FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Learn to Fly: Cloning the Behavior of a Pilot

César Manuel Nobre Medeiros

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Daniel Castro Silva

Second Supervisor: Rui Camacho

July 31, 2021

Learn to Fly: Cloning the Behavior of a Pilot

César Manuel Nobre Medeiros

Mestrado Integrado em Engenharia Informática e Computação

July 31, 2021

Abstract

Autonomous vehicles are increasingly becoming more relevant in the last years, from autonomous driving to autopilot systems on aircraft. With the recent growth of the UAV market and its use in the most diverse areas, there is often a need for a system that can navigate autonomously to perform the intended task.

To simulate these systems' dynamics, LIACC developed a platform based on Microsoft Flight Simulator X (FSX). This platform already includes an autopilot system for controlling aircraft. However, this system is quite limited, and similarly to real autopilot systems, it does not replace an expert pilot. It rather assists them in slow changes in the aircraft's altitude and heading.

Thus, this dissertation aims to create an automatic pilot capable of imitating a real pilot behavior by creating a model that performs aerobatic maneuvers that only experienced pilots can do. As an example, it is used the Immelmann turn maneuver with a parameterized target height to test the learned model capabilities.

In order to solve this problem, a behavior cloning approach is used, in which the model learns from the pilot's movements. Initially, relevant data about the pilot, aircraft, and environment is collected through a plugin developed for Microsoft Flight Simulator (FSX). Later, two different approaches, one with an end-to-end long short term memory (LSTM) and another with an end-to-end artificial neural network (ANN) are used to train two models from the human demonstrations. One to control the aircraft elevator and another to control the aircraft aileron.

Finally, both models' performance is evaluated by comparing the models' behavior with a human pilot in a well-determined experiment. The experiment consists of describing an Immelmann turn to a final height of 11000 feet with a Boeing F/A-18 in the same environmental conditions. It is compared the models performed trajectory with the average trajectory performed by a human. Other features like trajectory smoothness and distance to the target altitude are also evaluated. The elevator and aileron position predictions are also compared with human demonstrations.

The results show that the LSTM model is capable of performing the Immelmann turn to any reasonable desired altitude. The trajectory performed by the model is smooth and does not contain oscillations common with human operators.

Keywords: imitation learning, behavioural cloning, flight simulator, autonomous aircraft, UAV, LSTM, ANN

Resumo

Os veículos autónomos estão a tornar-se cada vez mais relevantes nos últimos anos, desde a condução autónoma em automóveis até aos sistemas de piloto automático em aeronaves. Com o recente crescimento do mercado de UAVs e a sua utilização nas mais diversas áreas, há frequentemente necessidade de um sistema que possa navegar autonomamente para realizar a tarefa pretendida.

Para simular a dinâmica destes sistemas, o LIACC desenvolveu uma plataforma baseada no Microsoft Flight Simulator X (FSX). Esta plataforma já inclui um sistema de piloto automático para o controlo de aeronaves. Contudo, este sistema é bastante limitado e, à semelhança dos sistemas de piloto automático reais, não substitui um piloto especializado. Pelo contrário, assiste-os em mudanças lentas na altitude e direção da aeronave.

Assim, esta dissertação visa criar um piloto automático capaz de imitar o comportamento de um piloto real, criando um modelo de controlo que executa manobras acrobáticas que só pilotos experientes podem fazer. Como exemplo, é utilizada a manobra de Immelmann com uma altura alvo parametrizada para testar as capacidades do modelo aprendido.

A fim de resolver este problema, é utilizada uma abordagem de behavioral cloning, na qual o modelo aprende com os movimentos do piloto. Inicialmente, são recolhidos dados relevantes sobre a aeronave e ambiente através de um plugin desenvolvido para o Microsoft Flight Simulator (FSX). Mais tarde, duas abordagens diferentes, uma com Long Short Term Memory (LSTM) e outra com Artificial Neural Network (ANN), são utilizadas para treinar dois modelos a partir das demonstrações humanas. Um modelo para controlar o elevator da aeronave e outro para controlar o aileron da aeronave.

Finalmente, o desempenho de ambos os modelos é avaliado comparando o comportamento dos modelos com um piloto humano, numa experiência bem defininda. A experiência consiste em descrever uma manobra de Immelmann a uma altura final de 11000 pés com um Boeing F/A-18 nas mesmas condições de ambiente. Compara-se a trajectória dos modelos realizados com a trajectória média realizada por um humano. Outras características como a suavidade da trajectória e a distância até à altitude alvo são também avaliadas. As previsões de posição do elevator e do aileron são também comparadas com demonstrações humanas.

Os resultados mostram que o modelo LSTM é capaz de executar a manobra de Immelmann para qualquer altitude desejada desde que acima da altura inicial. A trajectória executada pelo modelo é suave e não contém oscilações que são comuns com os operadores humanos.

Keywords: imitation learning, behavioural cloning, simulador de voo, aeronave autónoma, UAV, LSTM, ANN

Contents

List of Figures

List of Tables

Abbreviations

6DoF	Six degrees of freedom
AI	Artificial intelligence
ANN	Artificial neural network
API	Application Programming Interface
CSV	Comma-separated values
FEUP	Faculty of Engineering of the University of Porto
FSX	Microsoft Flight Simulator X
IRL	Inverse reinforcement learning
LIACC	Artificial Intelligence and Computer Science Laboratory Science
LQR	Linear quadratic regulator
LSTM	Long short term memory
MAV	Micro aerial vehicles
MPC	Model predictive control
NN	Neural network
PID	Proportional integral derivative
RC	Radio Control
RL	Reinforcement learning
SDK	Software Development Kit
SINDy	Sparse Identification of Nonlinear Dynamics
UAV	Unmanned Aerial Vehicle

Chapter 1

Introduction

This chapter introduces the dissertation topic, starting by describing the context, motivation, and objectives of this dissertation as well as the methodology used.

1.1 Context and Motivation

In recent years we have witnessed the automation of several processes with the application of robots in different areas, from commercial, business, and military. The challenge, at the moment, is related to intelligent autonomous robotics that performs complex tasks without the need for a human operator.

Autonomous vehicles are among the most attractive areas as they bring several advantages in carrying out civilian and military missions [Bouabdallah and Siegwart, 2007]. Among other things, it allows access to places that would be difficult for humans [Flint et al., 2002], perform long-term missions [Kanistras et al., 2013], and reduce operational costs [Koh and Wich, 2012].

To simulate the execution of a vast set of missions by diverse heterogeneous autonomous vehicles, the Artificial Intelligence and Computer Science Laboratory (LIACC) developed a multi-agent platform on top of the Microsoft Flight Simulator (FSX) [Silva, 2011].

In order to autonomously control vehicles in the simulator, LIACC developed a vehicle control agent. This agent has as its basis the AI autopilot build in FSX that allows the control of various vehicle types on the platform.

This agent contains two modules of particular interest for this dissertation. The first is the *Planning and High-Level Reasoning Module* responsible for generating high-level maneuvers, such as go-to point or making a circle. The latter is the *Vehicle Maneuvering Control Module* that transforms high-level maneuvers into low-level maneuvers that can be interpreted through the communication interface that Microsoft provides to connect the client application with FSX.

The AI autopilot approach to the present problem features some limitations. The rigidity of the autopilot is one of them. The autopilot is not capable of performing steep maneuverings. The

movements are composed of small increments that keep the plane stable throughout the journey. Although this behavior is desirable for commercial flights, it does not always apply to all types of missions. In military missions where completion time is crucial or even when necessary to cope with unusual or unforeseen circumstances occasionally encountered in routine flight, the ability to execute performance maneuvers can be decisive in its success.

1.2 Objectives and Methodology

This dissertation aims to extend the *Vehicle Control Agent* capabilities to take full advantage of each aircraft's capabilities. This results in more realistic mission strategies, approximating what real human pilots would do. Better mission time, as it can "shortcut" way-points instead of performing a stable trajectory and also supporting future missions that were not possible until now, like air combat and other military tasks.

In order to do this, there needs to be support for rapid changes in the aircraft's velocity vector and the aircraft's angular rates. One way to push the aircraft's capabilities to their limits is to use aerobatic maneuvers. These types of maneuvers put the aircraft in very unstable states, suitable for the rapid changes mentioned before. However, it presents a real challenge to perform them without losing control.

Creating a flight dynamics model that describes aircraft behavior in the simulator can be very difficult, mainly if the approach taken calculates the physical forces acting on a simulated aircraft. This method requires expert knowledge on each maneuver, aircraft, environment conditions, physics engine, among others, which would be infeasible in this case.

To work around this problem, the autopilot will learn the aerobatic maneuvers from human demonstration. First, all relevant data needs to be collected, including human control input data, aircraft state variables, and environment condition variables.

All data collected will then be combined and fed to an imitation learning module to learn the policies that will perform the desired behavior.

Another essential part of this process is performance evaluation, not only at the end but also during the learning process.

1.3 Document structure

The rest of the document consists of 6 chapters. Chapter 2 presents concepts and terms specific to the flight dynamics that will be used throughout the document. Chapter 3 reviews the literature and analyzes state of the art. Chapter 4 specifies the approach that is taken to solve the problem of this dissertation. Chapter 5 presents more in-depth the implementation of the several approaches and reasoning, how they work, and the reason behind the decisions taken. Chapter 6 contains the experiment done to evaluate the final solution presenting several metrics. Chapter 7 ends the report by presenting a summary of what has been accomplished and an outline for future work.

Chapter 2

Contextualization

This chapter presents some concepts and terms that will be used throughout the rest of the dissertation. It will serve to introduce the reader to concepts related to aircraft aerodynamics, 'The Platform' developed by LIACC, and the Microsoft Flight Simulator.

2.1 Aircraft flight dynamics

An aircraft is any vehicle, with or without an engine, that can fly by gaining support from the air. Flying can be achieved in different ways, thus emerging different aircraft with varying aerodynamics. Some examples are fixed-wing and rotary-wing aircraft. This dissertation, given its objective, will focus on the first.

To comprehend how they fly, first, we need to know the forces involved. There are four main forces applied to the aircraft: lift, thrust, drag, and weight (Fig. 2.1).

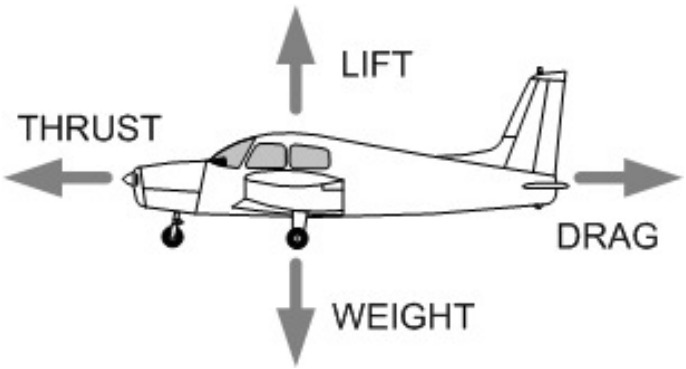


Figure 2.1: The Four Forces of Flight[1]

In a fixed-wing aircraft, the engine produces thrust that creates movement by pushing air in and then out in the opposite direction. Due to the airfoil shape wings, when there is a movement

[1]Retrieved from https://www.aeros.co.uk/latest-news/intro-principles-flight

relative to the air, a lift force is generated that makes the aircraft ascend if this force is greater than the weight. The drag represents the resultant of forces that act opposite the direction of motion and slow down the aircraft. To change the plane's direction the forces must be unbalanced in a specific way.

The main control surfaces of a fixed-wing aircraft are the ailerons, the elevators, and the rudder (Fig. 2.2).

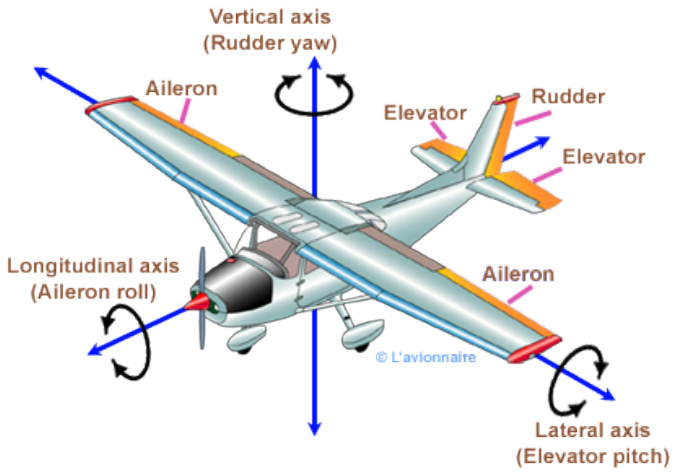


Figure 2.2: Primary Flight Controls[2]

The ailerons control the rotation around the longitudinal axis, also referred to as roll. They move in the opposite direction to increase the lift force difference on each wing and thus rotate to the side where the aileron is up.

The elevators control the rotation around the lateral axis, also known as pitch. Unlike the previous case, both elevators move in the same direction. When down, the lift force in the tail is more significant than in the nose, so the plane pitches down. The opposite occurs when the elevators are up.

The rudder controls the rotation around the vertical axis, commonly known as yaw. It is located on the tail and can move to the left or right. When positioned left, it creates a side force on the tail to the right, thus yawing the plane to the left. The opposite occurs when the rudder is to the right.
.

2.2 Aerobatic maneuvers

The majority of known aerobatic maneuvers are composed by three basic maneuvers or parts of them. There is the roll (Fig. 2.3) where the aircraft performs a full 360 degrees turn about its longitudinal axis by changing the deflection angle of the ailerons.

[2]Retrieved from https://www.lavionnaire.fr/VocableFlightControl.php

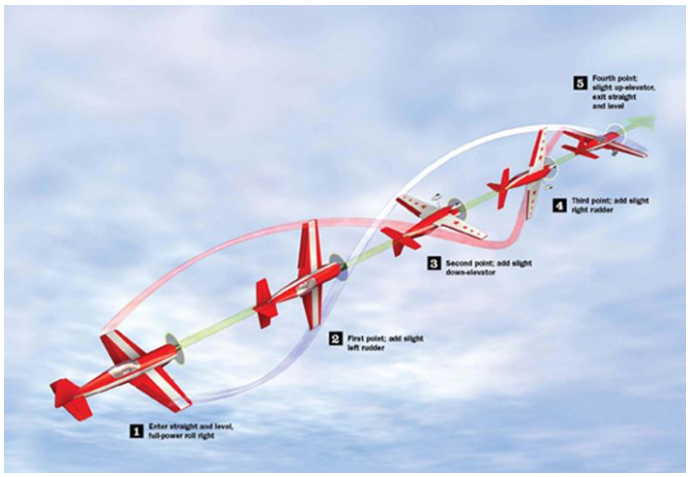


Figure 2.3: Roll[3]

The loop (Fig. 2.4) is another basic maneuver where the aircraft performs a full 360 degrees turn in the vertical plane by changing the deflection angle of the elevators.

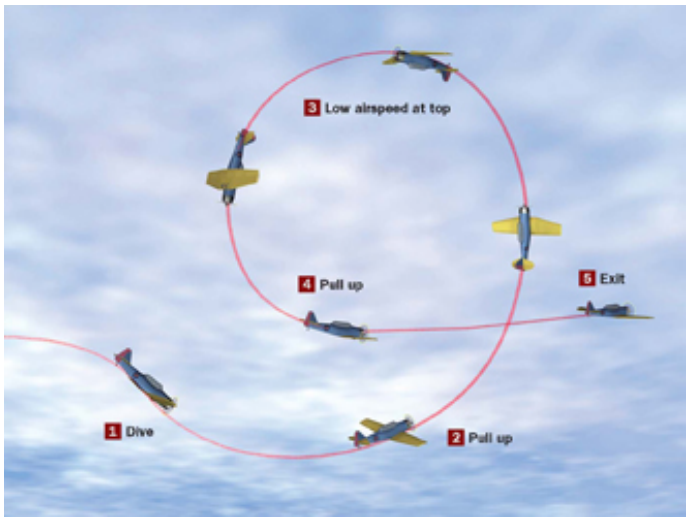


Figure 2.4: Loop[4]

Finally, the knife-edge turn (Fig. 2.5) pushes the bank angle limit, that is, the angle that the vehicle's longitudinal axis makes with respect to the horizontal. To perform this maneuver, the bank angle will need to be above 45º and can reach 180º in the case of a knife-edge turn.

[3]Retrieved from https://laptrinhx.com/perfecting-the-4-point-roll-1612734735/
[4]Retrieved from https://www.modelairplanenews.com/fly-scale-maneuvers/#outer-popup

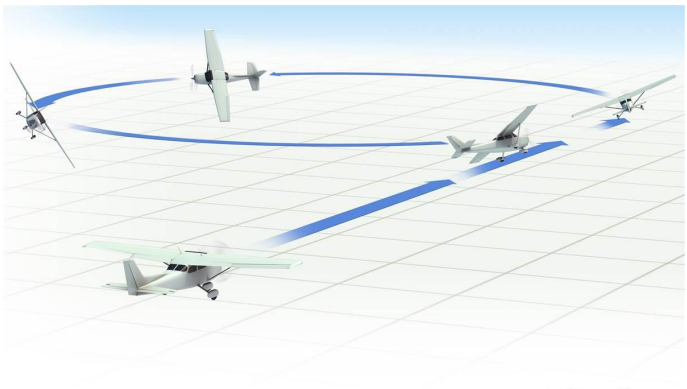


Figure 2.5: Steep turn[5]

By combining different basic maneuvers, we can achieve more complex ones such as the half cuban eight, canopy roll, split, and Immelmann turn.

The Half Cuban Eight (Fig. 2.6), is widely used in Red Bull Air Race competitions. This maneuver consists of reversing the direction of movement with the minimum horizontal displacement. During this maneuver, the center of mass remains in the same vertical plane in an ideal case.

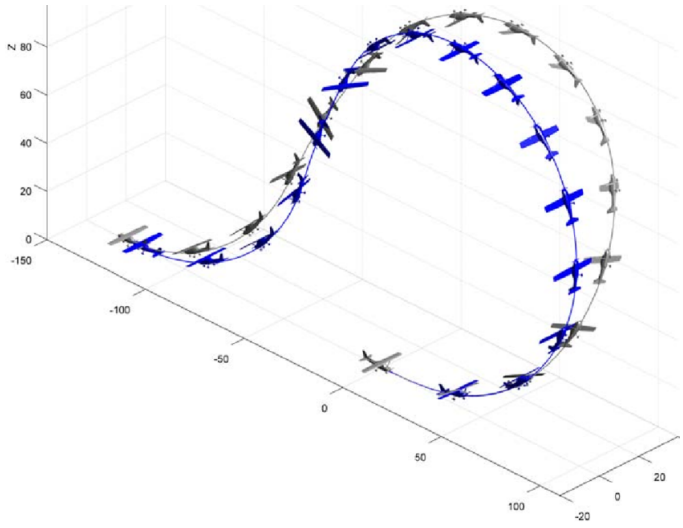


Figure 2.6: Half Cuban eight[6]

The Immelmann Turn (Fig. 2.7) consists in reversing the direction of movement increasing the altitude. It is used in air combat for repositioning after an attack. When the aircraft is parallel to the ground plane with the top facing up (0 degree bank angle and 0 degree pitch angle) the maneuver can be described in two part. An initial half-loop with the aircraft with diameter equal

[5]Retrieved from https://www.aopa.org/news-and-media/all-news/2012/march/flight-training-magazine/technique--the-steep-turn

[6]Retrieved from https://www.researchgate.net/figure/Half-Cuban-Eight-without-blue-and-with-wind-grey_fig4_269255066

to the difference between the desired altitude and current altitude. At the end the aircraft should be parallel to the ground, but this time the aircraft top should be facing down. After that a 180 degree roll is applied in order to the aircraft end up in a up right position.

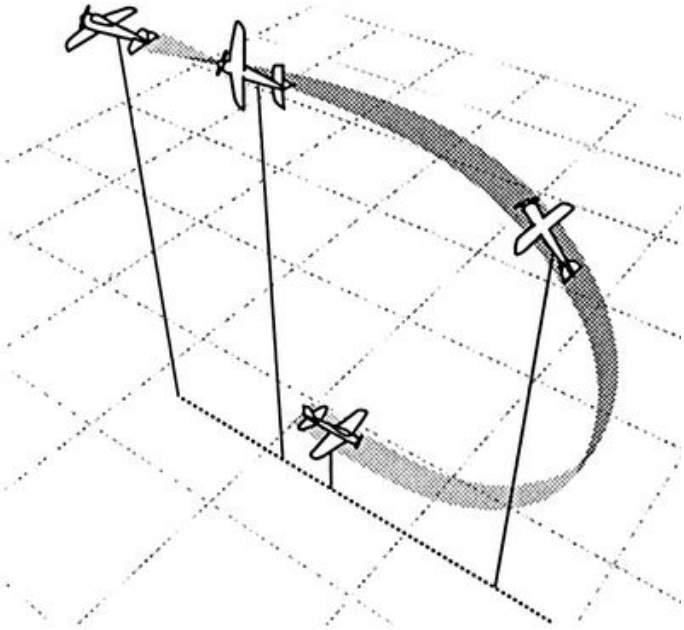


Figure 2.7: Immelmann turn[7]

The Split S(Fig. 2.8), on the other hand, is used to disengage from combat. It can be seen as an inverted immelmann turn. With the same initial state, aircraft parallel to the ground with the top facing up, it start by performing a 180 degree roll followed by a half loop with the top facing the center of the circumference described.

[7]Retrieved from http://steampunkaviatrix.blogspot.com/2010/05/history-of-immelmann-turn.html

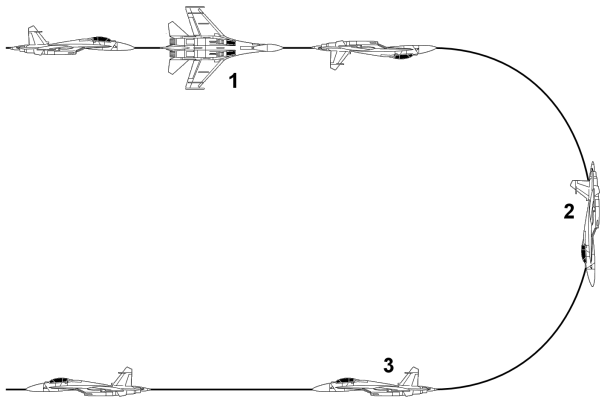


Figure 2.8: Split S[8]

2.3 Simconnect API

Simconnect is a software development kit (SDK) developed by Microsoft that connects the FSX simulator to an external application. It allows communication between the two programs, which makes it possible to control and track the simulator variables.

The Simconnect API supports both Key Events and Variables.

The Key Events allow control of all aircraft systems similar to a user using a joystick to control the simulation. These systems include Engine, Flight Controls, Fuel System, Instruments, Lights, and many others.

Variables, on the other hand, allow a more flexible communication allowing not only to change aircraft system variables but also reading all kinds of information related to the simulation. This includes environment variables like wind velocity and barometric pressure, aircraft control variables, like throttle level, control surfaces, and also aircraft state variables that allow reading and writing the velocity, acceleration, angle position, and angle rates.

SimConnnect is a software development kit (SDK) that contains all low-level methods capable of establishing communication between the user program and the FSX. It uses a client/server asynchronous format. This way, it triggers an event on the client each time the request is completed.

Then to pull the desired variables, they must be previously added to a data definition struct. This allows reading "simultaneously" instead of one by one, which maintains variable time coherence and allows for much faster processing.

Simconnect gives four options for the variable pulling rate. Only one time, every visual frame (every time it's rendered), every simulator frame (event if not rendered), and every second.

To write values to the variables, we simply need to initialize the data definition struct mentioned before and send it as an object to Simconnect. Every variable would be substituted by the new values. Officially it supports C/C++/.NET programs or DLLs.

[8]Retrieved from https://commons.wikimedia.org/wiki/File:Split_S.png

Chapter 3

Literature review

This chapter analyses the state of the art of autonomous vehicles and the different processes required to build and learn a controller model.

There are currently many distinct ways to autonomously control an aircraft, from traditional systems to systems based on neural networks. Each model offers a varying set of features like robustness, adaptability, optimality, simplicity, tracking ability, fast response, and disturbance rejection, and as such, there is no perfect solution for every application [Zulu and John, 2014].

Automating an aerial vehicle's navigation is a complex task given the nonlinear, high-dimensional (6DoF) properties of the system [McConley et al., 2000]. This is even more evident when we deviate from simple tasks such as maintaining altitude and enter a more complex tasks that require sudden changes of position and orientation.

3.1 Traditional controllers

It is prevalent in autonomous aerial vehicles to use a traditional approach to control the aircraft's position and orientation. This type of approach is strongly physics-based and depends significantly on the accuracy of the aircraft dynamics model.

For primitive tasks such as maintaining altitude or level flight, a simple model is enough. However, when the objective is performing aerobatic maneuvers, the accuracy needed for the model quickly rises.

In [Bulka and Nahon, 2017] and [Bulka and Nahon, 2019] the authors develop a fixed-wing aircraft control system that is able to follow a given trajectory by controlling the aircraft actuators, such as thrust, aileron, rudder, and elevator deflection.

This nonlinear system is physical-based, which means it was necessary to hand-specify aerodynamic equations of fixed-wing aircraft. These equations describe the aircraft's behavior and are used to change its position and orientation by changing the actuator's deflection.

This control is divided into three separate modules: an attitude controller, a position controller, and a thrust controller working together to follow a trajectory.

The paper also specifies a maneuver generator responsible for generating a time history of reference motion variables. This is done by applying constraints to the motion variables. The results show that even in trajectories that are not feasible, it was enough for the controller to follow a similar trajectory.

The maneuvers used in this paper were knife-edge, rolling Harrier, hover, and aggressive turnaround. They were tested both in a conventional simulator and a Hardware-In-The-Loop (HIL) simulator to further evaluate the model response in a more realistic scenario. The latter used X-Plane, a flight simulator similar to FSX, as a physics engine.

To compare the obtained trajectory with the ideal one, it was used the cross-track error(XTE), which is determined by the minimum distance between the actual position and the desired track measured at a 90 degrees angle from it. The aggressive turnaround has a 0,5m cross-track error. The knife-edge and rolling harrier have both a 1m.

A similar approach was used in [Levin et al., 2017] for performing a knife-edge maneuver. Only that in this paper, the trajectory is generated by a rapidly-exploring random trees (RRT) algorithm, which results in a smooth, collision-free overall trajectory. This also allows the integration of such maneuvers in a conventional path. To evaluate the proposed methodology of motion planning and control results, the author used Simulink with their dynamics model.

[McConley et al., 2000] use a different approach to maneuver an aircraft autonomously. It uses a hybrid method that combines off-line generation of primitive trajectories easily defined with on-line motion planning that choose when and which primitive trajectory to choose. This method consists of dividing the maneuvers' trajectory into time-parameterized curves. Instead of generating optimal control for a high-dimensional problem, it is only necessary to connect the primitives, thus reducing computational complexity. Some more complex maneuvers integrated into the primitives were loops, barrel rolls, and split-S. However, such maneuvers need to be hand-engineered and consequently not very dynamic. The motion planning is done by using rapidly exploring random trees (RRT's) to compute the optimal path. Despite being able to perform the intended maneuvers in simulation, this method would hardly yield good results in real-life scenarios where disturbances and errors would destabilize the primitive trajectories.

[Silva et al., 2009] uses the same platform as this dissertation to control an aircraft autonomously. In this paper, one of the approaches used is a PID controller. This approach noted some drawbacks since it is necessary to be continuously making adjustments to the aircraft's control surfaces. It would be necessary to send information at high rate, which wasted computational power, in addition to the fact that the latency of sending data would affect the accuracy of the movements performed. Furthermore, with different aircrafts and different environments, adjustments to proportional integral and derivative gains would be necessary.

3.2 Reinforcement Learning

In the previous section, we saw that in order to perform specific maneuvers, a traditional controller needs to have high detail *a priori* knowledge about the aircraft aerodynamic model and physics. This, as we can imagine, is not an easy task and certainly demands a profound knowledge about the aircraft and physical world. Reinforcement learning (RL) alleviates the need for a well-known dynamical model (model-based RL) or even discard it completely (model-free RL).

3.2.1 Model-free RL

Model-free reinforcement learning has a notable advantage over traditional methods given that it does not require an aircraft dynamical model. This dramatically reduces the expertise needed for the particular problem dynamics, and as a result, the learned model is more flexible to variances.

In [Abouheaf et al., 2018] is applied a model-free gradient-based solution to control an Unmanned Flexible Wing Aircraft autonomously in two directions, lateral and longitudinal. Also, Artificial neural networks were used to approximate the optimal policy of control. Several simulation scenarios were used to extensible test the learned model stabilization performance under a wide range of uncertainties and disturbances. A flexible wing hang glider was chosen since its characteristics are greatly affected by disturbances in the environmental dynamics. The results show a fast response to disturbances. However, this paper only presents the capacity to stabilize the glider and not aerobatic maneuvers, which would increase the problem complexity.

[Hwangbo et al., 2017] propose another way to stabilize an aircraft, in this case, a quadrotor. The authors used a neural-network deterministic on-policy that could directly map the state to an action that controls the rotors' thrust. The deterministic part allows for a more simple training model and more predictable actions by the agent. The training was done in a simulator platform to generate data faster and in great quantity without the need for trial-and-error in the real robot that would consume not only time but also resources. Both simulation and real hardware results show that the model is capable of waypoint tracking and stabilization even with harsh disturbances.

A more aggressive maneuver was introduced in [Lin et al., 2019] with a quadrotor flying through a narrow passage. The approach used an end-to-end neural network approach, and contrary to the article above, two neural networks were used — one for motion planning the trajectory and the other for controlling the quadrotor actuators. The performance is then optimized in a simulator with reinforcement learning. This process's necessary data was obtained through trial sets and consisted of 20000 trajectories with 1000 sample points each. Ultimately the agent was able to perform well in real hardware, similar to imitation learning, with the disadvantage of not having such a smooth trajectory.

3.2.2 Model-based RL

Model-based reinforcement learning techniques try to overcome the problem of lack of previous knowledge by allowing the agent to represent its environment [Janner et al., 2019]. This has great

impact on learning efficiency, as it requires less data and time to learn a policy when compared to Model-free techniques.

[Yoo et al., 2021] use a hybrid approach to control a micro quadrotor UAV. It consists in combining reinforcement learning to define the policy and trajectory and a Linear Quadratic regulator (LQR) to apply the correct actions to the UAV actuator. In the experiments, using ROS for learning and control the real quadrotor, the agent is capable of tracking a spline and circular reference trajectory. By using LQR as a control algorithm instead of relying entirely on reinforcement learning the authors were able to improve the convergence rate.

Also using a quadrotor as UAV, [Liang et al., 2018] propose a model-based algorithm that is able to increase training efficiency by combining data from different UAVs running in parallel. This time the trajectories pose a more difficult challenge with sharp turns and sudden altitude changes. The experiments were carried out in a simulator where it was possible to compare the proposed approach with traditional reinforcement learning. The performance obtained was similar in both situations, namely the tracking accuracy of the reference trajectory.

[Becker-Ehmck et al., 2020], different from what we have seen until now, without using any hand-engineered physics and aerodynamics equations the authors were able to create a dynamical model for a UAV quadrotor based on sensor observations. The method is composed by two parts, learn dynamical model and learn control model. For both tasks were used deep leaning methods. The experiments carried out in real hardware have proved to be successful in flying to predefined position by a marker.

3.3 Imitation Learning

Sometimes the best way to learn something is by watching someone else do it and then improve it. Imitation learning consists of learning to perform a specific task from expert demonstrations [Ho and Ermon, 2016].

In complex high-dimensional problems, it would be unfeasible to train a model from scratch, as it would require an enormous amount of time and computational resources for the model to begin converging on a solution.

The idea behind imitation learning is to learn a good base model from the several demonstrations given by an expert, usually a human. The learned policy should be enough to successfully complete the given task even though it can be later optimized to account for human inaccuracies, errors, and deviations. In some cases, the resultant model is even better than the human demonstrations.

The learning method can be direct if it learns the policy directly from the demonstration in a process similar to supervised learning, or it can be indirect if it learns first a reward function and then derives the policy from it [Hussein et al., 2017].

3.3.1 Trajectory definition

Imitation learning needs expert demonstrations, which in the case of this dissertation are the aerobatic maneuvers trajectories performed by a human in the simulator.

The closest the demonstrations are to the intended trajectory, the better, but we can assume that they will not be perfect. There needs to be a way of combining all the demonstrations into one that comes close to the target.

In [Coates et al., 2008] the authors recognize that expert demonstrations may be optimal for a small portion of the trajectory (sub-optimal), and with a large number of sub-optimal demonstrations is possible to infer the ideal trajectory.

They propose an algorithm (EM algorithm) to generate the intended target trajectory from the expert demonstrations. The method is based on considering the demonstration a noisy observation of intended trajectory dislocated in time. So the algorithm performs a time-alignment of all the demonstrations, allowing an inference of the ideal trajectory.

The proposed algorithms also incorporate prior knowledge that improves the learning trajectory quality and decreases the convergence time. Position drift that can result from input imprecision is also accounted for in the algorithm.

In [Abbeel et al., 2010], a similar approach was used to extract the ideal trajectory but applied in helicopter aerobatic maneuvers. This paper compares the performance of the trajectory learned from demonstration, and a hand-specified trajectory [Abbeel et al., 2007] for the same maneuver. This new method obtained better results in flips and rolls and was able to perform a new maneuver, tic-toc, that the previous approach could not achieve.

3.3.2 Direct policy learning

Direct policy learning consists of directly mapping the states visited in the trajectory demonstrations to actions. Behavioral cloning is the simplest form of imitation learning as it is similar to supervised learning, which uses the demonstrations as ground truth.

[Rodríguez-Hernandez et al., 2019] were able to use behavioral cloning to make a micro aerial vehicles (MAV) quadrotor fly through a gate. Data was collected through a camera and had more than 2000 desired states to train the agent. This approach has, however, a significant drawback, which is a lack of robustness. This occurs because the agent only knows what to do when it sees a previously visited state. In variable scenarios, like real-life, where new situations can arise, this method does not know which action to perform.

A different way to implement behaviour cloning is by using neural networks that are able to map the high non-linear relation between the current aircraft state to the values for the actuators (control surfaces and thrust).

3.3.2.1 Artificial Neural Networks

There are several types of Neural Networks. An Artificial Neural Network (ANN) is a feed-forward multi-layer Neural Network. It consists of several nodes (Fig. 3.1), each one analogous

to biological neurons and that are connected to each other. This connection mimics the brain synapses in a way that each input connection has a variable weight that determines the changes during learning according to its importance for the final result. The node then combines all inputs and using a known activation function calculates the output.

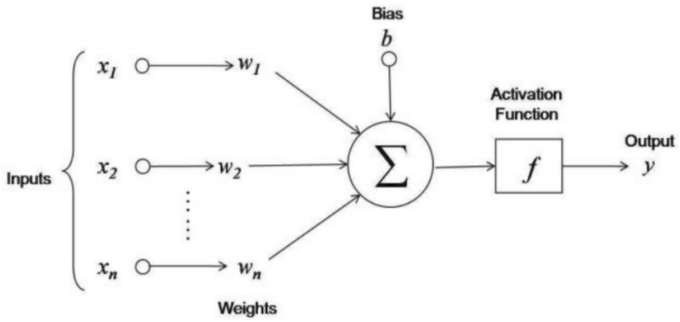


Figure 3.1: Artificial Neuron[1]

These nodes are arranged into layers that represent an abstraction of the input data. The nodes in each layer are connected to the nodes of the following layer (Fig. 3.2).

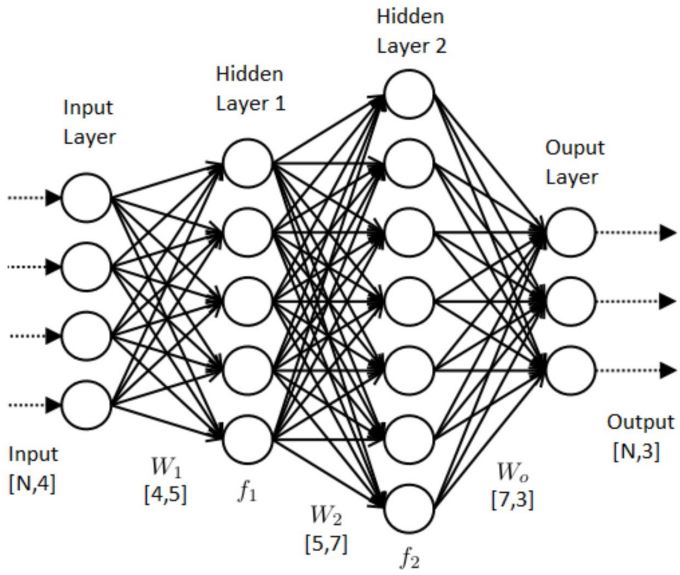


Figure 3.2: Artificial Neural Network[2]

In [Baomar and Bentley, 2016b] the authors present a method to perform tasks such as take-off, climb, and slow ascent using ANN's. The method consists of using four ANN's, that outputs values for the elevator, aileron, rudder and throttle. Given that the control system will be used to perform tasks in situations proximal to the aircraft's stability state and thus the mapping function more easily linearizable, the authors decided to use a single hidden layer for each ANN. The obtained results shows that the control method was able to imitate human behaviour both in calm and stormy weather conditions and generalizing for unseen conditions even though there were used limited examples (one for each ANN). It is also worth noting that despite the human pilot

demonstrations in thrust control presenting some oscillations, the control system was able to learn how to control the thrust smoothly.

A similar approach was used in [Baomar and Bentley, 2016a] but this time each ANN encodes a model for a different flight behavior. A meta-scheduling program is used to decide which model to used at a given condition. The model is trained for situations such as engine failure, rejected take off (RTO), and emergency landing. The topology for the ANN's used were composed by only one hidden layer based on the fact that the problems that require more than one are scarce [Heaton, 2008]. Also as a rule-of-thumb the number of nodes in the hidden layer must be less than or equal to twice the size of the input layer. During training the output values of each ANN were compared to the human pilot demonstration by calculating the mean squared error (MSE) stopping when a achieving an error below 0.001.

The experiments show the ability of Supervised Learning with Artificial Neural Networks to learn low-level control tasks and even performing better where fine control is needed.

[Shukla et al., 2020] includes a new technique to train an ANN model in order to increase its robustness. One problem that behavior cloning has is that in the presence of a unvisited states the model accuracy decreases. To address this problem the authors use the Dataset Aggregation (DAgger) algorithm which consists in introducing perturbed data to the training process.

In this case the authors already had a Guidance Navigation Control (GNC) system that acted as expert. This allows for a much faster data generation. The output from the ANN is fed to the aircraft simulator that outputs its new state. This new state include some small deviations from the desired behavior. So it is labeled by the GNC policy and added to the training dataset.

This article proposes a way to control a fixed-wing aircraft to follow a path defined by four waypoints by directly mapping the aircraft state to control surface values.

The inputs used a ANN with one hidden layer with 35 neurons and as inputs the North-East-Down (NED) distances for each waypoint, the velocity, roll angle and pitch angle outputting the throttle, elevator, aileron and rudder values.

In the article results show that standard supervised learning techniques are not able to train an end-to-end ANN autopilot to fly a fixed-wing aircraft through a path defined by waypoints, having only the GNC McDAgger approach yield good results.

3.3.2.2 Recurrent Neural Networks

We can see maneuvers as changes in position over to time. It doesn't only depends on the current state but also on previous states.

Feed-forward neural networks like ANNs are not ideal for sequential data or time series since it's output is independent from previous inputs. To address this problem a new type of neural networks emerged, the recurrent neural networks (RNN) (Fig. 3.3). Each RNN node includes its output in its input and thus being capable of acting as a memory cell. This class of neural networks are considered the state of the art in time sequential data being used by Apple's Siri and Google's voice search [Kumar et al., 2020].

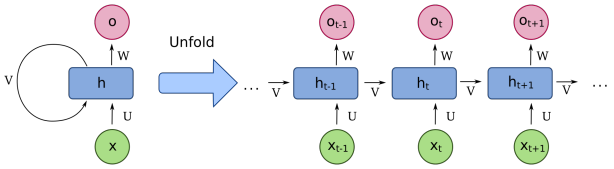


Figure 3.3: Recurrent Neural Network[3]

Long short-term memory (LSTM) originally proposed by Hochreiter & Schmidhuber is a specific type of RNN that can learn long-term dependence information of sequential data. A LSTM neuron has three different gates: the input gate, the forget gate and the output gate that controls the flow information and cell state. In classic RNN during training the gradients are back-propagated can easily tend to zero or infinity due the repeated computations involving the gradient called vanishing gradient problem. LSTM tries to solve this problem by allowing the gradient to flow unchanged [Pascanu et al., 2013].

[Li et al., 2019] use LSTM to develop a flight attitude control model in an air combat situation. To do that, they collected 200 demonstrations, each with 47 variables describing the aircraft state and battlefield situation. The data was then normalized to prevent influence of the data dimension on the model's prediction accuracy. In the experiments the authors used the Mean Squared Error (MSE) and Mean Absolute Error (MAE) between the output value and the real value to measure the performance evaluation index for the model. The LSTM model used also has only one hidden layer, like in many other papers, with the number of node determined experimentally given that it is a determinant parameter for the model accuracy. The proposed model has better prediction accuracy and convergence performance than traditional recurrent neural network.

3.3.3 Indirect policy learning

Contrary to the previous type, indirect policy learning tries to learn "why" the demonstrator is performing a specific action when seeing a specific state. This is done by understanding the reward function behind the demonstrated behavior and thus achieving the objective.

[Abbeel et al., 2010] apply this method to perform aerobatics autonomously in a Radio Control (RC) helicopter. They were the only ones to perform highly complex maneuvers in real hardware from all the literature reviewed. The successfully performed maneuvers were "flips, in-place rolls, loops and hurricanes, and even auto-rotation landings, chaos, and tic-tocs".

They start by building a dynamic model that describes the helicopter behavior. This is done by defining some physical constraints and using the demonstrations to tune the parameters to develop an accurate model. Each maneuver has a specific set of demonstrations used to create a target trajectory from its suboptimal portions.

[3]Retrieved from https://en.wikipedia.org/wiki/Recurrent_neural_network

All this data was then fed to a learning algorithm that outputs the rewards function for that maneuver. From that, it was only needed to apply an optimal control method to find the optimal policy that maximizes the reward.

This method proved to be more robust and provided better results than the direct approach.

Chapter 4

Proposed solution

This chapter presents some relevant decisions in the problem definition, specifically chosen aerobatic maneuver, input state variables, and output control variables. It also describes the approach used to solve the problem that we propose to solve at a high level. Includes methods used, data collected, and many other components that ultimately compose the final solution.

4.1 Aerobatic maneuvers selection

During section 2.2 it was presented several aerobatic maneuvers that are common in air races and air combat. From all the options presented, the Immelmann turn was the chosen one. It is a complex maneuver composed of three simple portions. First, at a pitch angle and bank angle of 0 degrees, the aircraft starts to perform a half loop to the desired altitude. The elevators must deflect upwards to create more lift force in the front of the aircraft than the back, causing it to rotate in the lateral axis and thus increasing the pitch until it reaches 180 degrees. After this maneuver, the aircraft should be at the desired altitude with a bank angle of 180 degrees, that is, upside down. To return to a normal position, a 180 degree roll is done.

This maneuver is very similar to other common aerobatic maneuvers like the half-cuban eight and the split. As such, it is a good starting point to extend the model's capabilities later. In the half-cuban eight, instead of maintaining the altitude after the half-loop, it continues descending until it reaches the starting altitude. In this case, the roll maneuver is done during the descending part of the maneuver. The split is an upside-down Immelmann turn, whose final altitude is below the initial altitude. From the same initial state, it performs the 180 degree roll first and then the half loop.

4.2 Learning method

Before deciding on which method to choose so that an agent learns to perform aggressive aerobatic maneuvers, it is advisable to remember the problem's characteristics in question.

Firstly, the agent will act on fixed-wing aircraft in the FSX simulator. The fact that the aim is to control an aircraft in a simulator and not in the real world makes data acquisition much more convenient, faster, and with greater volume.

In a simulator, it is also easier to carry out trial and error experiments, and as such, the learning process becomes more efficient. However, it is worth noting that the FSX has a maximum simulation speed limited to 16x real-time and that the higher this rate is, the lower the accuracy of the simulation, which can add other types of unexpected disturbances.

Furthermore, the aircraft will be subject to different environmental conditions. FSX can simulate different weather conditions, as well as atmospheric pressure, temperature, and variable wind. All of this can affect the system's dynamics, and as such, the agent must be immune to these disturbances.

This simulator has several aircraft models, from fighter jets to aerobatic aircraft that, of course, behave differently. The agent should be able to generalize to various aircraft types without having to start the learning process again.

It would also need to allow to add maneuvers intuitively without requiring expert knowledge of physics and aerodynamics.

There is no perfect solution, but in this case, behavior cloning using a neural network (ANN OR LSTM) seems to be the best approach due to several factors. First, it is simple to train. Behavior cloning is similar to supervised learning, so to train the neural network that maps the input states to output control only needs the state and action from the human expert demonstrations. There is no need for mathematical aircraft dynamical models that increase the complexity and change considerably with the aircraft type in question. All this being said, we can also predict some problems. The robustness to new circumstances is very dependent on the neural network capability of generalizing, and it is usually a problem with this method. We know from the literature that we need large quantities of demonstration data in several different conditions to minimize this problem.

4.3 Architecture

The solution architecture is divided in three main applications. The first one, a data collector, that is responsible for collecting the human demonstrations for the immelmann turn 4.1. The second one that takes the demonstrations, prepossesses them and trains a model with them as input.

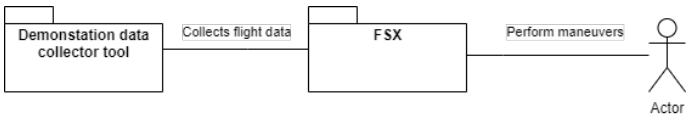


Figure 4.1: Data collector architecture

Finally, a third application that uses the model from the previous one to predict the best position for the elevator and aileron for a given state 4.2.

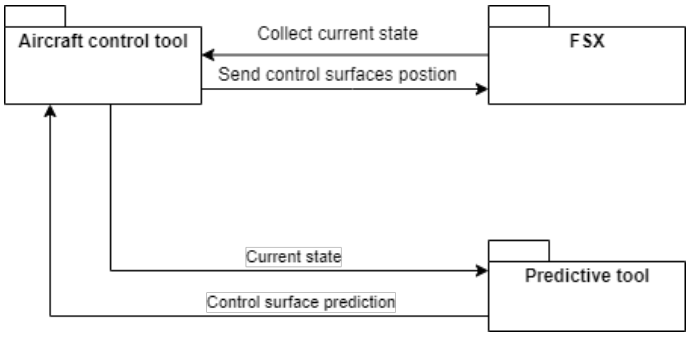


Figure 4.2: End-to-end neural control architecture

4.4 Collected variables

As mentioned before, FSX makes available several variables and events through Simconnect. However, only a small subset is of interest for our purposes.

The following tables do not list, exhaustively, all variables existent but rather only the ones that influence substantially the dynamic behaviour of the aircraft. Whether they are related to control aircraft control or environment conditions.

In this list are also pinpoints the variables later used as features input for the behavior cloning model. All information presented in these tables were retrieved from the Microsoft references [1] [2].

4.4.1 Environment

Table 4.1 shows FSX ambient variables that might affect the aircraft dynamic during flight. All the variables listed are read-only.

[1] Event listing provided in https://docs.microsoft.com/en-us/previous-versions/microsoft-esp/cc526980(v=msdn.10)

[2] Variable listing provided in https://docs.microsoft.com/en-us/previous-versions/microsoft-esp/cc526981(v=msdn.10)

Table 4.1: FSX environment state variables

Variable	Description	Settable	Use
AMBIENT DENSITY	Ambient density	N	
AMBIENT TEMPERATURE	Ambient temperature	N	
AMBIENT PRESSURE	Ambient pressure	N	
BAROMETER PRESSURE	Barometric pressure	N	
SEA LEVEL PRESSURE	Barometric pressure at sea level	N	
STRUCT AMBIENT WIND	X (latitude), Y (vertical) and Z (longitude) components of the wind.	N	

4.4.2 Aircraft engine control

Table 4.2 shows thrust controls via events which are write-only operations. Table 4.3 presents a variable used to read/write the throttle position.

Table 4.2: FSX engine control events

Event	Description	Use
KEY_THROTTLE_FULL	Set throttles max	
KEY_THROTTLE_CUT	Set throttles to idle	
KEY_THROTTLE_SET	Set throttles exactly (0- 16383)	

Table 4.3: FSX engine control variables

Variable	Description	Settable
GENERAL ENG THROTTLE LEVER POSITION:index	Percent of max throttle position	Y

4.4.3 Aircraft control surfaces

Table 4.4 list events capable of controlling the most relevant control surfaces on an aircraft.

Table 4.4: FSX events for aircraft control surfaces

Event	Description
KEY_FLAPS_UP	Sets flap handle to full retract position
KEY_FLAPS_1	Sets flap handle to first extension position
KEY_FLAPS_2	Sets flap handle to second extension position
KEY_FLAPS_3	Sets flap handle to third extension position
KEY_FLAPS_DOWN	Sets flap handle to full extension position
KEY_ELEV_UP	Increments elevator up
KEY_ELEV_DOWN	Increments elevator down
KEY_ELEVATOR_SET	Sets elevator position (-16383 - +16383)
KEY_AILERONS_LEFT	Increments ailerons left
KEY_AILERONS_RIGHT	Increments ailerons right
KEY_AILERON_SET	Sets aileron position (-16383 - +16383)
KEY_RUDDER_LEFT	Increments rudder left
KEY_RUDDER_CENTER	Increments rudder left
KEY_RUDDER_RIGHT	Increments rudder left
KEY_RUDDER_SET	Sets rudder position (-16383 - +16383)
KEY_SPOILERS_ON	Sets spoiler handle to full extend position
KEY_SPOILERS_OFF	Sets spoiler handle to full retract position
KEY_SPOILERS_SET	Sets spoiler handle position (0 to 16383)

4.4.4 Aircraft state

In order to learn how to control a plane is essential to know the aircraft state over time. Table 4.5 list relevant variables about its position, orientation, velocity and control surface deflection.

Table 4.5: FSX aircraft state variables

Variable	Description	Settable	Use
PLANE PITCH DEGREES	Pitch angle	Y	Y
PLANE BANK DEGREES	Bank angle	Y	Y
PLANE HEADING DEGREES MAGNETIC	Heading relative to magnetic north	Y	
VELOCITY BODY X/Y/Z	True lateral, vertical and longitudinal speed respectively relative to aircraft axis	Y	Y
ROTATION VELOCITY BODY X/Y/Z	Rotation relative to aircraft axis	Y	Y
ACCELERATION BODY X/Y/Z	Acceleration relative to aircraft axis, in east/west, vertical and north/south direction respectively	Y	
PLANE ALTITUDE	Altitude of aircraft	Y	Y
GROUND ALTITUDE	Altitude of surface	N	
RUDDER POSITION	Percent rudder input deflection	Y	
ELEVATOR POSITION	Percent elevator input deflection	Y	Y
AILERON POSITION	Percent aileron input left/right	Y	Y
FLAPS AVAILABLE	True if flaps available	N	
FLAPS HANDLE INDEX	Index of current flap position	Y	
SPOILERS HANDLE POSITION	Spoiler handle position	Y	

4.4.4.1 Miscellaneous

Table 4.6 and 4.7 present the events and variables respectively that control and track brake engagement, gear position and fuel consumption.

Table 4.6: FSX events for miscellaneous tasks

Event	Description	Use
KEY_BRAKES	Increment brake pressure	
KEY_GEAR_UP	Sets gear handle in UP position	
KEY_GEAR_DOWN	Sets gear handle in DOWN position	

Table 4.7: FSX aircraft state variables

Variable	Description	Settable	Use
GEAR POSITION	Position of landing gear	Y	
FUEL TOTAL QUANTITY WEIGHT	Current total fuel weight of the aircraft	N	
ESTIMATED FUEL FLOW	Estimated fuel flow at cruise	N	

Chapter 5

Implementation

This chapter describes the implementation process. It talks about all parts involved in autonomously maneuvering an aircraft and the reasoning behind each one. It also presents several other attempts that ended up not being as successful.

5.1 Demonstration data collector tool

In the previous chapter, we saw that the best approach to this thesis problem is using behavior cloning. However, one drawback that we saw during the literature review is that this method requires a significant amount of data to generalize complex control tasks.

Having this in mind, we need a tool to easily and quickly collect human pilot demonstrations. It also needs to output data that is easily processed by the learning method. This tool would then be shared with the community to get diverse and large amounts of data.

Similar to what was done on the platform developed by LIACC (mentioned in section 1.1) for uniformity, ease of integration, and documentation availability, this tool will be developed in C# using the SimConnect library to communicate with FSX.

This tool was developed using the graphical interface Windows Forms for .Net framework due to its simplicity to develop.

The process starts with the struct definition with the variables chosen in section 4.4 to be relevant for our case. This data already represents the input state, in the case of the environment and aircraft state variable, and the ground Truth represented by the control surfaces and thrust position. There are several rates at which Simconnect can pull data. Initially, the pulling was made at each sim frame which provided a more or less constant time interval between each collection. However, later we will see that predicting and sending control data takes up to double the time it takes to collect data. Because of this incoherence, the practical results were not good even though, during learning, the MSE has small.

The approach that ended up being used consists of an infinite loop where the data collector method is called only once. This allows more fine control when the data is collected and allows to within the loop create a random sleep time between calls of the same order of magnitude. The

delta time between calls is also stored and will be used as input to our neural network. Because now the delta time is not constant, the neural network will be able to generalize and take into account this parameter.

The struct data is then saved in a list, not on a file, to prevent i/o delay during the data collection. In the end, all data is written to a CSV file using the CsvHelper library. This format was chosen because it is widespread in all programming languages and therefore offered more flexibility when choosing where to train the neural network.

5.2 Learning model

Before going into detail on the final model used to train our neural network, it is worth noting several other attempts used to create a functional maneuver controller.

5.2.1 Sparse Identification of Nonlinear Dynamics (SINDy)

In the first attempt to create a controller for the aircraft, a data-driven system identification algorithm was used in combination with a model predictive controller.

SINDy is an algorithm that identifies a non-linear system based on measured data. The values and variable selection are made with the idea that few terms characterize a system behavior in most systems. So, terms that do not contribute significantly to the model accuracy will not be considered in this method [Kaiser et al., 2018]. The value of how much accuracy gain is significant is asked to the user to specify.

This makes the model more robust to noise in the training data and even more general even when trained with few data compared to Neural Networks.

In this method, the user also needs to specify candidate unitary non-linear functions that he thinks will describe the system better.

The implementation starts by identifying the dynamic system. For that PySINDy, a python package for system identification was used. Given a list of state variables, PySINDy tries to infer governing equations in the form of a dynamical system. The threshold parameter defines the minimum threshold of the variable importance to be included in the final equations. The final result is a set of equations, one for each input variable determines how that variable changes according to the other state variables. Unfortunately, all the learning history data is encapsulated by PySINDy, so there is no way to analyze metrics like loss per batch which are essential to see if a model is making progress during learning.

Having the previous governing equations, we can calculate the displacement of the control surfaces using model predictive control (MPC). MPC is an online optimization method and, unlike Linear Quadradic regulator (LQR), can optimize non-linear system dynamics.

After testing, the SINDy-MPC method was revealed to be more suitable for smooth slow dynamics but not aerobatic maneuvers, which is the focus of this thesis. SINDy cannot fully capture the aircraft behavior in unstable states, and defining objectives in MPC to define an aerobatic maneuver trajectory is not trivial and requires manual definition.

5.2.2 End-to-end linear model

The second method involves using a linear model to describe the plane behavior. Initially, we were only interested in testing the capabilities of this approach. That is why we only try to control the elevator position.

So, to start, we collected several minutes of flight data using the *Demonstration data collector tool* containing all the relevant flight data, as seen in section 4.4. We use Keras with TensorFlow backed to create an optimized linear regression model. It consists of two layers. The first one is a Normalization layer used to transform the input data to approximate a standard normally distributed function (Gaussian with zero mean and unit variance). This is especially useful in data with very different orders of magnitude as it can significantly improve the performance of several machine learning algorithms.

The second layer is a regular densely connected neural network layer with one node. For the optimizer, we use Adam with a learning rate of 0.001, and the mean squared error gives the loss value.

In the end, we can extract the equation that gives the elevator value from the input state by reading the Dense layer kernel values.

Despite during training, the loss value decreased, it was still high for this application, which became apparent when implemented in the aircraft controller.

5.2.3 ANN and LSTM

All the previous attempts had a common problem, the model was not complex enough to describe the aircraft behavior correctly. This time, the idea is to use neural networks to directly predict the control surfaces' position from the aircraft and environment state. In other words, an end-to-end controller.

Two different types of neural networks were tested, a simple Feed-forward neural network ANN and a recurrent neural network LSTM. This decision is strongly influenced by previous work in aircraft control reviewed in section 3.3.2.

Before even starting applying the new learning methods, it is necessary to aggregate a quality dataset. Using *Demonstration data collector tool* there are collected 30 human demonstrations of an Immelmann turn in different environments and with different aircraft. Unfortunately, the number of demonstrations is meager when we consider that the problem in question is high dimensional and non-linear. All demonstrations in the dataset need to be close to perfect so that the model learns correct maneuver patterns instead of noise. This is also the reason behind collecting data in a different condition to help the model generalize. With a limited number of human demonstrators none of them being an expert it becomes challenging to generate an extensive dataset. This constitutes a significant disadvantage of using this approach. The input features in the dataset are then normalized using the scikit-learn MinMaxScaler. It takes each feature and maps it to a value between [0,1] if positive and between [-1, 1] if negative, preserving the value distribution.

Even though the dataset is now normalized, it cannot be used directly as input to any neural network. Using the dataset as is, the model becomes incapable of doing any type of maneuvers. This behavior can be attributed to one of the following reasons. First, some collected variables are not independent, being the world velocity vector and the body velocity vector are examples of this. The other reason is that, in this configuration, the problem in question is a high-dimensional small dataset problem. Professor Yaser Abu-Mostafa says in one of his lectures [1]that the sample size should be at least about ten times the dataset dimension. Otherwise, the network will overfit as it has no sufficient data to correlate the feature impact on the output.

With 14 input features, there need to be at least 140 samples in the dataset, which are 4.6 bigger than ours. The solution is to reduce the number of input features and use only those that impact the model performance. Based on generic aerodynamic equations reviewed in section 3.1 and experimentation, the final feature selection was the following.

For controlling the elevator position the features used where *angle of attack*, *pitch angle*, *bank angle*, *vertical speed*, *longitudinal speed*, and *distance to target altitude*.

To control the aileron position the features used where *angle of attack*, *pitch angle*, *bank angle*, *elevator position*, *roll speed*, and *distance to target altitude*.

Finally, the dataset was divided into two groups, training set (80%) and testing set (20%). Later from the training set is extracted validation data that provides feedback on each epoch's changes. Validation set and testing serve entirely different purposes. The validation set is used as feedback during training, while the testing set is only used as a performance evaluation and never during training.

The implementation of both ANN and LSTM models was done using Keras with the TensorFlow backend. Because Keras provides a high-level abstraction, both networks structure were very similar except for the hidden layer.

As a starting point for the simulation state, there is the input layer. It is composed by 6 nodes, the same number as the input features. The hidden layer has the same number of nodes as the input layer. It was determined experimentally that this configuration offers the best performance.

In the ANN, the hidden layer is just a conventional, deeply connected neural network layer. It multiplies the node's input with the kernel weight, shifts its value with a bias, and uses an activation function to transform the obtained value into an output. The kernel and bias values are learned throughout the NN training. For the activation function, it was used the Rectified Linear Unit (ReLU) that prevent the gradient saturation problem in which, after several training iterations, the neuron output value tends to the low or high extremes. Using this activation function significantly accelerates the convergence of stochastic gradient descent compared to other activation functions like sigmoid and tanh [Krizhevsky et al., 2012].

In the LSTM, the hidden layer also has 6 nodes. It was determined experimentally that this configuration is able to capture the pretended behavior without over-fitting.

During training, the loss is evaluated using the Mean Squared Error (MSE) of the difference between the elevator/aileron predictions and the elevator/aileron values from demonstrations.

[1]Retrieved from https://www.youtube.com/watch?v=Dc0sr0kdBVI

The adaptive moment estimation (ADAM) is used as the optimizer to reduce the value of the loss to its minimum.

The number of batch size, hidden layer nodes, and learning rate were determined experimentally during training using hyper-parameter scanning and optimization. For the number of epochs, an early stopping method monitors the validation set loss, and if there is no improvement, the training stops preventing over-fitting.

From the learning process, for each neural network trained, it outputs two Hierarchical Data Format (.H5) files containing the model, one to control the elevator and another to control the aileron. It also outputs the normalization weight used to normalize the input data. This information is posteriorly loaded by another tool responsible for controlling the aircraft.

5.3 AI Maneuver control tool

There are several ways to implement the AI Maneuver control tool based on performance, homogeneity, and complexity.

One way is to have a single python program that does all the main three functions.

Collect state data using Simconnect from FSX, serving as input to the predictive neural network model, which produces the control data that is later sent via Simconnect to the FSX.

This approach, however, has some implementation problems. First, the Simconnect library is compiled to a 32 bits architecture, whereas the TensorFlow framework uses a 64 bits architecture. Therefore, these two modules cannot run on the same python instance. This problem can be solved by using two python instances: one 32 bits tool that collects the flight data and writes the control data to the FSX, and another 64 bits tool that uses the trained model to predict the control data. These two tools communicate via sockets with the ZeroMQ implementation.

Another problem that became apparent later was that the Simconnect library in python, unlike in C# and C, cannot pull the variables of interest "simultaneously". Instead, it pulls the variables one by one. As a result, it takes a lot more time, up to 10 times slower than the C# equivalent. Furthermore, as if it were not enough, the variables are often from different instants rather than representing a snapshot from that specific moment.

To solve all these problems, we end up using two programs C# and python, for FSX communication and model prediction, respectively. The following subsections describe in more detail the final implementation.

5.3.1 Aircraft control tool

The *Aircraft control tool* is very similar to the *Demonstration data collector tool*. It uses the same principle to collect the flight state data from FSX. This data is then pre-processed to the correct shape, order, and data structure determined by the predictive model (section 5.2.3). To send and receive data, ZeroMQ is used, which is an asynchronous messaging library that provides a message queue to concurrent processes. The predictive model, whose function is described in the following

subsection, receives the input features and sends back the output array with the aileron, elevator, rudder, and throttle values.

Immediacy after receiving the response, the *Aircraft control tool* sends to FSX the control surfaces position absolute values and throttle level again using Simconnect.

One complete iteration takes around 60ms, resulting in a rate of 16 new controls every second.

5.3.2 Predictive tool

The predictive tool is a straightforward tool that uses the learned model and the normalization input weights in section 5.2.3 to predict the actuator's values from the aircraft state. First, it opens a TCP socket in the localhost using the Pyzmq package, a python implementation of ZeroMQ. During the communication, this tool will act as a server and the *Aircraft control tool* as a client. After loading the trained models and normalization input weights, it starts waiting for requests. When it receives a new request, it normalized the input and sent back the output from the prediction.

To keep it simple, we use python to run Keras with Tensorflow backend the same way it was done during the model training despite exiting an unofficial implementation of the Tensorflow API in C# called TensorFlow.NET (TF.NET) that would allow us to unify the control and prediction in one C# application. The reasoning behind this is because the platform on which this thesis is being developed is constantly being extended with new features every year. Therefore, it becomes necessary to implement a stable solution that will not be discontinued in the near future.

The model prediction is the task that takes the most time during the aircraft control process. So, to have a more fluid motion and better response, we need to optimize the TensorFlow the best we can. Tensorflow can run on CPU or GPU, with the latter offering the performance. Neural networks, at low-level, can be represented by operations between matrices that can run in parallel in multiple cores. So it comes as no surprise that a GPU with a number of cores around 500 - 1000 offers better performance than a CPU with 4-8 cores.

Unfortunately, not all GPUs are supported. Tensorflow requires GPUs with Compute Unified Device Architecture(CUDA), an API created by Nvidia to facilitate parallel computations.

The machine used during this thesis did not support CUDA, which will undoubtedly affect our controller's performance. So, from now on, we will focus on how to improve CPU performance. Tensorflow offers a compiler option to activate Advanced Vector Extensions(AVX), which uses new floating point registers called YMM to perform a Single Instruction on Multiple pieces of Data (SIMD) operations. AVX is not enabled by default because not every processor supports it. Luckily our machine does, and after making these changes, we saw a reduction in each iteration from around 100ms to 60ms, an improvement of 40%.

Chapter 6

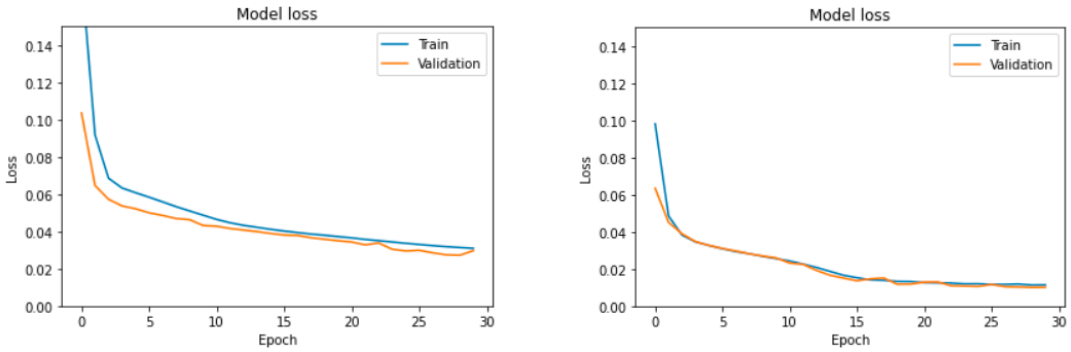
Results

This chapter compares the results between the two most promising approaches, the ANN and LSTM models, with the demonstration data. It presents information regarding model loss, elevator/aileron position, and trajectory path.

The dataset for training both models is the same, and its characteristics are specified in the section 4.4. It consists of a sample of 30 demonstrations of an Immelmann Turn using the Boeing F/A-18 aircraft in different weather conditions, different airports, and different target altitudes, which is the final altitude after the maneuver. This parameter can have any value as long as it is above the initial altitude, a constrain from the maneuver itself.

6.1 Model training

The figure 6.1 indicates that in both ANN and LSTM models, there are no signs of underfitting when the validation loss is smaller than the training loss, nor overfitting, that happens when the validation loss starts to climb and becomes significantly higher than training loss. In both cases, the validation and training loss is very similar and decreases over time, indicating that the models can capture patterns from the dataset.



(a) ANN

(b) LSTM

Figure 6.1: Elevator model train and validation loss

The comparison between the validation loss of each model is presented in the figure 6.2. The validation loss for both cases is very small due to the elevator position domain being also very small from -1 to 1. Anyway, we can see that LSTM loss is 0.01 where the ANN loss is 0.03, providing a decrease in loss of around 66%.

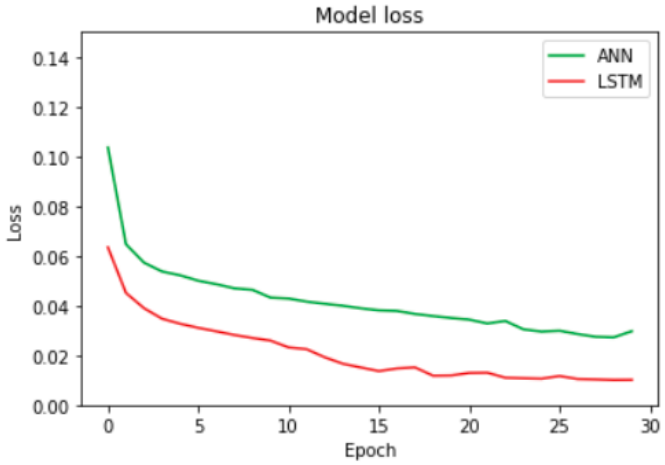
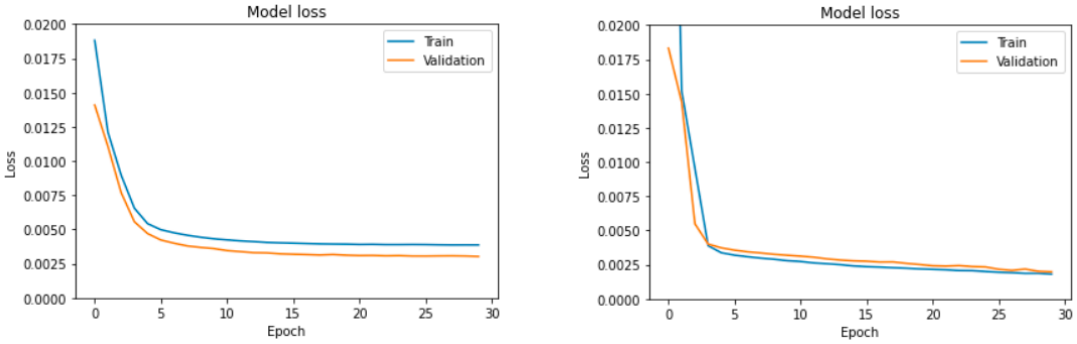


Figure 6.2: ANN vs LSTM: Elevator validation loss

The figure 6.3 shows how the loss evolves during the aileron model training. In the ANN model, we can see that validation loss is lower than the training loss indicating a small underfitting. This model would probably benefit from more epochs. The LSTM aileron model loss history is similar to the LSTM elevator model. We can see a consistent decrease with the training and validation loss converging to the same point.



(a) ANN (b) LSTM

Figure 6.3: Aileron model train and validation loss

Once again we see that LSTM is able to get lower validation loss that the ANN, 0.00205 against 0.00308 providing a 33% decrease in loss (Fig. 6.4).

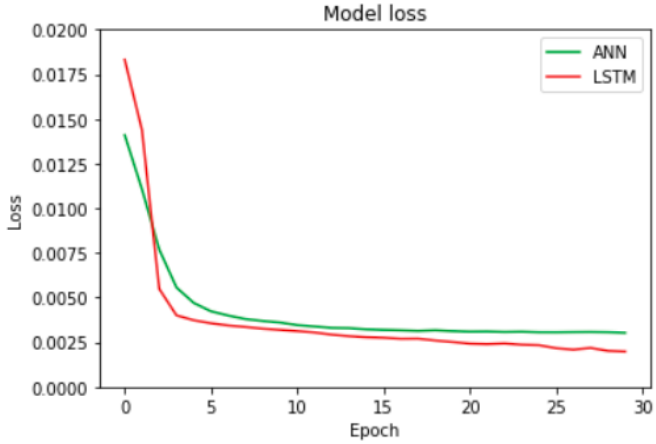


Figure 6.4: ANN vs LSTM: Aileron validation loss

6.2 Model testing

The original dataset was divided into two parts, 80% for training and 20% for testing. The figure 6.5 shows how the elevator position varies thought time on the test set. We can see that the LSTM model does a much better job imitating the ground truth than the ANN, which is expected due to lower validation loss during training. It is worth noticing that in the last portion, LSTM reaches negative values for the elevator to compensate for the higher peak.

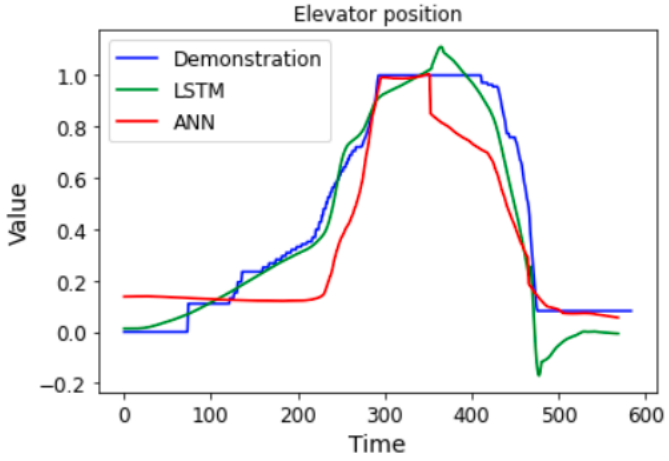


Figure 6.5: Demonstration vs ANN vs LSTM: Elevator position

Similar to what happened in the elevator model, the LSTM model does a better job of imitating the human demonstration than the ANN model. We can see that in the lowest valley, ANN overshoots without compensating later with a positive aileron value.

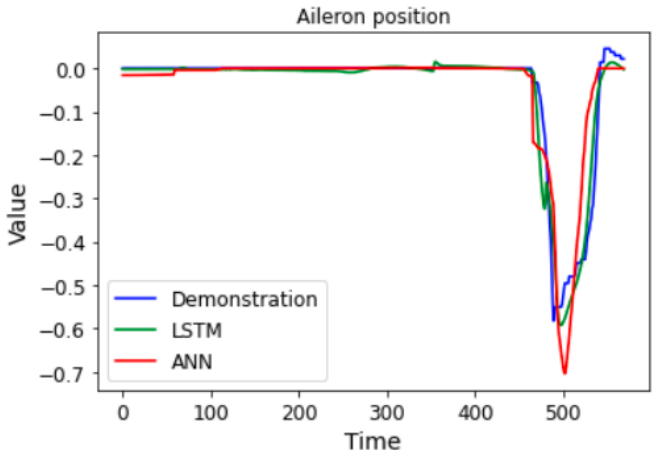


Figure 6.6: Demonstration vs ANN vs LSTM: Aileron position

6.3 Experiment

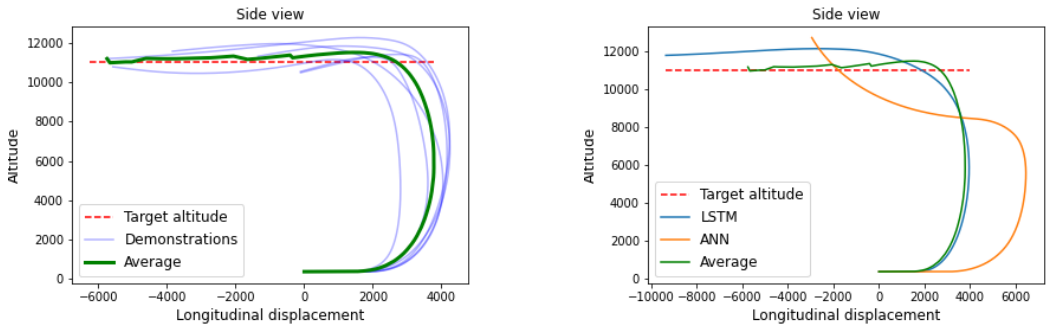
In order to test in practice the models' behaviour and how they compare to a human, an experiment was designed. It consists of performing an Immelmann Turn with a target altitude of 11000 feet (the choice of this value was arbitrary with the condition that it needs to be superior than the initial altitude). It was used the Boeing F/A-18 aircraft in the airport Lisbon Portugal, in fair weather, which is characterized by high-level clouds and light winds. For consistency, these conditions will be used for the human demonstrator and for the ANN and LSTM models.

For comparison purposes, five new human demonstrations were collected with the target altitude of 11000 feet in mind. It is worth noting that these new demonstrations were not used by the models as input. Otherwise, it would significantly change the models' behavior to that specific altitude which is not intended. The idea of this experiment is to show how the models behave with any target altitude without further tuning.

Figure 6.7 shows the side projection of the Immelmann turn. This view is particularly influenced by the elevator position. In the left figure is noticeable that human demonstrations are very inconsistent even though all reach the 11000 feet mark.

On the right side, the first thing to notice is that the ANN elevator model is incapable of performing the half-loop portion of the maneuver correctly. It ends the half loop at an altitude of 8000 feet, following a steep climb that overshoots the desired 11000 feet.

Despite using the same dataset and input features, the LSTM elevator model outperforms the ANN model, performing an almost perfectly smooth half loop to a height of 12000 feet. After the half-loop, the LSTM model slowly starts to descent to the target 11000 feet altitude.

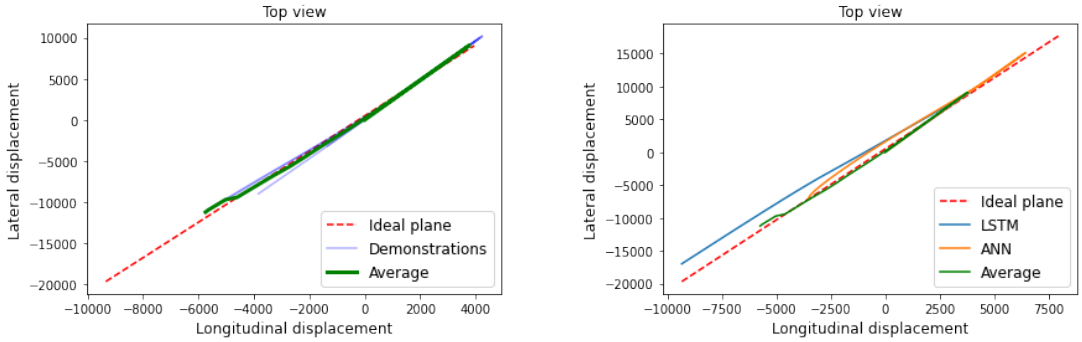


(a) Demonstration average (b) Method comparison

Figure 6.7: Side view caparison

One characteristic of the immelmann turn is that all points from the maneuver should describe a vertical plane, that viewed from the top should describe a straight line. If it does not happen it means that the roll portion of the maneuver is not being done correctly either because it is not fast enough and thus during the roll the lift on the wings causes the aircraft to dislocate laterally, or the final bank angle after the roll is not zero.

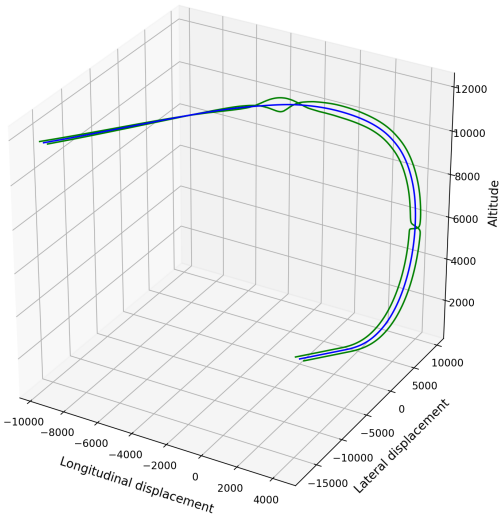
The figure 6.8 left shows that the human demonstrator is able maintain straight line when performing the maneuver. On the right however we see that the LSTM trajectory goes straight until the roll then deviates a little and goes straight again, meaning that the roll maneuver should be performed faster. This time the ANN trajectory is more similar to the ideal trajectory indicating that the ANN aileron model performed better that the LSTM aileron model.



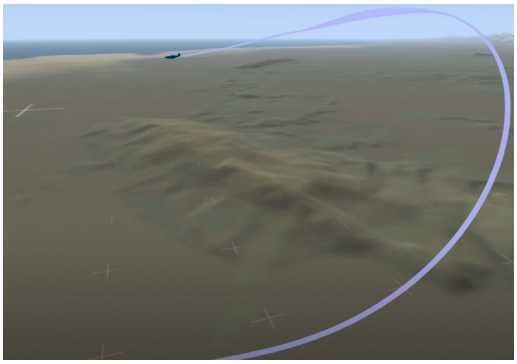
(a) Demonstration average (b) Method comparison

Figure 6.8: Top view caparison

The figure **??** shows, in 3D, on the left, the trajectory described by the aircraft using the LSTM model and on the right the immelmann turn trajectory. As we can see the LSTM model was capable of capturing the immelmann turn characteristics making both trajectories are very similar. The initial smooth half loop with a 180 degree roll on the top maintaining then the same altitude.



(a) LSTM trajectory



(b) Immelmann turn trajectory

Figure 6.9: Trajectory comparison

[0]Retrieved from http://steampunkaviatrix.blogspot.com/2010/05/history-of-immelmann-turn.html

Chapter 7

Conclusions and Future Work

Performing complex maneuvers on a high-dimensional non-linear system as an aircraft is not an easy task. The traditional approaches use mathematical modulation of the aircraft dynamics and are not flexible enough to easily extend to another aircraft and maneuvers. There are several alternative approaches to this problem presented in the literature. In the state of the art are analyzed the strengths and weaknesses of methods of reinforcement learning, inverse reinforcement learning, system identification with non-linear optimizers, and behavioral cloning. In a context where the autonomous model needs to be versatile enough to cope with changes in the environment condition and various aircraft models, behavioral cloning is chosen as the best approach. The model used in this dissertation has a comparable performance with a traditional approach in an Immelmann turn, without the need of specifying the aircraft aerodynamic equations or objective equations, and thus become easier to add new capabilities to the autopilot. Regarding non traditional methods, it demonstrates its potential by performing a complex maneuver instead of simpler tasks such as level flight, take off and land.

Using an end-to-end long short-term memory (LSTM) neural network, it was able to perform an Immelmann turn to the desired altitude in a smooth trajectory with only 30 samples, with an error in the final altitude of 7%. The end-to-end artificial neural network (ANN) was not capable to perform such maneuvers with the demonstrations available.

Both these models follow a behavior cloning approach, and, as such, have well-known limitations. One of them is the lack of robustness in non-visited states during training. And the other is the poor explainability of the neural network decisions, and, because of that, is not ideal for sensitive operations in the real world.

Compared with other approaches in the literature this method is able to

More extensions can be done on top of this work. The Immelmann turn can be extended to other aircraft using a broader dataset. Some experiments were done in the *Extra 300S* aircraft revealing that the same LSTM model can control more than one aircraft.

The model could also be extended to similar aerobatic maneuvers like half-cuban eight, and split using transfer learning techniques. More distinct aerobatic maneuvers, like knife-edge turns and canopy rolls, would probably require a new feature selection and model tuning.

After these extensions, it would be interesting to have a meta scheduler responsible for monitoring the simulation state and, depending on the objective invoking different maneuvers models with adequate parameters.

Bibliography

[Abbeel et al., 2010] Abbeel, P., Coates, A., and Ng, A. Y. (2010). Autonomous helicopter aerobatics through apprenticeship learning. *International Journal of Robotics Research*, 29(13):1608–1639. doi: 10.1177/0278364910371999.

[Abbeel et al., 2007] Abbeel, P., Coates, A., Quigley, M., and Ng, A. (2007). An application of reinforcement learning to aerobatic helicopter flight. In Schölkopf, B., Platt, J., and Hoffman, T., editors, *Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems, December 7 2006, Vancouver, Canada,*, volume 19, pages 1–8. MIT Press. url:https://proceedings.neurips.cc/paper/2006/file/98c39996bf1543e974747a2549b3107c-Paper.pdf.

[Abouheaf et al., 2018] Abouheaf, M., Gueaieb, W., and Lewis, F. (2018). Model-free gradient-based adaptive learning controller for an unmanned flexible wing aircraft. *Robotics*, 7:66. doi: 10.3390/robotics7040066.

[Baomar and Bentley, 2016a] Baomar, H. and Bentley, P. J. (2016a). An intelligent autopilot system that learns flight emergency procedures by imitating human pilots. In *Proceedings of the IEEE Symposium Series on Computational Intelligence (SSCI), Dec. 9 2016, Athens, Greece*, pages 1–9. doi:10.1109/SSCI.2016.7849881.

[Baomar and Bentley, 2016b] Baomar, H. and Bentley, P. J. (2016b). An intelligent autopilot system that learns piloting skills from human pilots by imitation. In *Proceedings of the International Conference on Unmanned Aircraft Systems (ICUAS), June 10 2016, Arlington, USA*, pages 1023–1031. doi:10.1109/ICUAS.2016.7502578.

[Becker-Ehmck et al., 2020] Becker-Ehmck, P., Karl, M., Peters, J., and Smagt, P. V. D. (2020). Learning to fly via deep model-based reinforcement learning. *ArXiv*, abs/2003.08876.

[Bouabdallah and Siegwart, 2007] Bouabdallah, S. and Siegwart, R. (2007). Full control of a quadrotor. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Nov. 2 2007, San Diego, USA*, pages 153–158. doi:10.1109/IROS.2007.4399042.

[Bulka and Nahon, 2017] Bulka, E. and Nahon, M. (2017). Autonomous control of agile fixed-wing uavs performing aerobatic maneuvers. In *Proceedings of the International Conference on Unmanned Aircraft Systems (ICUAS), June 16 2017, Miami, USA*, pages 104–113. doi:10.1109/ICUAS.2017.7991437.

[Bulka and Nahon, 2019] Bulka, E. and Nahon, M. (2019). Automatic control for aerobatic maneuvering of agile fixed-wing uavs. *Journal of Intelligent & Robotic Systems*, 93. doi:10.1007/s10846-018-0790-z.

[Coates et al., 2008] Coates, A., Abbeel, P., and Ng, A. Y. (2008). Learning for control from multiple demonstrations. ICML '08, page 144–151, New York, NY, USA. Association for Computing Machinery. doi:10.1145/1390156.1390175.

[Flint et al., 2002] Flint, M., Polycarpou, M., and Fernandez-Gaucherand, E. (2002). Cooperative control for multiple autonomous uav's searching for targets. In *Proceedings of the 41st IEEE Conference on Decision and Control (Proc. IEEE Conf. Decis. Control), Dec. 13 2002, Las Vegas, USA*, volume 3, pages 2823–2828. doi:10.1109/CDC.2002.1184272.

[Heaton, 2008] Heaton, J. (2008). *Introduction to Neural Networks for Java, 2nd Edition*. Heaton Research, Inc., 2nd edition. isbn:1604390085.

[Ho and Ermon, 2016] Ho, J. and Ermon, S. (2016). Generative adversarial imitation learning. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc. url:https://proceedings.neurips.cc/paper/2016/file/cc7e2b878868cbae992d1fb743995d8f-Paper.pdf.

[Hussein et al., 2017] Hussein, A., Gaber, M. M., Elyan, E., and Jayne, C. (2017). Imitation learning: A survey of learning methods. *ACM Computing Surveys*, 50(2). doi:10.1145/3054912.

[Hwangbo et al., 2017] Hwangbo, J., Sa, I., Siegwart, R., and Hutter, M. (2017). Control of a quadrotor with reinforcement learning. *IEEE Robotics and Automation Letters*, 2(4):2096–2103. doi:10.1109/lra.2017.2720851.

[Janner et al., 2019] Janner, M., Fu, J., Zhang, M., and Levine, S. (2019). When to trust your model: Model-based policy optimization. In *Proceedings of the Conference and Workshop on Neural Information Processing Systems (NeurIPS), Dec. 02 2019, Vancouver, Canada*. url:https://proceedings.neurips.cc/paper/2019/file/5faf461eff3099671ad63c6f3f094f7f-Paper.pdf.

[Kaiser et al., 2018] Kaiser, E., Kutz, J. N., and Brunton, S. L. (2018). Sparse identification of nonlinear dynamics for model predictive control in the low-data limit. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 474(2219):20180335. doi:10.1098/rspa.2018.0335.

[Kanistras et al., 2013] Kanistras, K., Martins, G., Rutherford, M. J., and Valavanis, K. P. (2013). A survey of unmanned aerial vehicles (uavs) for traffic monitoring. In *Proceedings of the International Conference on Unmanned Aircraft Systems (ICUAS), May 31 2013, Atlanta, USA*, pages 221–234. doi:10.1109/ICUAS.2013.6564694.

[Koh and Wich, 2012] Koh, L. P. and Wich, S. A. (2012). Dawn of drone ecology: Low-cost autonomous aerial vehicles for conservation. *Tropical Conservation Science*, 5(2):121–132. doi:10.1177/194008291200500202.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems, Dec. 6 2012, Lake Tahoe, Nevada*, NIPS'12, page 1097–1105, Red Hook, NY, USA. Curran Associates Inc.

[Kumar et al., 2020] Kumar, N., Chauhan, R., and Dubey, G. (2020). Applicability of financial system using deep learning techniques. In Hu, Y.-C., Tiwari, S., Trivedi, M. C., and Mishra, K. K., editors, *Proceedings of the Ambient Communications and Computer Systems(RACCCS), August 17 2019, Ajmer, India*, pages 135–146, Singapore. Springer Singapore. isbn:978-981-15-1518-7.

[Levin et al., 2017] Levin, J. M., Paranjape, A., and Nahon, M. (2017). Agile fixed-wing uav motion planning with knife-edge maneuvers. In *Proceedings of the International Conference on Unmanned Aircraft Systems (ICUAS), June 16 2017, Miami, USA*, pages 114–123. doi:10.1109/ICUAS.2017.7991475.

[Li et al., 2019] Li, B., Gao, P., Li, X., and Chen, D. (2019). Intelligent attitude control of aircraft based on lstm. volume 646. doi:10.1088/1757-899X/646/1/012013.

[Liang et al., 2018] Liang, X., Zheng, M., and Zhang, F. (2018). A scalable model-based learning algorithm with application to uavs. *IEEE Control Systems Letters*, 2(4):839–844. doi:10.1109/LCSYS.2018.2849576.

[Lin et al., 2019] Lin, J., Wang, L., Gao, F., Shen, S., and Zhang, F. (2019). Flying through a narrow gap using neural network: an end-to-end planning and control approach. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Nov.8 2019, Macau, China*, pages 3526–3533. doi:10.1109/IROS40897.2019.8967944.

[McConley et al., 2000] McConley, M. W., Piedmonte, M. D., Appleby, B. D., Frazzoli, E., Feron, E., and Dahleh, M. A. (2000). Hybrid control for aggressive maneuvering of autonomous aerial vehicles. In *Proceedings of the 19th Digital Avionics Systems Conference (19th DASC), Oct. 13 2000, Philadelphia, USA*, volume 1, pages 1E4/1–1E4/8 vol.1. doi:10.1109/DASC.2000.886897.

[Pascanu et al., 2013] Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In Dasgupta, S. and McAllester, D., editors, *Proceedings of the 30th International Conference on International Conference on Machine Learning(ICML'13), June 16 2013, Atlanta, USA*, volume 28 of *Proceedings of Machine Learning Research*, pages 1310–1318, Atlanta, Georgia, USA. PMLR. url:http://proceedings.mlr.press/v28/pascanu13.html.

[Rodríguez-Hernandez et al., 2019] Rodríguez-Hernandez, E., Vasquez-Gomez, J. I., and Herrera-Lozada, J. C. (2019). Flying through gates using a behavioral cloning approach. In *Proceedings of the International Conference on Unmanned Aircraft Systems (ICUAS), June 14 2019, Atlanta, USA*, pages 1353–1358. doi:10.1109/ICUAS.2019.8798172.

[Shukla et al., 2020] Shukla, D., Keshmiri, S., and Beckage, N. (2020). Imitation learning for neural network autopilot in fixed-wing unmanned aerial systems. In *Proceedings of the International Conference on Unmanned Aircraft Systems (ICUAS), Sept. 4 2020, Athens, Greece*, pages 1508–1517. doi:10.1109/ICUAS48674.2020.9213850.

[Silva, 2011] Silva, D. C. (2011). *Cooperative Multi-Robot Missions: Development of a Platform and a Specification Language*. PhD thesis, Faculty of Engineering, University of Porto, Porto, Portugal.

[Silva et al., 2009] Silva, D. C., Silva, R., Reis, L. P., and Oliveira, E. (2009). Agent-Based Aircraft Control Strategies in a Simulated Environment. In Dignum, F., Bradshaw, J. M., Silverman, B. G., and van Doesburg, W. A., editors, *Proceedings of the 1st International Workshop on Agents for Games and Simulations (AGS 2009), May 11 2009, Budapest, Hungary*, volume 5920 of *Lecture Notes in Computer Science*, pages 190–205. Springer Berlin / Heidelberg.

[Yoo et al., 2021] Yoo, J., Jang, D., Kim, H. J., and Johansson, K. H. (2021). Hybrid reinforcement learning control for a micro quadrotor flight. *IEEE Control Systems Letters*, 5(2):505–510. doi:10.1109/LCSYS.2020.3001663.

[Zulu and John, 2014] Zulu, A. and John, S. (2014). A review of control algorithms for autonomous quadrotors. *Open Journal of Applied Sciences*, 04:547–556. doi:10.4236/ojapps.2014.414053.