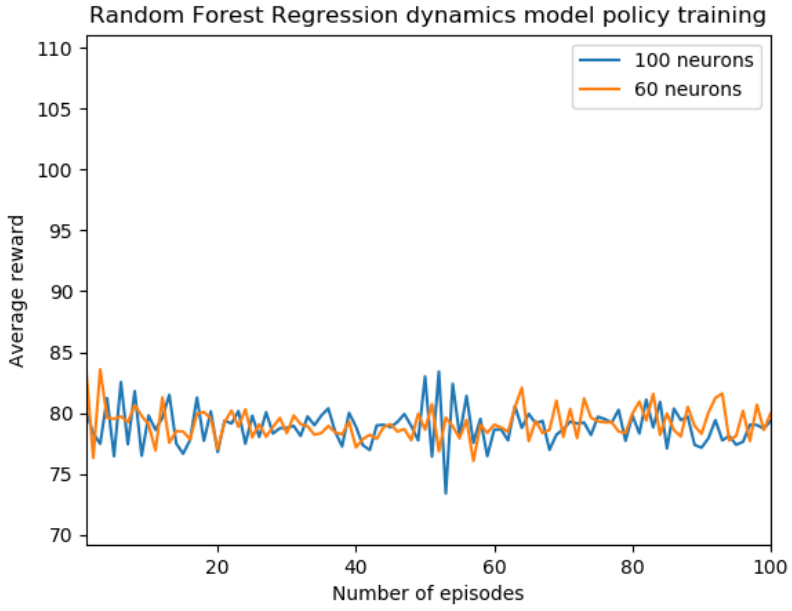
(a) 5 - 40 neurons

(b) 60 - 100 neurons

Figure 4.15: K-NN dynamics model - Policy trained with the upgraded best practices reward function

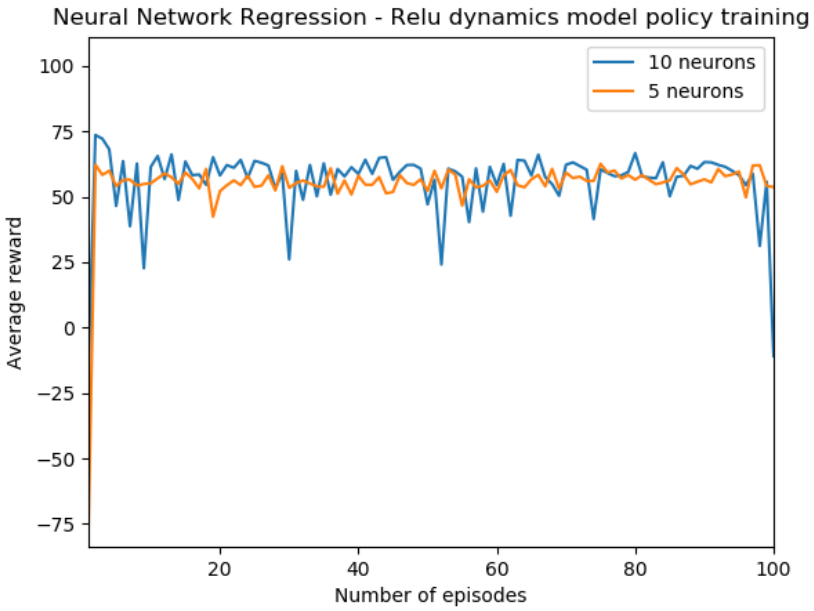


(a) 5 - 40 neurons



(b) 60 - 100 neurons

Figure 4.16: Random forest dynamics model - Policy trained with the upgraded best practices reward function



(a) 5 - 40 neurons

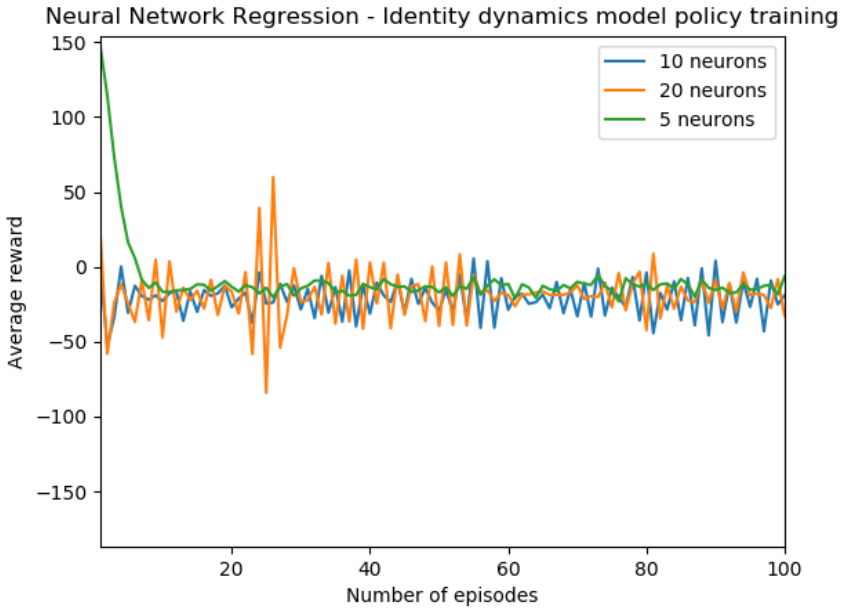
Figure 4.17: NN Relu dynamics model - Policy trained with the upgraded best practices reward function

with a single reward function using various sizes of agent took up to 24 hours of training on our hardware setup.

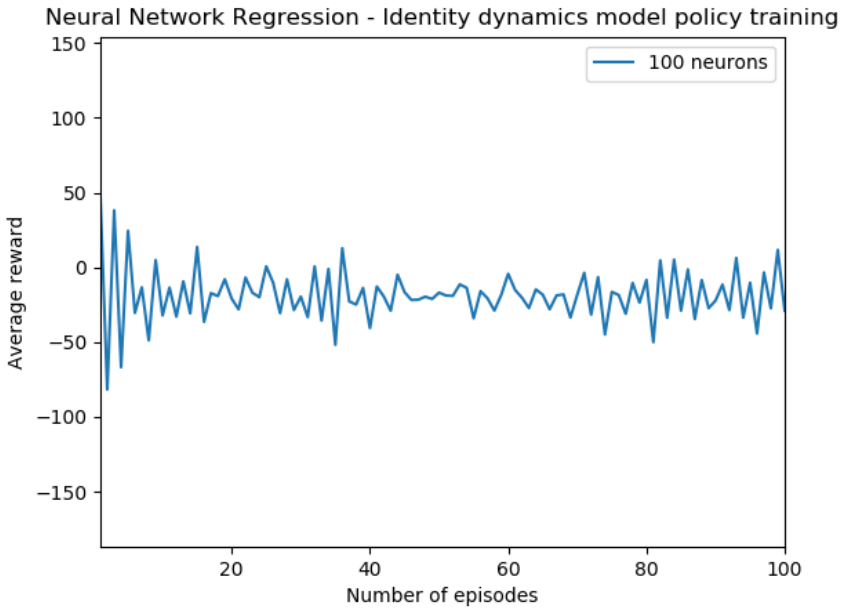
Distance reward function The second type of reward function we proposed previously was the distance reward function. The reward for the current state is the negative value of distance from the closest state of the last two flights (9 and 10, those not used for training of dynamics model) from the database. The distance metric used is squared difference per attribute added up. This reward function is far more complex than the best practices reward functions and thus we expected more variance in results.

As we see from the pictures 4.19, the growth of average rewards for policies trained using the K-NN dynamics model and distance reward function was not comparable to the best practices reward functions. We clearly see there are relatively big peaks in both directions, but te overall trend is constant and thus this combination does not seem like leading to teaching a valid policy (according to the first 100 training episodes).

From the pictures 4.20 we see that the rewards are overall higher than the ones gained by policies trained on K-NN dynamics model. However, the growth of the average reward per session is not visible here either. The average reward of -8000 per session (which sometimes peaked to -5500) shows that

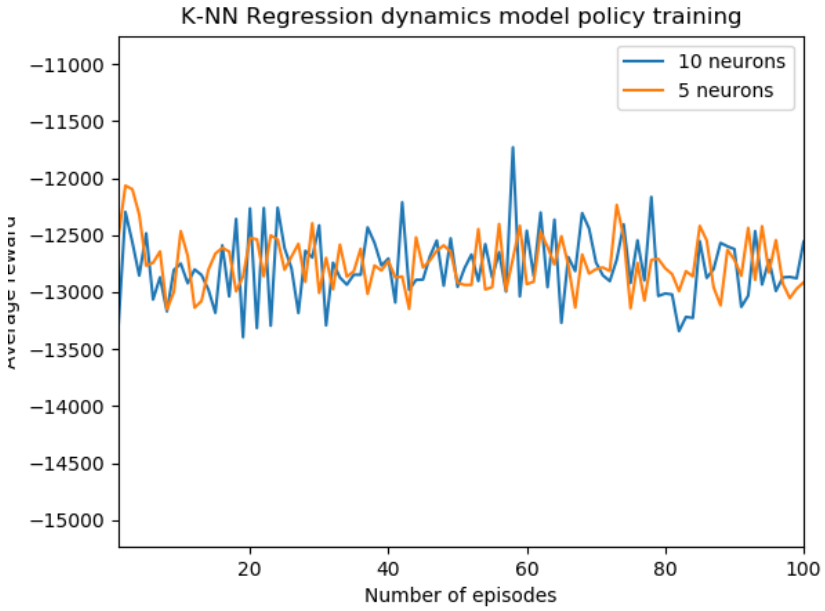


(a) 5 - 40 neurons

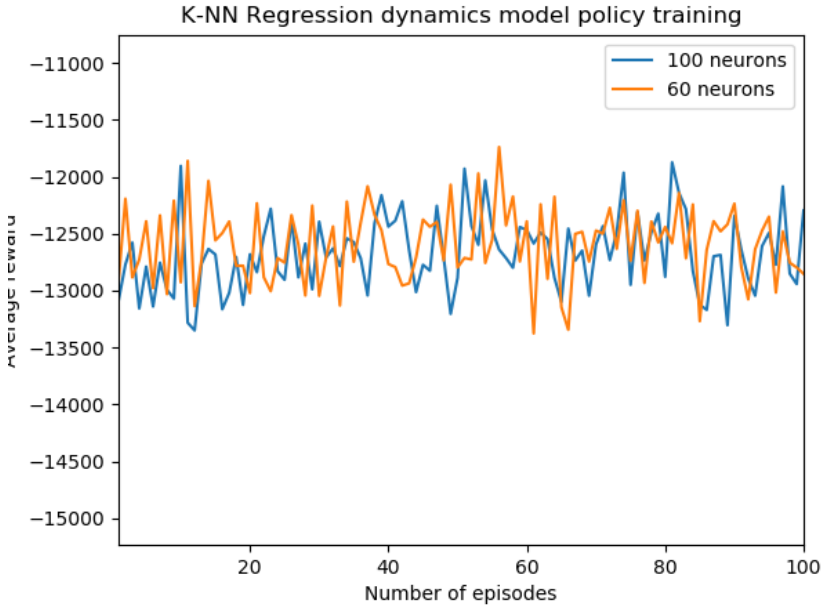


(b) 60 - 100 neurons

Figure 4.18: NN Identity dynamics model - Policy trained with the upgraded best practices reward function

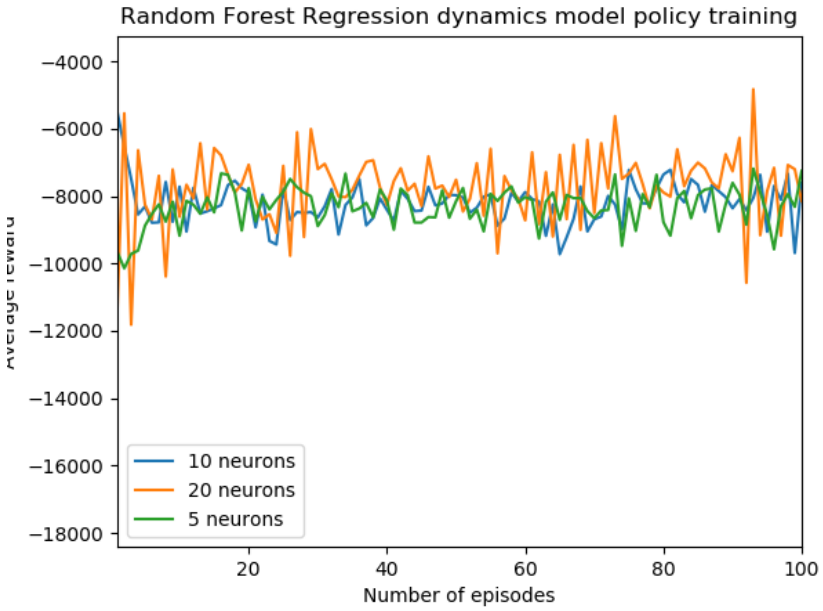


(a) 5 - 40 neurons



(b) 60 - 100 neurons

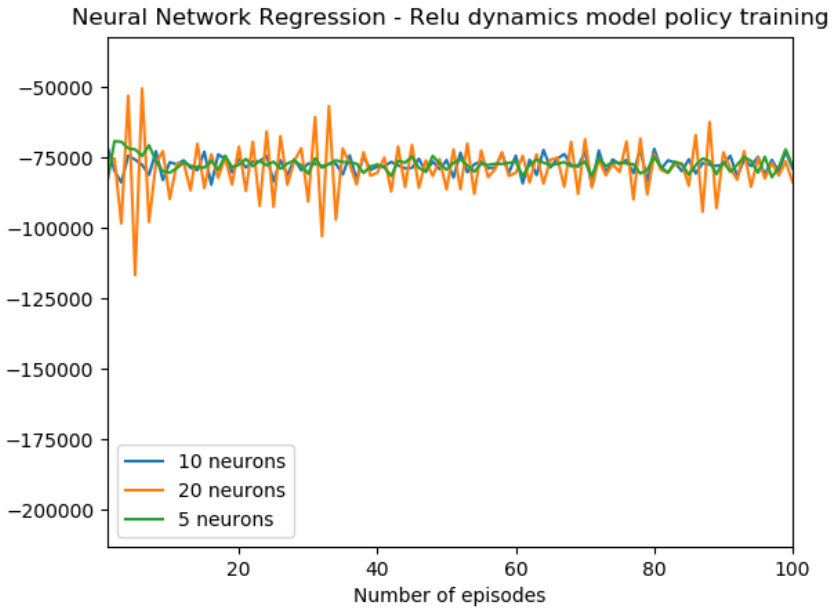
Figure 4.19: K-NN dynamics model - Policy trained with the distance reward function



(a) 5 - 40 neurons

(b) 60 - 100 neurons

Figure 4.20: Random forest dynamics model - Policy trained with the distance reward function



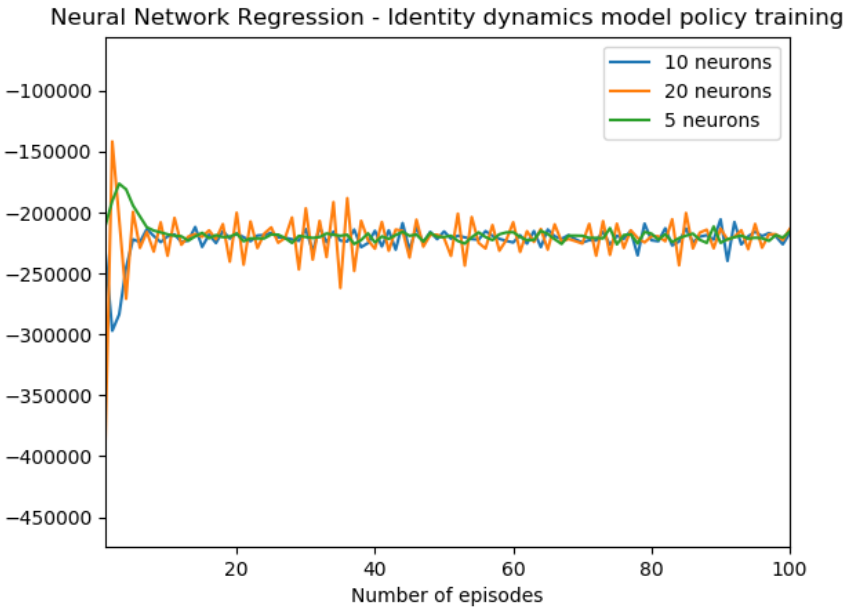
(a) 5 - 40 neurons

Figure 4.21: NN Relu dynamics model - Policy trained with the distance reward function

average distance between the predicted state and reference states (flights 9 and 10) was approximately 40, which means that average squared error per attribute here is roughly 2.2. This shows, that even though we do not see a growing trend in the rewards, the policy may be utilized in X-Plane itself. Some peaks using agents with more neurons in hidden layer reached average reward peaks of -3800.

Third dynamics model were neural networks with relu activation function. The results of distance reward function trained policy with mentioned dynamics model (shown in the pictures 4.21) had similar trend as the one trained with best practices reward function. The average reward is worse than the random forest dynamics model and we can say that during the first 100 episodes of training policies trained with combination of neural networks with relu activation function and distance reward function did not reach satisfactory results. All networks with sizes above 20 neurons provided unreliable results and so are omitted.

The last combination was neural networks with identity activation function dynamics model with distance reward function. The results can be seen in the pictures 4.22 and we see similar trend as using relu activation function, but the results are three times worse. This shows that the combination did not produce a valid policy in the first 100 training episodes. All networks with



(a) 5 - 40 neurons

Figure 4.22: NN Identity dynamics model - Policy trained with the distance reward function

sizes above 20 neurons provided unreliable results and so are omitted.

Note: Training distance reward policies was even more time demanding than utilizing best practices reward function.

4.3.4 Experiments conclusion

As we saw, some of the combinations of dynamics models with various reward functions showed either interesting results, or the trends seemed to be promising enough to look at them more closely. As we mentioned, we omitted the most unreliable results. These unreliable results were mostly the policies with bigger neural networks as agents (increased size of neural network increased the bias as well).

Best practices reward function Although finding a hole in the original best practices reward function, the results and also the trend of the neural networks with relu activation function dynamics model showed exceptional results (compared to other models) and the average reward per session was high enough that at least some percent of the sessions were able to land successfully. To be specific, we will focus solely on the agents with 5 neurons in hidden layer as it showed the steepest growth.

After upgrading the best practices reward function the average rewards fell as we introduced penalizing. Also, it brought greater peaks in both directions, which made the observation of growing trend more difficult. Here the best performing dynamics model was the random forest regressor again with a small neural network as a sub-agent (10 neurons in each hidden layer).

Distance reward function For the distance reward function, there were various results. There was no obvious growth in any of the cases, but as we use the distance reward function, we can assess how much does the aircraft fly according to defined trajectory. We reached average reward peaks of -3800 with random forest regression dynamics model and agent with 80 neurons in hidden layer, which seems like it copies the desired trajectory very well and as we remember from the previous section, the random forest regression model was considered the most reliable. This is why we selected this combination for further evaluation.

4.3.5 Training the elite agents

As we described, we selected three agents which appeared to be perspective and gave them 1000 training episodes to see whether they produce valid policies. These policies were further used for evaluation with X-Plane itself. The reward results are shown below.

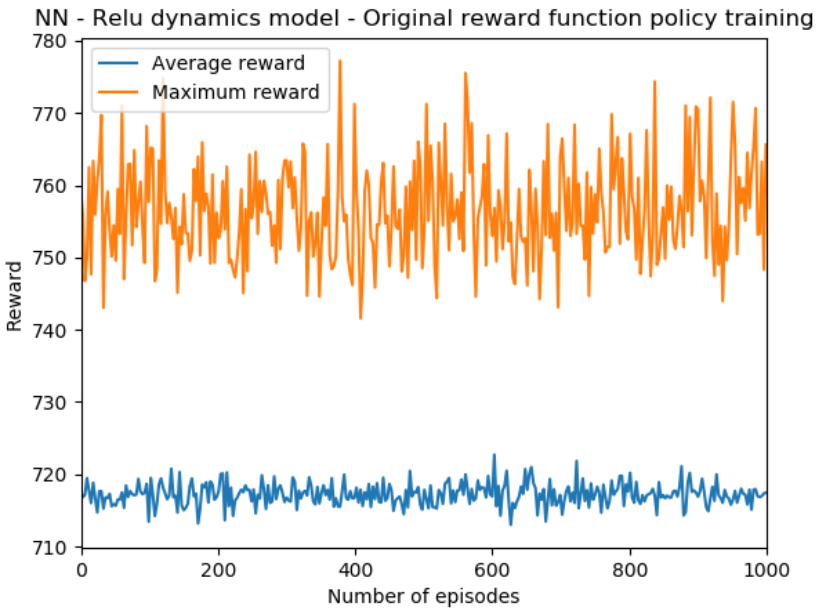


Figure 4.23: NN - Relu dynamics model - Original reward function policy training

As the picture 4.23 shows, the combination of neural network with relu activation function dynamics model and original best practices reward function using agents with two hidden layers of 5 neurons did not rise rapidly after the first 100 training episodes. Average reward per session was always approximately 717 points and varied little, but the maximum reward per session varied a lot more (up to 778.3). The maximum reward shows, that some of the flights were indeed able to land.

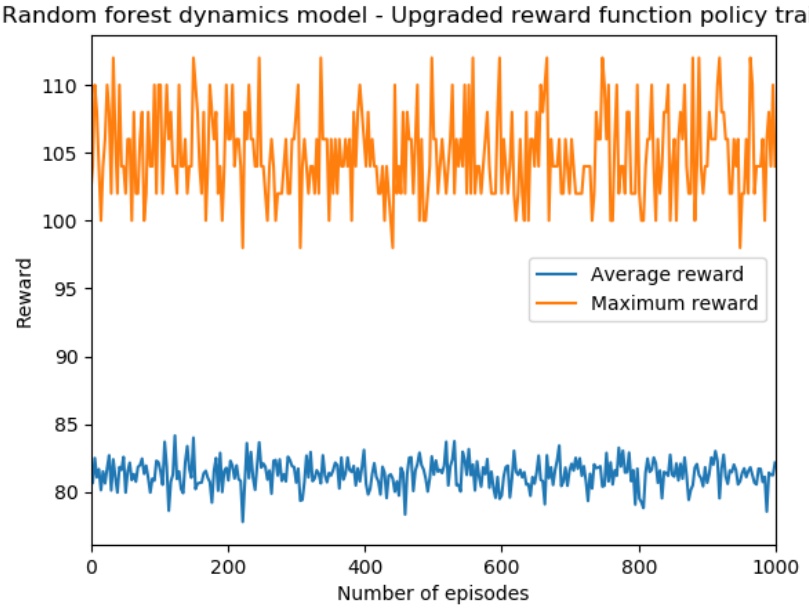


Figure 4.24: Random forest dynamics model - Upgraded reward function policy training

Using the random forest dynamics model with upgraded reward function and agents with 10 neurons in hidden layers, we see from the picture 4.24 that the graphs progress resemble the ones in 4.23. There are relatively big peaks in the maximum reward achieved per session reaching values of 114 and the average reward per session varied very little and achieved values of approximately 82.

The last elite combination tested was random forest dynamics model with distance reward function and agents with 80 neurons in hidden layers. The result can be seen in the picture 4.25. Here, the trend is the opposite - relatively big peaks in average reward (approximately -8500), but stable maximum reward per session (around -1900 points). In this case we can also interpret how good is the flight with maximum reward. Assuming we reached reward of -2000 points and an average flight having 200 steps, the average squared error per step is then -10 points. Having 18 attributes representing state gives us

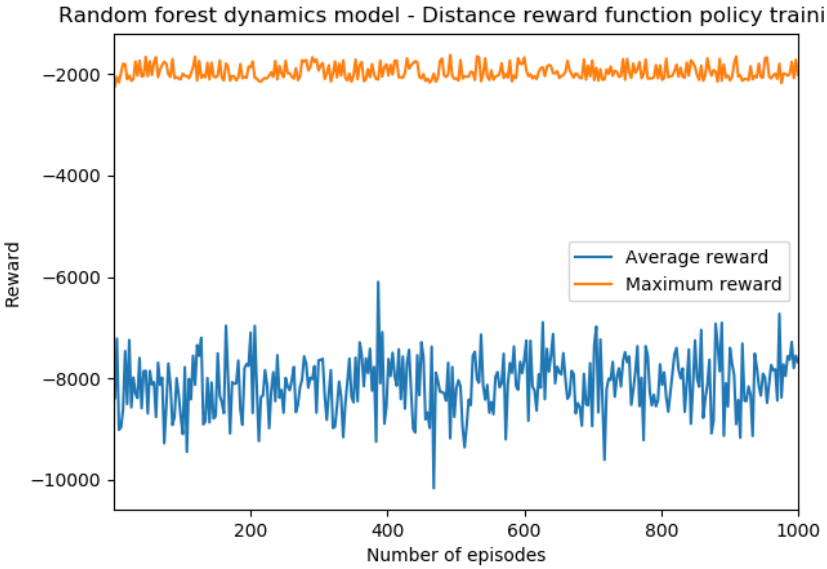


Figure 4.25: Random forest dynamics model - Distance reward function policy training

the average squared error per attribute of 0.5, which appears to be satisfying.

4.3.6 Back to X-Plane

Using the elite policies described and trained in previous section, we wanted to evaluate how well we are able to operate the aircraft in X-Plane. Using the connector described in 3.2.3, we connected to X-Plane via UDP (using our modified XPC)and set the state to our initial state. After this, we periodically read the state of X-Plane and according to a policy we created a reaction to this state. As we are producing discrete actions there is a need to transform $1, 0, -1$ back to valid values which can be sent back to X-Plane as datarefs.

We start with the action taken from the database as an initial action (a continuous action) and always construct the next continuous action by taking our new predicted action produced by the policy (discrete), going through it element by element and adding 1/100 of the range of the current continuous attribute to the previous action element if the predicted action element is 1, subtracting 1/100 of the range if the predicted action element is -1 and if the predicted action element is 0 taking the previous value without change. This is not applicable to the flaps handle which has 9 positions, gear handle and air brakes handle which have two positions. Here we simply add 1/8 or 1 to the previous value if the predicted discrete action is 1. After producing the continuous action we set the corresponding datarefs to its values. This whole cycle happens periodically once per second.

Chapter 5

Evaluation and Discussion

In previous chapter we described the process of implementation of connection to X-Plane, tuning and training. In this chapter we evaluate the outputs of the implementation.

5.1 Results in X-Plane

The three selected elite policies trained in 4.3.5 produced different results when connected to X-Plane. This evaluation was done by placing the aircraft in the initial state and taking actions according to trained policy until the aircraft touches the land. Videos of all three policies landing can be found on enclosed media.

5.1.1 Original best practices reward function

The first elite policy we selected was trained using the original best practices reward function and neural networks with relu activation function dynamics model. For the first few steps it produced valid actions, where we extended flaps, pulled out the gear and started applying air brakes, but the throttle control was not valid at all. When we needed to accelerate a bit as the aircraft pitch was too low, it tried to apply reverse throttle (which is used only for slowing down after the touchdown). Because of this, the aircraft started to descend too fast and hit the ground roughly, resulting in a crash.

We tried the same scenario several times, always with the same result - crash. The touchdown happened at around 6.5 nautical miles from the runway, which in the combination of insufficient smoothness of it disqualified this policy. An example of a touchdown produced by this policy can be seen in the picture 5.1.

To assess this approach formally, we look onto the stabilized approach conditions (the ones not applicable to our case were left out):

Figure 5.1: Original best practices policy touchdown

- The aircraft is on the correct flight path - ✗- As we failed to reach the runway, this condition is not satisfied.
- Only small changes in heading/pitch are necessary to maintain the correct flight path ✓- There were no rapid or drastic changes in heading/pitch (although not flying the correct flight path)
- The airspeed is not more than $V_{REF} + 20kts$ indicated speed and not less than V_{REF} - ✗- Descending too steeply caused the aircraft to fly around 200kts
- The aircraft is in the correct landing configuration - ✓- Our policy was able to reach the landing configuration.
- Sink rate is no greater than 1000 feet/minute. ✗- Too steep descend.

5.1.2 Upgraded best practices reward function

The second elite policy was trained using the upgraded best practices reward function and random forest dynamics model. This policy performed far better than the original one. It also pulled out the gear almost right after running the policy, but applied the flaps more delicately, not extending them fully in the first few steps. The descend was not as fast as with the original reward function and the touchdown itself was not smooth, but it did not result in a crash like the previous elite policy.

Figure 5.2: Upgraded best practices policy touchdown

Here, we also tried the same scenario multiple times to see how the policy performs in general and although it did not land directly on the runway, we clearly see improvement compared to the original reward function policy. The touchdown happened at approximately 4.5 nautical miles from the runway, without a crash and can be seen in the picture 5.2.

Taking into consideration stabilized approach conditions:

- The aircraft is on the correct flight path - ✗- As we failed to reach the runway, this condition is not satisfied.
- Only small changes in heading/pitch are necessary to maintain the correct flight path ✓- There were no rapid or drastic changes in heading/pitch (although not flying the correct flight path)
- The airspeed is not more than $V_{REF} + 20kts$ indicated speed and not less than V_{REF} - ✗- Descending too steeply caused the aircraft to fly around 180kts
- The aircraft is in the correct landing configuration - ✓- Our policy was able to reach the landing configuration.
- Sink rate is no greater than 1000 feet/minute. ✗- Still too steep descend.

Figure 5.3: Distance policy touchdown

5.1.3 Distance reward function

The last elite policy was trained using the distance reward function and random forest dynamics model. This policy seemed like it was going to crash fast at first, as it applied flaps and airbrakes fast (just like the original best practices policy), but even though it was descending fast, it was able to stabilize the aircraft by altering the pitch and finally resulting in a very smooth landing. The touchdown (seen in the picture 5.3) was still not on the runway, but this policy got the closest to it - roughly 4 nautical miles from the runway.

Looking onto the stabilized approach conditions:

- The aircraft is on the correct flight path - ✗- As we failed to reach the runway, this condition is not satisfied.
- Only small changes in heading/pitch are necessary to maintain the correct flight path ✓- There were no rapid or drastic changes in heading/pitch (although not flying the correct flight path)
- The airspeed is not more than $V_{REF}+20kts$ indicated speed and not less than V_{REF} - ✓- In this case, the aircraft was able to maintain a speed fit to land (not in all cases but mostly this condition was satisfied).
- The aircraft is in the correct landing configuration - ✓- Our policy was able to reach the landing configuration.

- Sink rate is no greater than 1000 feet/minute. ✗- The beginning of the approach was too steep, which was later stabilized.

5.1.4 Summary

As we can see the performance of policies varied, but none of the policies reached a pilot-like performance. However, we showed that our artificial intelligence was able to land an aircraft in a smooth way even though the accuracy of the touchdown was not sufficient. The goal of this thesis was a proof of concept and we can say that this was accomplished.

5.2 Future vision

The results produced by this thesis are interesting from the machine learning point of view. The policies themselves do not reach such a performance that could replace a real pilot in an aircraft, but rather show that replacement of the crew is possible. There are still many experiments that could have been done, but are out of scope of this diploma thesis. Here are a few ideas that could lead to better policies:

- Extract a different subset of state and action representing datarefs out of X-Plane. Inaccurate representation of these could lead to invalid dynamics model and inability to control the aircraft so that it lands successfully.
- Experiment with different methods of action discretization. We discretize the actions in order to create a dynamics model and policies produce discrete output as well. There are many other methods of discretization (maybe even work with continuous actions).
- Tune the dynamics models. In this work we focused on overview of more dynamics models and each of these models can be further researched to reach a better performance.
- Tune the policies. In this work we settled for a cross entropy method for reinforcement learning with a percentile of selected elites of 80. There is still space to tune this parameter or even use different method for training.

Conclusion

The goal of this thesis was to provide a proof of concept that the cockpit crew can be replaced by an artificial intelligence that would fully operate the aircraft. We first provided a research of how much airlines spend on the pilots and thus how much this concept would save. After, we did a survey of approaches to this problem (or similar ones) and came up with our own solution inspired by the survey.

We chose X-Plane 11 as our simulation environment and flew 10 example approaches to construct a dynamics model that would capture the dynamics of an aircraft during a descend. This was done to speed up the learning process of reinforcement learning as running the experiments in X-Plane would take too much time. The extraction of data was done via a Python plugin for launching scripts from inside X-Plane and is described in 4.1. We selected a subset of datarefs (data pointers inside X-Plane) which was consulted with a senior aviation system engineer and after extraction we reduced the number of attributes even more as some of them were constant. We further split these datarefs into two categories: 18 state representing datarefs and 6 action datarefs. In order to capture the dynamics of descent, we needed to have a model that takes tuple $[state_t, action_t]$ as input and produces $state_{t+1}$ as the output. As the output is expected to be continuous, we selected 5 models: linear regression, K-Nearest neighbors regression, decision tree regression, random forest regression and neural network regression. To tune the parameters in order to maximize the accuracy of our dynamics model we came up with two metrics described in 4.2.3 and according to them we performed experiments. We disqualified linear regression and decision tree regression because of poor performance.

After selecting the best performing parameters for each of the regression models, we set up the environment for training policies (described in the section 4.3.1). We came up with three different reward functions (two based on pilot best practices and one distance based reward function) and defined the agent of the policy in 4.3.2. The agent consisted of 6 sub-agents, each of

which was a neural network with two hidden layers and hyperbolic tangent activation function. As we trained them iteration by iteration, we set the parameter warm start to true. We selected cross entropy method for teaching the agents and selected the percentile of the most successful sessions to feed to the agent to 80. We also experimented with the size of number of neurons in hidden layers (from 5 to 100) in combination with different dynamic models and reward functions (described in section 4.3.3). This way we assessed the most successful combinations by either looking onto the steepness of learning (the average reward) or overall average score reached (our proposed reward functions were easily reversible to know how well the aircraft really performed) during the first 100 training episodes.

This way we chose three elite combinations for a 1000 episode training (more detailed in 4.3.5):

1. Neural network with 'relu' activation function, two hidden layers with 25 neurons in each of them as the dynamics model, original pilot best practices reward function with sub-agents with two hidden layers of 5 neurons.

2. Random forest regression with 20 trees and maximum depth of 15 per tree and MSE criterion for splitting as the dynamics model, upgraded pilot best practices reward function with sub-agents having 10 neurons per hidden layer.

3. Random forest regression with 20 trees and maximum depth of 15 per tree and MSE criterion for splitting as the dynamics model, distance reward function with sub-agents containing 80 neurons per hidden layer.

These combinations resulted in three different policies (their training outputs can be seen in the section 4.3.5) which we then tried back in X-Plane to see the real performance as until then we worked only with our trained dynamics models. None of the policies reached the runway, but the results were interesting nevertheless. We assessed the quality by looking onto the stabilized approach conditions and checking whether the flight operated by the policy satisfies them.

- The aircraft is on the correct flight path. None of the policies was able to satisfy this condition as all landed before the runway. The first policy landed at about 6.5 nautical miles before, the second 4.5 and the distance policy landed around 4 nautical miles before the runway.

- Only small changes in heading/pitch are necessary to maintain the correct flight path. As none of the policies landed on the runway, we looked onto this condition as smoothness in operation. Our policies did not perform any rapid or drastic maneuvers during the landing.

- The airspeed is not more than $V_{REF} + 20kts$ indicated speed and not less than V_{REF}, where V_{REF} is 140kts. The first two policies were not able to maintain such a speed and the distance policy maintained this speed for landing in most of the cases.
- The aircraft is in the correct landing configuration. All of the policies managed to get the aircraft into the landing configuration (full flaps, gear out).
- Sink rate is no greater than 1000 feet/minute. None of the policies were able to satisfy this condition. However, the distance policy was able to stabilize itself later to perform a smoother touchdown.

From the perspective of smoothness of landing, the first policy crashed the aircraft into the ground which disqualifies this policy, but the other two did not. Although the upgraded policy landed roughly and made a jump during the touchdown, it did not result in a crash. The last policy (the distance one) performed a rather smooth touchdown, which could be considered safe (more discussed in 5).

Overall, as this diploma thesis proves, the idea of landing using artificial intelligence is possible even though to achieve professional pilot performance it requires further development which is over the scope of this work.

Bibliography

[1] Falcus, M. The World's Largest Airlines [online]. December 2017, [Cited 2019-02-10]. Available from: http://www.airportspotting.com/worlds-largest-airlines/

[2] American Airlines Fleet Details and History [online]. February 2019, [Cited 2019-02-10]. Available from: https://www.planespotters.net/airline/American-Airlines

[3] Airbus. Airbus 2018 Price List Press Release [online]. January 2018, [Cited 2019-02-10]. Available from: https://www.airbus.com/newsroom/press-releases/en/2018/01/airbus-2018-price-list-press-release.html

[4] Prices of Boeing aircraft in 2018 [online]. January 2018, [Cited 2019-02-10]. Available from: https://www.statista.com/statistics/273941/prices-of-boeing-aircraft-by-type/

[5] Definition of Block Hour [online]. [Cited 2019-02-10]. Available from: https://www.lawinsider.com/dictionary/block-hour

[6] Available seat miles [online]. November 2018, [Cited 2019-02-10]. Available from: https://en.wikipedia.org/wiki/Available_seat_miles

[7] American Airlines Seat Maps [online]. [Cited 2019-02-10]. Available from: https://www.seatguru.com/airlines/American_Airlines/American_Airlines_AA_Boeing_737-800_C.php

[8] Inc., A. A. G. American Airlines Group Reports Fourth-Quarter and Full Year 2017 Profit [online]. January 2018, [Cited 2019-02-10]. Available from: http://news.aa.com/news/news-details/2018/American-Airlines-Group-Reports-Fourth-Quarter-and-Full-Year-2017-Profit/default.aspx

[9] Swelbar, W. S.; Belobaba, P. P. Employees Compensation [online]. 2018, [Cited 2019-02-10]. Available from: http://web.mit.edu/airlinedata/www/Employees&Compensation.html

[10] Grierson, M. Aviation History—Demise of the Flight Navigator [online]. October 2008, [Cited 2019-02-10]. Available from: https://www.francoflyers.org/2008/10/aviation-histor.html

[11] Boeing raises prospect of only one pilot in the cockpit of planes [online]. February 2018, [Cited 2019-02-10]. Available from: https://www.theguardian.com/world/2018/feb/09/boeing-raises-prospect-of-only-one-pilot-in-the-cockpit-of-planes

[12] Batchelor, T. SINGLE-PILOT PASSENGER PLANES COULD SOON TAKE TO THE SKIES, SAYS BOEING [online]. August 2018, [Cited 2019-02-10]. Available from: https://www.independent.co.uk/travel/news-and-advice/single-pilot-plane-boeing-autonomous-jet-technology-cockpit-a8506301.html

[13] American Airlines [online]. January 2019, [Cited 2019-02-10]. Available from: https://www.airlinepilotcentral.com/airlines/legacy/american_airlines

[14] Donohoe, A. What Is the Average Salary for an American Airline Pilot That Flies International Routes? [online]. June 2018, [Cited 2019-02-10]. Available from: https://work.chron.com/average-salary-american-airline-pilot-flies-international-routes-25650.html

[15] Safety, A.; Airplanes, B. C. Statistical Summary of Commercial Jet Airplane Accidents [online]. October 2018, [Cited 2019-02-10]. Available from: http://www.boeing.com/resources/boeingdotcom/company/about_bca/pdf/statsum.pdf

[16] WHAT IS ANGLE OF ATTACK? [online]. October 1999, [Cited 2019-02-10]. Available from: http://www.geocities.ws/pegasusair_tr/aoa.htm

[17] [Cited 2019-02-10]. Available from: https://i.stack.imgur.com/matiD.png

[18] L'hotellier, E. IFR ARRIVAL AND APPROACH PROCEDURES [online]. May 2017, [Cited 2019-02-10]. Available from: https://www.ivao.aero/training/documentation/books/SPP_APC_Arrival_Approach_procedure.pdf

[19] Crane, D. Dictionary of Aeronautical Terms. *Aviation Supplies & Academics*, , no. 3, 1997: pp. 213–241.

[20] Instrument Landing System (ILS) [online]. October 2014, [Cited 2019-02-10]. Available from: https://www.skybrary.aero/index.php/Instrument_Landing_System_(ILS)

[21] Stabilised Approach [online]. December 2018, [Cited 2019-02-10]. Available from: https://www.skybrary.aero/index.php/Stabilised_Approach

[22] Julia Behrend, F. D.; Koechlin, E. Impulsivity modulates pilot decision making under uncertainty. October 2017. Available from: https://www.hfes-europe.org/wp-content/uploads/2017/10/Behrend2017.pdf

[23] LDG. May 2010, [Cited 2019-02-10]. Available from: https://www.skybrary.aero/index.php/LDG

[24] Autoland. September 2017, [Cited 2019-02-10]. Available from: https://www.skybrary.aero/index.php/Autoland

[25] Why don't pilots always use autoland? January 2011, [Cited 2019-02-10]. Available from: https://aviation.stackexchange.com/questions/910/why-dont-pilots-always-use-autoland

[26] Anitha, G.; Kumar, R. G. Vision Based Autonomous Landing of an Unmanned Aerial Vehicle. *Procedia Engineering*, volume 38, 2012: pp. 2250 – 2256, ISSN 1877-7058, doi:https://doi.org/10.1016/j.proeng.2012.06.271, iNTERNATIONAL CONFERENCE ON MODELLING OPTIMIZATION AND COMPUTING. Available from: http://www.sciencedirect.com/science/article/pii/S1877705812021844

[27] Ng, A. Y.; Coates, A.; et al. *Autonomous Inverted Helicopter Flight via Reinforcement Learning*. Springer Berlin Heidelberg, 2006, p. 363–372, doi:10.1007/11552246_35. Available from: http://dx.doi.org/10.1007/11552246_35

[28] Kim, H. J.; Jordan, M. I.; et al. Autonomous Helicopter Flight via Reinforcement Learning. In *Advances in Neural Information Processing Systems 16*, edited by S. Thrun; L. K. Saul; B. Schölkopf, MIT Press, 2004, pp. 799–806. Available from: http://papers.nips.cc/paper/2455-autonomous-helicopter-flight-via-reinforcement-learning.pdf

[29] Abbeel, P.; Coates, A.; et al. An Application of Reinforcement Learning to Aerobatic Helicopter Flight. In *Advances in Neural Information Processing Systems 19*, edited by B. Schölkopf; J. C. Platt; T. Hoffman, MIT Press, 2007, pp. 1–8. Available from: http://papers.nips.cc/paper/3151-an-application-of-reinforcement-learning-to-aerobatic-helicopter-flight.pdf

[30] Choi, S.; Kim, S.; et al. Inverse reinforcement learning control for trajectory tracking of a multirotor UAV. *International Journal of Control, Automation and Systems*, volume 15, no. 4, Aug 2017: pp. 1826–1834, ISSN 2005-4092, doi:10.1007/s12555-015-0483-3. Available from: https://doi.org/10.1007/s12555-015-0483-3

[31] Hazi, O. Final Approach – An Automated Landing System for the X-Plane Flight Simulator. 2017. Available from: http://cs229.stanford.edu/proj2011/Hazi-FinalApproach.pdf

[32] Baomar, H.; Bentley, P. J. An Intelligent Autopilot System that learns piloting skills from human pilots by imitation. In *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, June 2016, pp. 1023–1031, doi:10.1109/ICUAS.2016.7502578.

[33] Ferrante, R. A Robust Control Approach for Rocket Landing. 2017.

[34] van Otterlo, M.; Wiering, M. *Reinforcement Learning and Markov Decision Processes*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, ISBN 978-3-642-27645-3, pp. 3–42, doi:10.1007/978-3-642-27645-3_1. Available from: https://doi.org/10.1007/978-3-642-27645-3_1

[35] Sutton, R. S.; Barto, A. G. *Reinforcement Learning: An Introduction*. MIT Press, 1998. Available from: http://www.cs.ualberta.ca/~sutton/book/the-book.html

[36] Kaelbling, L. P.; Littman, M. L.; et al. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, volume 4, May 1996: p. 237–285, ISSN 1076-9757, doi:10.1613/jair.301. Available from: http://dx.doi.org/10.1613/jair.301

[37] Rubinstein, R. Y.; Kroese, D. P. *The Cross-Entropy Method*. Springer New York, 2004, doi:10.1007/978-1-4757-4321-0. Available from: http://dx.doi.org/10.1007/978-1-4757-4321-0

[38] Abbeel, P.; Ng, A. Y. Exploration and apprenticeship learning in reinforcement learning. In *Proceedings of the 22nd international conference on Machine learning - ICML '05*, ACM Press, 2005, doi:10.1145/1102351.1102352. Available from: http://dx.doi.org/10.1145/1102351.1102352

[39] X-Plane - Dataref Search [online]. [Cited 2019-02-10]. Available from: https://www.siminnovations.com/xplane/dataref/

[40] X-Plane - Command Search [online]. [Cited 2019-02-10]. Available from: https://www.siminnovations.com/xplane/command/

[41] PYTHON INTERFACE PLUGIN [online]. [Cited 2019-02-10]. Available from: http://www.xpluginsdk.org/python_interface.htm

[42] XPlaneConnect [online]. [Cited 2019-02-10]. Available from: https://github.com/nasa/XPlaneConnect

[43] Delta Air Lines, I. *737NG Operations Manual*. June 2008. Available from: http://conmachine.com/pluginfile.php/9934/mod_resource/content/1/737NG-FCOM.pdf

[44] BOEING 737-800 [online]. [Cited 2019-02-10]. Available from: https://www.skybrary.aero/index.php/B738

[45] Boeing 737–800 Flight Notes [online]. [Cited 2019-02-10]. Available from: http://krepelka.com/fsweb/learningcenter/aircraft/flightnotesboeing737-800.htm

[46] Brockman, G.; Cheung, V.; et al. OpenAI Gym. *CoRR*, volume abs/1606.01540, 2016, 1606.01540. Available from: http://arxiv.org/abs/1606.01540

Appendix A

Acronyms

ACARS Aircraft communications addressing and reporting system

AOA Angle of attack

API Application programming interface

APV Approach procedures with vertical guidance

ASM Available seat mile

CAS Calibrated airspeed

CASM Cost per available seat mile

CAST Commercial Aviation Safety Team

CE Cross entropy

DBSCAN Density-based spatial clustering of applications with noise

DNA Deoxyribonucleic acid

EAS Equivalent airspeed

GPS Global Positioning System

GS Ground speed

IAF Initial approach fix

IAP Instrument approach procedure

IAS Indicated airspeed

ICAO The International Civil Aviation Organization

IF Intermediate fix

IFR Instrument flight rules

ILS Instrument landing system

IMC Instrument meteorological conditions

KIAS Knots-Indicated airspeed

KNN K-Nearest neighbor

KTAS Knots-True airspeed

MAE Mean absolute error

MDP Markov decision process

MLP Multi-layer perceptron

MLS Microwave landing system

MSE Mean squared error

NASA National Aeronautics and Space Administration

NG Next generation

NN Nearest neighbor

NPA Non-precision approach

PA Precision approach

RL Reinforcement learning

SDK Software development kit

STAR Standard instrument arrival

TAS True airspeed

UDP User datagram protocol

VMC Visual meteorological conditions

XPC X-Plane Communication Toolbox

Appendix B

Contents of CD

- `readme.txt` ..... the file with CD contents description
- `database_scripts` ..... the scripts for manipulation with databases
  - `create_database.py` ..... create database from extracted data
  - `database_work.py` ..... database manipulation script
  - `discretize_actions.py` ..... discretize actions
  - `discretize_actions_binning.py` ..... discretize actions with binning
  - `read_from_database.py` ..... select features
- `databases` ..... databases of states and actions
  - `discrete_actions.db` ..... database with discrete actions
  - `features_selected_flights.db` ..... database with feature selection
  - `flights.db` ..... original database
- `environment` ..... models of environments described in this thesis
- `model_scripts` ... training, tuning and evaluating dynamics model scripts
- `original_data` ..... csv files of original extracted data from X-Plane
- `policy` ..... policy oriented scripts - training, controlling X-Plane
- `results` ..... trained policies, dynamics models and rewards
- `thesis_text` ..... the text of this thesis
- `videos` ..... videos of elite policies landing