Trabajo Fin de Grado
Ingeniería Aeroespacial

Sistema de aterrizaje automatizado de aeronaves basado en técnicas de aprendizaje reforzado

Automated aircraft landing system based on reinforcement learning techniques

Autor: Lorena Calvente Roldán

Tutor: Carlos Vivas Venegas

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Trabajo Fin de Grado
Ingeniería Aeroespacial

Sistema de aterrizaje automatizado de aeronaves basado en técnicas de aprendizaje reforzado
Automated aircraft landing system based on reinforcement learning techniques

Autor:

Lorena Calvente Roldán

Tutor:

Carlos Vivas Venegas

Profesor Contratado Doctor

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Trabajo Fin de Grado:	Sistema de aterrizaje automatizado de aeronaves basado en técnicas de aprendizaje reforzado Automated aircraft landing system based on reinforcement learning techniques

Autor:	Lorena Calvente Roldán
Tutor:	Carlos Vivas Venegas

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

Estando en el final de esta etapa educativa y echando la vista atrás, estos 4 años han supuesto no solo un aprendizaje académico, sino un descubrimiento de mi persona. He sido capaz de enfrentarme a retos, de luchar por lo que me he propuesto para mejorar y superarme a mí misma. Ha habido momentos difíciles, tanto en lo personal como en los estudios, porque sí, este grado ha resultado ser todo lo complejo que decían o incluso más al vivirlo de primera mano. Aún así, todo ello ha llevado a que ahora esté más orgullosa de lo que he conseguido. No se me olvidan todas estas experiencias por las que he pasado junto con profesores y compañeros, con los que he compartido risas y preocupaciones. Mi familia ha sido un gran apoyo, en especial mis padres y mi hermano, a los que les debo gran parte de lo que soy.

Asimismo, agradezco toda la ayuda prestada a Carlos Vivas, tutor de este proyecto, cuya guía a lo largo del mismo ha permitido llevarlo a cabo.

Resumen

El objetivo de este proyecto es estudiar las posibilidades de aplicación de técnicas de aprendizaje supervisado (apprenticeship learning) para diseñar un piloto automático capaz de aterrizar un avión en un aeropuerto específico desde una aproximación final a la altitud del patrón de tráfico establecido, confiando solo en instrumentos instalados en la aeronave.

Los procesos de decisión de Markov (Markov Decision Process (MDP)) proporcionan un marco útil para optimizar el comportamiento de sistemas utilizando aprendizaje reforzado. La exploración de posibles políticas de control en sistemas que tienen espacios de estado de gran dimensión puede ser computacionalmente desafiante, especialmente si la dinámica del sistema es desconocida o difícil de modelar, como en el caso de estudio planteado. La técnica conocida como apprenticeship learning es una alternativa en la que un experto humano guía la exploración de políticas de control mediante la ejecución manual de la tarea deseada. Se ha demostrado que esto resulta en rendimientos casi óptimos en relación al rendimiento del ser humano, y es computacionalmente eficiente.

El simulador de vuelo X-Plane se utilizará para probar y demostrar el comportamiento del piloto automático diseñado. X-Plane es un simulador de vuelo comercial desarrollado por Laminar Research que utiliza la teoría de los elementos de pala para modelar en tiempo real las fuerzas aerodinámicas que intervienen en las distintas partes de un avión, lo que resulta en un comportamiento realista incluso en situaciones complejas.

Como se describe a lo largo del documento, el trabajo propuesto es ambicioso e implica la integración de ideas y conceptos de disciplinas diversas, así como el desarrollo de software específico para su implementación.

Desafortunadamente, a fecha de finalización de este trabajo, las diferentes etapas del proyecto han sido cubiertas con una eficacia desigual, no siendo posible obtener una aplicación cerrada en estado final de uso. La memoria describe en detalle el trabajo realizado y su grado de desarrollo.

Abstract

The objective of this project is to study the possibilities of application of supervised learning techniques (apprenticeship learning) in order to design an automatic pilot which is capable of landing an aircraft at a specific airport from a final approach at the altitude of the established traffic pattern, relying only on instruments installed on the aircraft.

Markov Decision Processes (MDP), provide a useful framework to optimize the behaviour of systems using Reinforcement learning. The exploration of possible control policies in systems that have large-scale state spaces may be computationally challenging, especially if the dynamics of the system is unknown or difficult to model, as in the case of the proposed study. The technique known as apprenticeship learning is an alternative in which a human expert guides the exploration of control policies through the manual execution of the desired task. It has been shown that this results in almost optimal performance in relation to human performance, and is computationally efficient.

The X-Plane flight simulator will be used to test and demonstrate the behaviour of the designed autopilot. X-Plane is a commercial flight simulator developed by Laminar Research that uses the theory of blade elements to model in real time the aerodynamic forces that intervene in different parts of an airplane, resulting in realistic behaviour even in complex situations.

As it is described in the document, the proposed work is ambitious and implies the integration of ideas and concepts of diverse disciplines, as well as the development of specific software for its implementation.

Unfortunately, at the deadline of the work, the different stages of the project have been covered with an unequal effectiveness, and it is not possible to obtain a closed application in the final state of use. This document describes in detail the work done and its degree of development.

Contents

Agradecimientos I
Resumen III
Abstract V

1 Introduction **1**
1.1 Overview 1
1.2 Landing 1
1.3 Current approach and landing systems 3
1.4 Why Machine Learning 6
1.5 Procedure 7

2 Previous investigations **9**

3 Tools **11**
3.1 Simulator: X-Plane® 11
3.1.1 Why X-Plane 10 and not X-Plane 11 11
3.2 Programming platform: MATLAB® 11
3.3 Connection between X-Plane and MATLAB 12
3.4 Machine Learning in MATLAB 12

4 First steps with X-Plane **13**
4.1 Getting to know X-Plane 13
4.1.1 Airport: KSEA 16
4.1.2 Aircraft: A380 16
4.1.3 Settings of a flight 16
4.2 Controls and displays 26
4.2.1 Using joysticks or similar 30
4.2.2 Using keyboard and mouse 30
4.3 Setting a landing manually 30

5 X-Plane Connect **33**
5.1 Installing plugin in X-Plane 33
5.2 Inner functions of X-Plane Connect 33
5.3 Data 34
5.3.1 Notes 36
5.3.2 Equipment of the aircraft 36
5.4 Own code in X-Plane Connect 37
5.4.1 Start X-Plane 37
5.4.2 Import X-Plane Connect 37
5.4.3 Setup 37
5.4.4 Set the initial position 38

5.4.5 Set initial conditions: autothrottle, localizer, approach 38
5.4.6 Parameter extraction 38
5.5 Considered cases 39

6 Machine Learning (Inverse Reinforcement Learning) 41
6.1 Markov Decision Processes 42
6.2 IRL algorithms 42
6.2.1 IRL from sampled trajectories 43

7 IRL Toolkit 45
7.1 Installation 45
7.2 Directory overview 46
7.2.1 IRL algorithms 46
7.2.2 MDP solvers 49
7.2.3 Example domains 49
Functions that define the domain 50
7.2.4 Miscellaneous 50
7.3 Usage 51
7.4 Definition of the current problem 51
7.4.1 Airport data 52
7.4.2 Conversions 53
From degrees into coordinates 53
Ratios to coordinates 54
7.4.3 landing 55
7.5 XPC IRL converter 56
7.5.1 Options 57
File configuration 57
Airport information 57
IRL algorithm information 58
7.5.2 Processing 59
7.5.3 First structure, *mdp_params* 59
7.5.4 Second structure, *example_sample* 59
7.6 Own codes in IRL Toolkit 60
7.6.1 landingcoordstostate 60
7.6.2 landingstatetocoords 62
7.6.3 coordstoratios 63
7.6.4 landingdefaultparams 63
7.6.5 Unfinished functions 63
7.7 Continuation 64

8 Considerations and future perspective 67

9 Own codes 69
9.1 XPC Toolbox 69
9.1.1 SERP3nmiapp.m 69
9.2 landing.m 75
9.2.1 airportdata.m 80
9.3 XPC_IRL_converter.m 81
9.3.1 ratiostocoords.m 86
9.3.2 degreestocoords.m 88
9.3.3 landingcoordstostate.m 91
9.4 IRL Toolkit 92
9.4.1 landingcoordstostate.m 92
9.4.2 landingstatetocoords 92
9.4.3 coordstoratios.m 93

9.4.4 landingdefaultparams.m 94
Attachments 97

List of Figures 99
List of Tables 101
List of Codes 103
Bibliography 105
Glossary 109

1 Introduction

The current situation of airports around the world is very different. Depending on the area they are located, it is found some aids systems (of diverse categories) or not.

It is true that nowadays in such a modern society, the phases of a flight have been automatized with flight directors, autopilots, however not all of them can be performed by software. In a future, planes will have only computers and a pilot to supervise that everything the machine commands is correct instead of two pilots and a computer that helps them.

Landing is a difficult operation that always ends in humans taking control of the situation. Although landing systems aids could be implemented in all the airports to make it easy, the cost would be extremely high.

This work aims to provide a computational solution to the problem from a Machine Learning approach.

1.1 Overview

The intention is to develop a software capable of landing an aircraft without human intervention or external devices to the plane. The question that arises is where to start.

In the past few decades machine learning as revealed itself as a invaluable tool to deal with complex computational problems requiring to deal with large unstructured amounts of data. This consists on making a machine learn from training data, mimicking the capabilities of a human brain, by means of apropriate algoritmic techniques (described in Section 1.4). The resulting algorithm allows the computer to take decisions autonomously when presented with similar situations to those used in the training phase.

The result is a more economically feasible solution than implementing the complex and costly landing aid technology required at every airport. Moreover, there exist airports where this technology can not be deployed due to physical constraints (buildings or rugged terrain nearby for instance), that could benefit from the proposed approach since no external equipment to the aircraft are required.

In the first place, a more accurate definition of landing is provided. The next step is deciding the procedure. Due to considerations of practical nature and facing the impossibility of accessing real data of aircraft flights, a flight simulator (X-Plane) has been employed to obtain the data for the project. Due to the inexperience of the author in flying an aircraft, it has been decided to make the simulator govern the plane, with the necessary landing aids to get a smooth touchdown. The final aim is to develop algorithms which, taking as reference the simulator landing trajectories, are capable of generating aircraft piloting actions autonomously in scenarios not presented to the system in the training phase. However during this phase, unexpected problems have delayed the resolution of the problem, causing the software to not be able to perform the landing at the moment of the presentation of this document. Hence, it can't produce graphs of airport approach trajectories or similar results, although its obtaining is described.

1.2 Landing

Landing is a mandatory phase of a flight. Generally, an airplane is descending and decelerating until it reaches the final approach speed, which allows to deploy flaps to slow down the speed and increase the resistance. Near the ground the aircraft must increase the angle of attack to get a gentle landing, performing a flare, reducing the rate of descent. Upon touchdown, spoilers (sometimes called "lift dumpers") are deployed to dramatically reduce the lift and transfer the aircraft's weight to its wheels, first to the main gear and then to the auxiliary gear as well, until it comes to complete stop. Some extra resources can be speed brakes or reverse thrust.

In [1], 5 phases are distinguished:

- base leg: the pilot decides the altitude and distance to start the landing to reach to the desired spot. It depends on wind, flaps, altitude....
- final approach: with the longitudinal axis of the airplane aligned with the centreline of the runway, the final flap setting is completed and the pitch attitude adjusted as required for the desired rate of descent. It is affected by the four fundamental forces that act on an airplane (lift, drag, thrust, and weight), as well as wind. It is controlled throughout the approach with slight adjustments in airspeed, attitude, power and drag (flaps or forward slip); so that the airplane lands in the center of the first third of the runway. Flaps allow to produce what can be appreciated in Figure 1.1:
  - greater lift, permitting lower landing speed
  - greater drag, permitting a steeper descent angle without airspeed increase
  - reducing the length of the landing roll
- round out (flare): a slow, smooth transition from a normal approach attitude (nose low) to a landing attitude (nose up), gradually rounding out the flightpath to one that is parallel with, and within a very few inches above, the runway. It starts 10 or 20 feet above the ground, as a continuous process of changing angle of attack until the touchdown in Figure 1.2.
- touchdown: smooth setting of the airplane onto the landing surface. As the previous phase, it is done at minimum controllable airspeed so that the airplane touches down on the main gear at approximately stalling speed. At this point, the proper landing attitude is attained, decreasing the angle of attack, to touch the ground with the auxiliary gear.
- after-landing roll: the landing phase is completed when the aircraft decelerates to the normal taxi speed or has been brought to a complete stop when clears of the landing runway.

This project is focused on the final approach and the flare phases and to simplify, the after-landing roll is ended with the plane stopped.

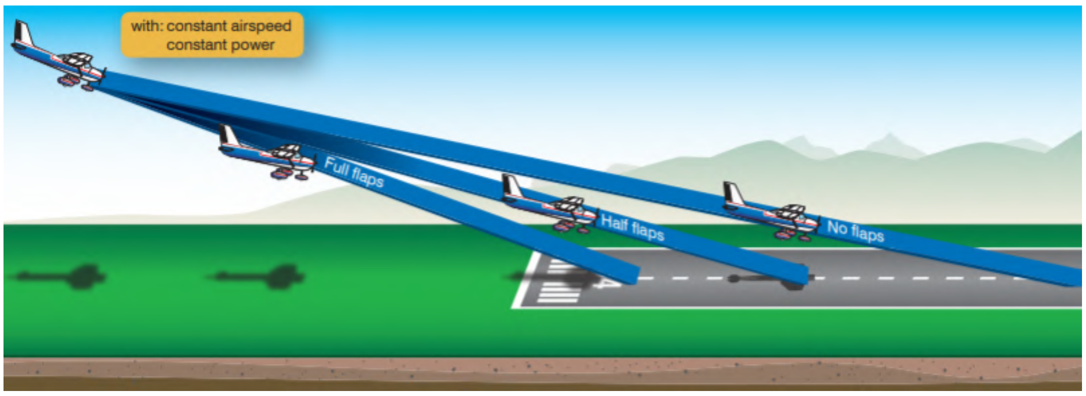


Figure 1.1 Effect of flaps on the landing point [1].

Regarding the final approach speed, it depends on weight, type of aircraft, crosswind, weather, runway altitude, visibility, avionics, ... For instance, in light aircraft, the stall speed may be reached, while in large aircraft, the airspeed and attitude are adjusted to kept them well above stall speed, at a constant rate of descent. As to the landing distance, it is related with aircraft gross weight and airfield elevation.

While in traditional descents aircraft repeatedly level off and power up the engines with traditional staircase descents, due to the fact of being constrained by the availability and proximity of ground-based navigation aids following the Required Navigation Performance (RNP); nowadays, things are changing. More precise Area Navigation (RNAV) technologies based on GPS provide a more efficient and smoother arrival, with less fuel usage and less noise [2].

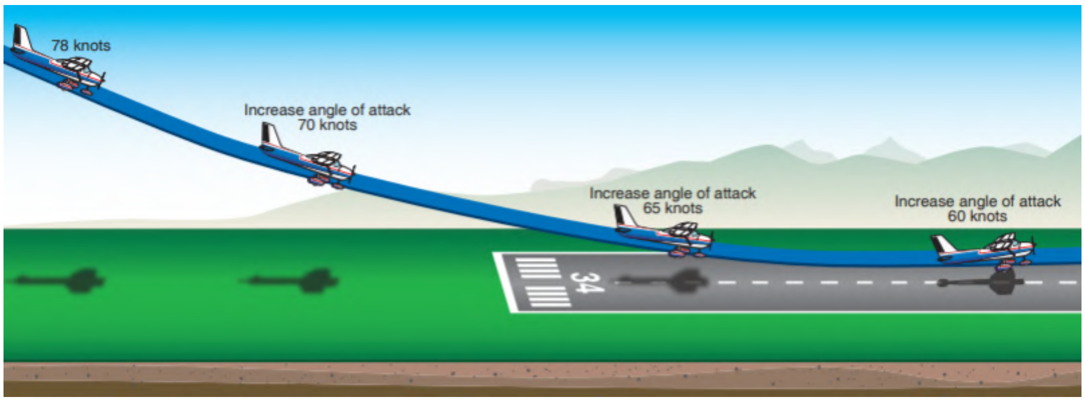


Figure 1.2 Changing angle of attack during flare [1].

1.3 Current approach and landing systems

In the first place, it would be appropriate to know about the systems (landing aids) that exist in some places, and how they are used due to their importance to this project.

In this type of navigation systems, some ground infrastructures are needed to send information to aircraft equipments using radiocommunication technology.

The approach and landing phases are those where the aircraft goes from being in the air at cruise speed, to leaving the runway. It is divided into two segments characterized by the touchdown point or touchdown zone. The take-off and landing operations are those which most compromise the security of the flight under bad weather conditions. In air navigation, these are called Instrument Meteorological Condition (IMC), instead of Visual Meteorogical Conditions (VMC) if the atmospheric conditions allows a landing without the help of external assistance. In order to precise when a case is given, visibility is defined as the horizontal and vertical distance in which it is possible recognize objects [3]. The ceiling is the height between the runway and the clouds. Whenever the visibility is lower than 2600 feet and the ceiling 200 feet, it will be required necessarily electronic assistance.

In the approach phase, the plane has to find the glide slope and follow it until the touchdown point. It is align horizontally with the runway axis and an elevation of 3 degrees. This is achieved transmitting radio signals, from the ground installations, that are received on board indicating to the pilot if he is navigating correctly or the horizontal or vertical course have to be corrected. Not all the navigation systems can provide the accuracy and precision required in the aeronautical world, such as:

- Very High Frequency Omnidirectional Range (VOR)
- Distance Measuring Equipment (DME), which can replace the markers of the Instrument landing System, explained later.
- Automatic Directional Finding/Non Directional Beacon (ADF-NDF)

For instance, although VOR can provide the horizontal heading, it would not indicate the height to navigate to. Regarding the other two systems, they do not have enough precision. [4]

Therefore the systems adequate to assist in the operation are:

- Instrument Landing System (ILS). It has been the mainstay of landing navigation aids and provide aircraft with precision vertical and horizontal navigation guidance information during approach and landing. [5]
- Microwave Landing System (MLS). A developed system with better capabilities than ILS and with the purpose of replacing it. However it has not had future due to the economical inversion required and the emergency of the satellite-based navigation systems.
- Global Position System (GPS) is one of those satellite-based navigation systems which consist of three segments: spatial, terrestrial and user, with a group of satellites that orbit around the Earth and emitting radioeletric signals, provide the necessary information to the user system to determine its position. The system has some deficiencies such as jamming (easily corrupted) and a weak signal. If precision improvement techniques are included, it is known as a whole as Global Navigaton Satellite System (GNSS).

- Wide Area Augmentation System (WAAS) is a type of GNSS with North America coverage that provides service for all classes of aircraft in all phases of flight including vertically-guided landing approaches in instrument meteorological conditions at all qualified locations through the United States National Airspace System (U.S. NAS) accordingly to Federal Aviation Administration (FAA).[6]
- Ground Based Augmentation System (GBAS), previously known as Local Area Augmentation System (LAAS), provides differential corrections and integrity monitoring of GNSS in the vicinity of the host airport (23 nautical miles, via a Very High Frequency (VHF) radio data link from a ground-based transmitter, as it is demonstrated in Figure 1.3. It achieves an accuracy less than one meter in both the horizontal and vertical axis. [7]

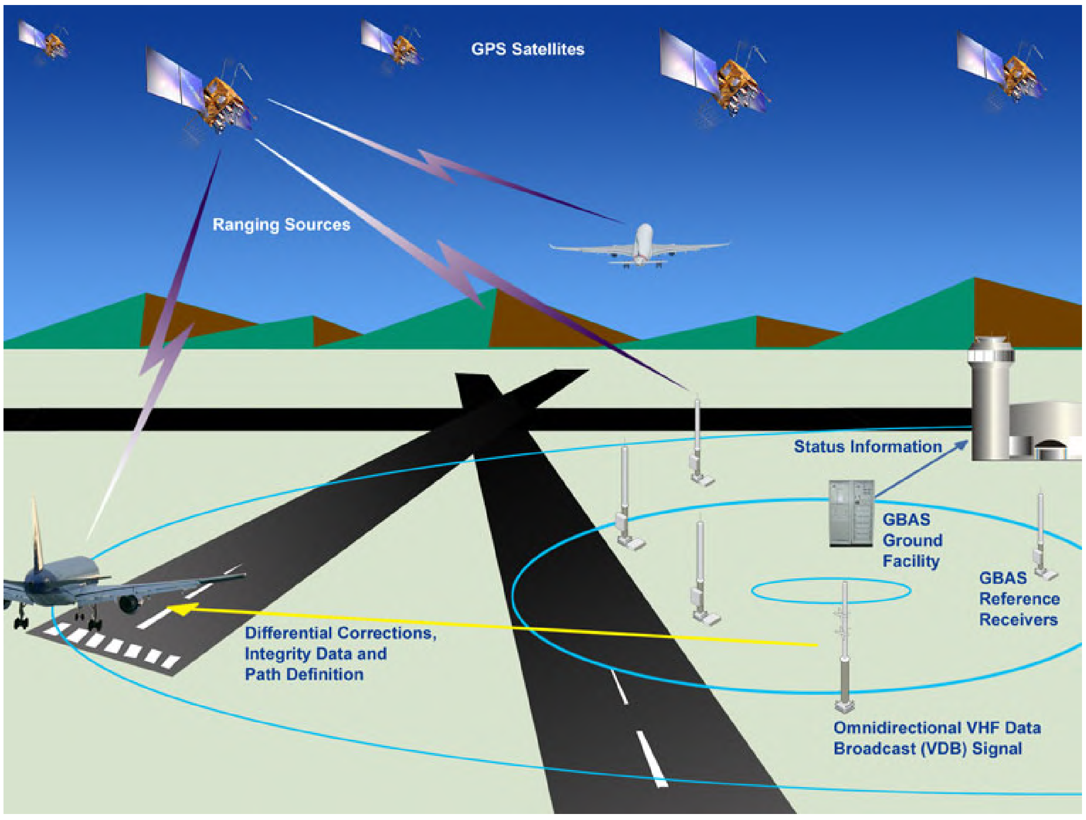

GPS Satellites
Ranging Sources
Status Information
GBAS Ground Facility
GBAS Reference Receivers
Differential Corrections, Integrity Data and Path Definition
Omnidirectional VHF Data Broadcast (VDB) Signal

Figure 1.3 GBAS [7].

The ILS is going to be explained with a more detailed vision due to the influence into this project,which counts on it to extract data to train the machine.

It consists of three different subsystems:

- Localizer (LOC), which indicates how to maintain the plane horizontally in the correct trajectory using an array of aerials, located 300 meters behind the runway.
- Glideslope (G/S), which is focused in the vertical mode with two or three aerials on a support providing two bars in the artificial horizon, emplaced 30 m from the edge of the contact area with the runway.
- Markers, which indicate how far is the runway with directional radiobeacons and the Morse message broadcast that implies that a light switches on as blue, amber or white.

While localizer systems are sensitive to obstructions in the signal broadcast area, such as large buildings or hangars, glide slope systems are also limited by the terrain in front of the glide slope aerials. Since the ILS signals are pointed in one direction by the positioning of the arrays, glide slope supports only straight-line approaches with a constant angle of descent. This kind of system is expensive because of sighting criteria and the complexity that involves. In fact, if the glide slope is inoperative, the ILS reverts to a non-precision localizer approach; or if the localizer is inoperative, an ILS approach is not authorized. [8]

The system allows the pilot to know if he is on the left, on the right or in the right position. If the plane deviates to the right, the vertical bar does it to the left indicating to the pilot that he has to turn to the left. In the same way if it is flying high the horizontal bar is under the horizon indicating to the pilot that he must descent.

A more extensive development of ILS is found in [4].

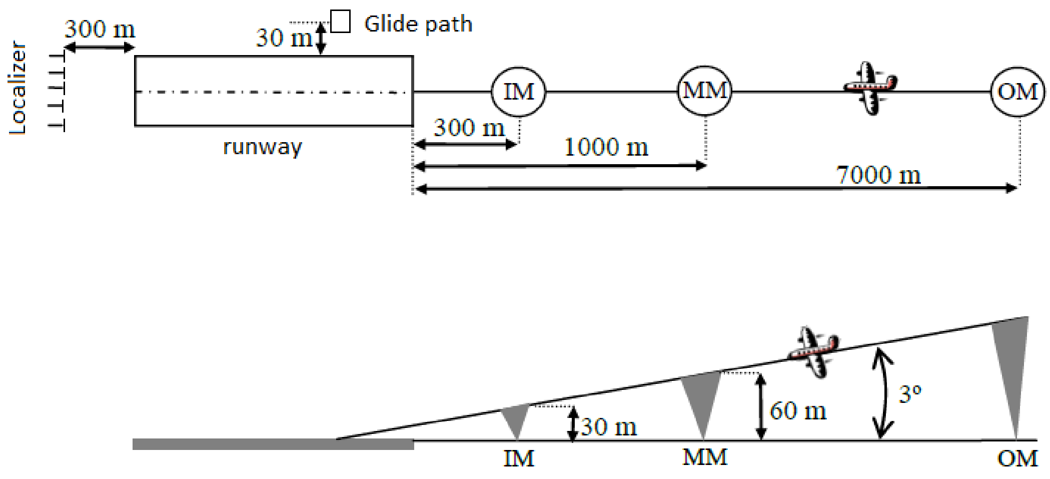


Figure 1.4 Horizontal and vertical view of ILS [4].

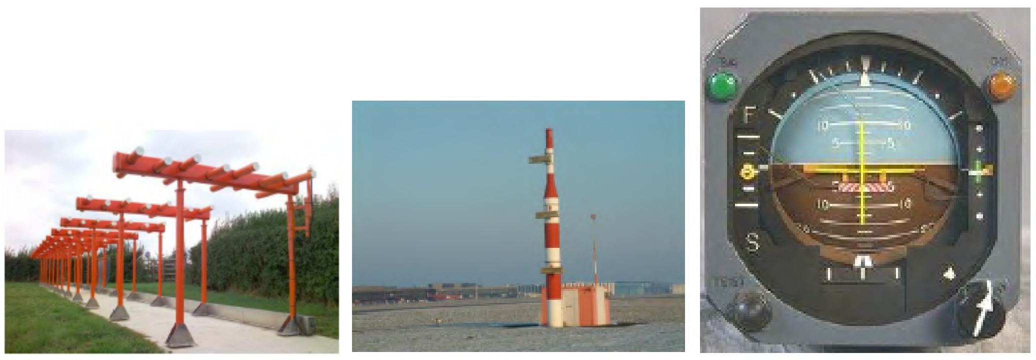


Figure 1.5 Localizer aerial, glide slope installations and ILS indicator [4].

A variation of ILS is the Transponder Landing System (TLS), which does not need to be installed at the end of the runway, and that can be perceived by the aircraft as if it were using ILS, without adding new avionics. [9]

It is important to take into consideration the lowest authorized ILS minimum visibility values, with all required ground and airborne systems components operative as it is demonstrated in Table 1.1. Some definitions may help to understand how they are divided:

- Decision height (DH). "The glide slope transmitter is located between 750 feet and 1,250 feet from the approach end of the runway (down the runway) and offset 250 to 650 feet from the runway centreline. It transmits a glide path beam 1.4 degrees wide (vertically). The signal provides descent information for navigation down to the lowest authorized decision height (DH) specified in the approved ILS approach procedure. The glidepath may not be suitable for navigation below the lowest authorized DH and any reference to glidepath indications below that height must be supplemented by visual reference to the runway environment."[8]
- Runway Visual Range (RVR) measures visibility, background luminance and runway light intensity to determine the distance a pilot should be able to see down the runway. [10]

Regarding the normative to be applied, an aspect to be highlighted is the minimum height to fly above the highest obstacle is 1000 feet within a horizontal radius of 2000 feet of the aircraft over any congested area with people, both in Europe [11] and America [12].

Therefore, the data that are going to be taken will be related to CAT IIIc ILS (the worst conditions), with a runway that offers it and with the intention of landing respecting the minimum altitude.

Table 1.1 ILS Categories [8].

Category	Decision height	RVR
I	200 ft	1800 ft
Special I	150 ft	1400 ft
II	100 ft	1000 ft
III a	None or 100 ft	700 ft
III b	None or 50 ft	150 ft
III c	None	None

1.4 Why Machine Learning

This project focuses on Machine Learning (ML) as a way of responding the objective of producing a non-expensive aircraft landing aid, and a software capable of performing it itself.

Artificial Intelligence (AI), is a technique that allows machines to replicate the human behaviour. It includes Machine Learning, which is a data analysis that automatizes the process of constructing models, employing algorithms. Machines learn from the data, and are capable of facing new situations without human intervention. It is important to distinguish them and to differentiate from Deep Learning (DL), concept on the rise, a particular branch of ML which employs huge quantities of data (not available to this purpose) to learn from, using advances in computational science.

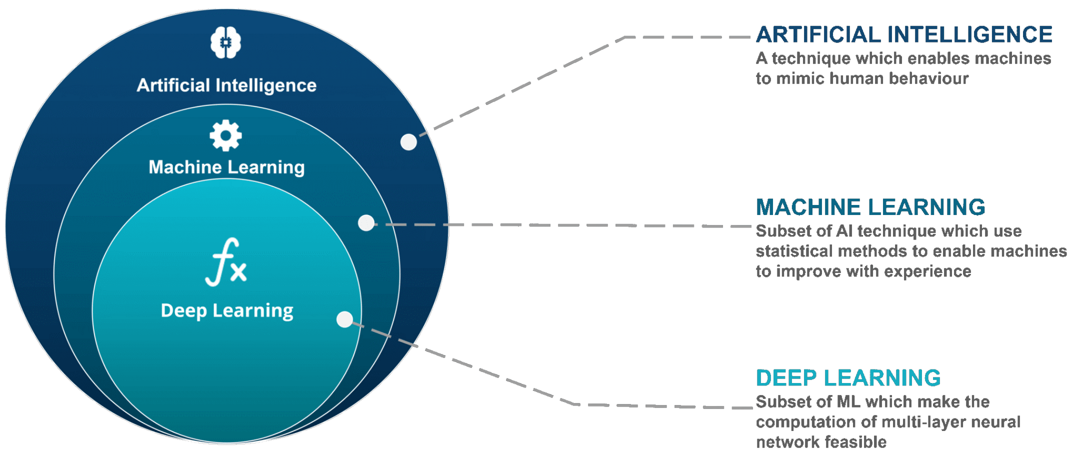

Artificial Intelligence
Machine Learning
fx
Deep Learning
ARTIFICIAL INTELLIGENCE
A technique which enables machines to mimic human behaviour
MACHINE LEARNING
Subset of AI technique which use statistical methods to enable machines to improve with experience
DEEP LEARNING
Subset of ML which make the computation of multi-layer neural network feasible

Figure 1.6 AI vs ML vs DL [13].

From its origin in the search of artificial intelligence which result in the creation of neural nets, the application of statistical methods in simple algorithms to the automatic apprenticeship, an the Bayesian methods; this discipline evolved into approximations based in data instead of knowledge. Lately, unsupervised methods, deep learning, cyber-physic systems and in-situ machine learning have being developed [14].

The main classification is:

- Supervised learning with the aim of finding a function which, given the input data, assigns it the adequate labelled output.
- Unsupervised learning if the labels are unclear and the system has to be capable of recognize patterns to label new inputs.[15]

In order to resolve the different subsets, several algorithms can be implemented. Some decision tables exist like the one of [16].

Reinforcement learning (RL) is one of the latest investigated. It can not be positioned in supervised learning nor unsupervised learning. It is based on improving the answer of the model using a feedback procedure. It encourages the apprenticeship from humans observing the environment and evaluating the response to various actions. In that way the system learns by trial-error. It is not supervised learning because it does not needs a group of labelled data. Likewise, it is not an unsupervised learning due to known expected reward when modelling the apprentice.

Specifically, if what is available as entry to the problem is some trajectories made by an "expert", the problem gets catalogued as an Inverse Reinforcement Learning case.

To finish this previous explanation of Machine Learning, it is remarkable to take care of the data management even in this project where there are various parameters to control in time. Nowadays, the quantity of data that is moving around the world is extremely immense. For instance in a cross-country flight, almost 2.5 thousands of terabytes are generated which leads to being able of extracting the correct information from the appropriate parameters through the tool of Big data [17] (for much larger numbers of data, exceeding the present situation).

1.5 Procedure

For the purpose of clarifying the upcoming contents of the document and providing a general overview, a list of intended steps to follow is presented in the following, (graphically described in Figure 1.7).

- Investigating about related projects in Chapter 2
- Selecting the tools to use in Chapter 3.
- Learning how to use the simulator in Chapter 4.
- Managing the tool to extract data from it and creating an algorithm with the conditions that define the situations to study and the data to extract that interest to the project in Chapter 5.
- Investigating types of Machine Learning, where Inverse Reinforcement Learning Inverse Reinforcement Learning (IRL) is going to be seen as the best option in Chapter 6.
- Processing data through a program to convert it into the kind that is treated by the framework where the algorithm IRL is executed and developing an own environment for the project in Chapter 7. Nonetheless, it has resulted poorly documented and with a number of bugs and errors complicating and requiring a considerable amount of time to be dealt with. [Unfinished]
- Processing acquired data in real time and obtaining the actions to perform in the simulator from the machine, briefly described. [Unfinished]

Unfortunately, it has to be said that due to lack of time and the subestimated complexity of the project (above all of the codes in the toolkit used to develop the IRL algorithm, which have originated unexpected problems), not all these points have been achieved. That is, solving the processing of data to be managed by the IRL framework has been preferred in order to maintain a sequential and logical structure without gaps in the reasoning. However, the steps to follow are described in detail as well as most of the tools needed.

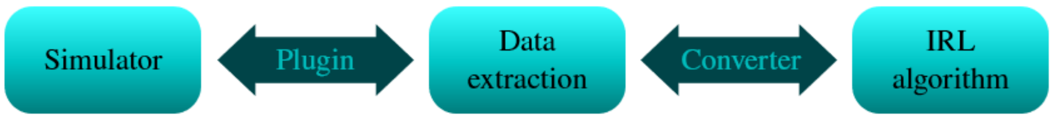


Figure 1.7 Procedure in blocks.

2 Previous investigations

The present project is inspired in the ideas of the summary document of the work of Oren Hazi [18] as CS 229 (Stanford course in Machine Learning (ML)) Final Project in 2011.

In the aforementioned work, after writing a plug-in to export flight parameters from within an aircraft simulator, X-Plane, the author flew approximately 100 training approaches and landings. Once the dynamics were estimated, two planning algorithms were used to guide the airplane in order to reduce costs in two different phases: approach and flare, transitioning when the aircraft reaches 10 feet Above Ground Level (AGL). It has to be said that the costs increase linearly with varying parameter for non ideal states, as well as, the directional movement was not taken into account due to lack of yaw axis control in the joystick.

One of the aspects which makes the present work an improved approach compared to Hazi's is the fact that the data recording has been automatized and generalized. This implies a faster process with a better organization of the extracted information. Moreover, this work's approach does not depend on the ability of a human pilot, because it has been employed the autopilot that controls the throttle, and with the help of the ILS, follows the generated optimal path. To acquire some sense of reality, the expert providing the data to learn from, must be that, an expert, which is not possible in the present case, so the solution is taking the suggested controls of the programmed autopilot in X-Plane. The procedures developed are nonetheless generic in the sense that, provided an adequate data set from actual expert landing manoeuvres, the algorithms should readily provide effective and improved control policies.

It is that kind of efficiency that would be required in every airport independently of humans and the conditions to which the zone is subjected.

During the research for this document, other similar cases of Machine Learning in aviation have been found. Some of the most closely related to the work at hand are briefly described in the following:

- Gizzi and Zabner report [19], which focuses on using Reinforcement Learning (RL) to control the aircraft with high level commands and to increase flight safety, without access to expert instructors to apply apprenticeship learning. It employs X-Plane (the same as in this project) as simulation environment and later, VISTAS (a rapid proto-typing simulation environment of National Aeronautics and Space Administration (NASA)).
- Baomar and Bentley have several interesting papers [20], [21], [22], [23] on the topic in which they apply Artificial Neural Networks (ANN) to manage the data, much larger than what it is going to be treated here.
- Morales and Sammut proceeding of the 21st International Conference on Machine Learning, celebrated every year with updates off experts in this field, combines RL with Behavioural Cloning [24].

Regarding a student approach to this theme, several courses have been consulted, both their lectures and videos, such as:

- CS 229, Stanford course about Machine Learning [25], with special mention to the lectures:
  - Supervised Learning
  - MDPs, Bellman Equations. Value iteration and policy iteration, where Reinforcement Learning is explained.
- CS 188, Berkeley course about Introduction to Artificial Intelligence, remarking the lectures:

  - Markov Decision Processes I and II
  - Reinforcement Learning I and II

A more technical point of view has been acquired through this papers, in a chronological order:

- Algorithms for Inverse Reinforcement Learning, one of the first articles about this modern method of solving problems [26]
- Apprenticeship Learning via Inverse Reinforcement Learning [27]
- Non-linear Inverse Reinforcement Learning with Gaussian Processes [28], where non-linear relations can be established between features to define the problem. They even developed their own algorithm, Gaussian Process Inverse Reinforcement Learning (GPIRL) which is more complex but represents better the reality of complex non-linear problems, giving more accurate results than other algorithms (Maximum Entropy IRL, FIRL and margin-based methods: MMP, MWAL, MMPBoost and LEARCH). The paper includes several examples to demonstrate all the theory, such as an autonomous car driving along a highway avoiding obstacles, with the attached code.

It has to be noted that Machine Learning is not included in any of the subjects of the academic curriculum belonging to the Aerospace Engineering Degree in the Escuela Técnica Superior de Ingeniería (ETSI) of the Universidad de Sevilla (US).

3 Tools

In this chapter the tools needed along the project are exposed, justifying their election.

3.1 Simulator: X-Plane®

The chosen tool to obtain data from and to test the design algorithm has been X-Plane, one of the most complete flight simulator on the market. Other options could have been Microsoft Flight Simulator® which is more orientated as a game although it has the possibility in previous versions of playing a demo in two airports and several airplanes.

The disadvantage of the free demo of X-Plane, in either of its latest versions, 11 [29] or 10 [30], is that it only allows to fly around Seattle-Tacoma International Airport (ICAO code) (KSEA) for 15 minutes. The positive point is that if the simulation is paused to take some notes or data, the time will not be discounted.

Figure 3.1 X-Plane 11 logo [31].

Figure 3.2 X-Plane 10 logo [32].

3.1.1 Why X-Plane 10 and not X-Plane 11

Despite the fact that the first idea of this project was to be implemented in the latest update of the program of January 2019, it results in some way a worse option than X-Plane 10. This one offers the possibility of having the screen divided in two while the newest forces the full screen and its procedures of turn on, settings and controls are a lot much slower.

All these arguments tipped the balance towards using X-Plane 10.51 from October 2016, with lower definition graphics but a more agile response.

This doesn't prevent the codes from been able to be used in either of the versions.

3.2 Programming platform: MATLAB®

MATLAB is a specifically designed platform for engineers and scientists with its own matrix-based language. ETSI is characterized by providing a quite advanced training to the students, above all to those in Aerospace

Engineering Degree, including in the academic curriculum subjects that apply this tool. In most of them, using it for some exercise, while in others, it is a fundamental component, such as:

- Programming fundamentals, where the first contact is taken.
- Numeric Methods, which emphasizes its optimal use, saving time and computational operations.
- Fluid Mechanics II, combined with the application of the theory to develop codes about laminar and turbulent boundary layers.
- Aerodynamics II, with codes related to wings in subsonic and supersonic flight regimes.
- Computational Mathematics, starting from the basis and with the perspective of developing applications.

Figure 3.3 MATLAB logo [33].

3.3 Connection between X-Plane and MATLAB

After the installation and the first use of X-Plane (MATLAB was already implemented in the computer for the project, specifically, version R2018a), a form of communication between both software was required.

The two first libraries [34] and [35] found in the Community of MathWorks® did not provide a correct communication neither the required capability of taking a closer look at the code of the Plug-in to be able to automatize the data recording.

However, after some investigation, the third library was found, surprisingly, as part of a NASA repository in GitHub [36]. The open source research tool allows to send and receive all kind of commands and data incorporating them in some useful examples that have been taken as a baseline and developed in order to adapt them and acquire what is pursued in the project.

3.4 Machine Learning in MATLAB

In the first place several toolboxes about Reinforcement Learning [37] or Markov Decision Processes (MDP) [38] [39] were found, but they did not take into account the fact of having as entry condition expert data, only the policy, which is too much complicate to obtain from this kind of problem.

This limitation motivates the use of the IRL Toolkit [40] that allows us to incorporate expert decision data into the model.

Now that all the required parts of the project have been briefly described to provide a quick and simple understanding, the Figure 1.7 can be further detailed as presented in Figure 3.4. In the following sections an in-deep step-by-step description will be provided in order to make everyone able to replicate all the procedures taken in a chronological order.

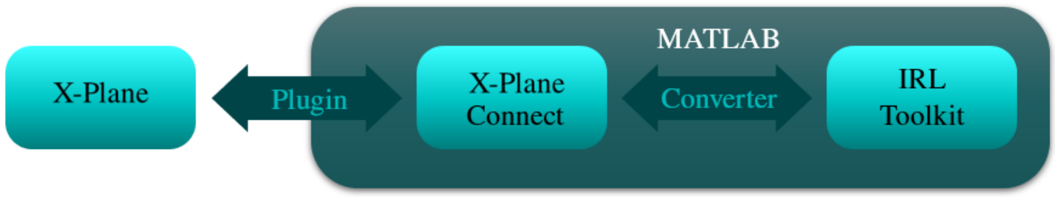


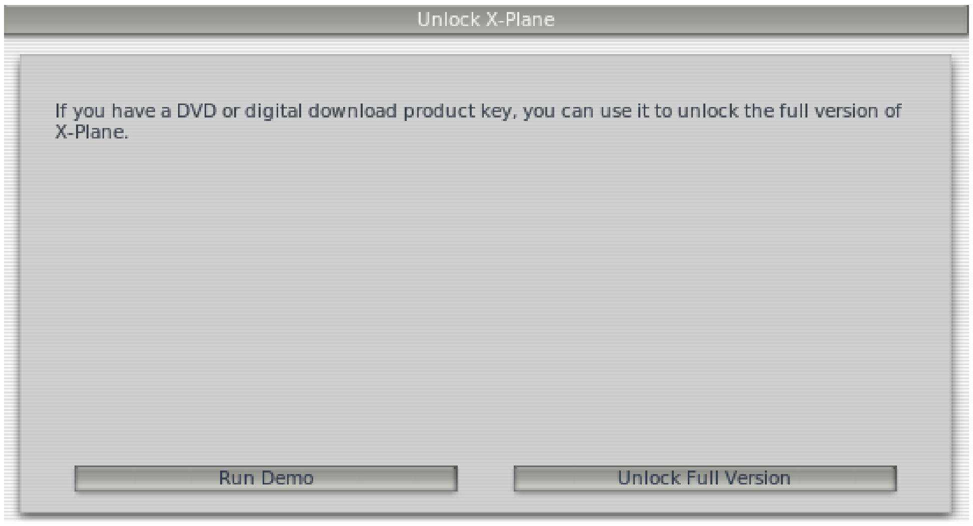
Figure 3.4 Tools in blocks.

4 First steps with X-Plane

The usage of the chosen simulator is going to be treated from a beginner approach. It is focused on this project following all the steps that have been taken by the author of the document for who it is her first simulator.

4.1 Getting to know X-Plane

Some of the main characteristics of X-Plane 10 are going to be detailed. It is launched from X-Plane.exe in X-Plane 10, the folder unzipped of the downloaded one, obtained from [30]. Several seconds are required and a small window opens to select under which option the program is going to be launched. As it has been commented, the version of this project is the free one, and therefore **Run Demo** is chosen in Figure 4.1.



If you have a DVD or digital download product key, you can use it to unlock the full version of X-Plane.

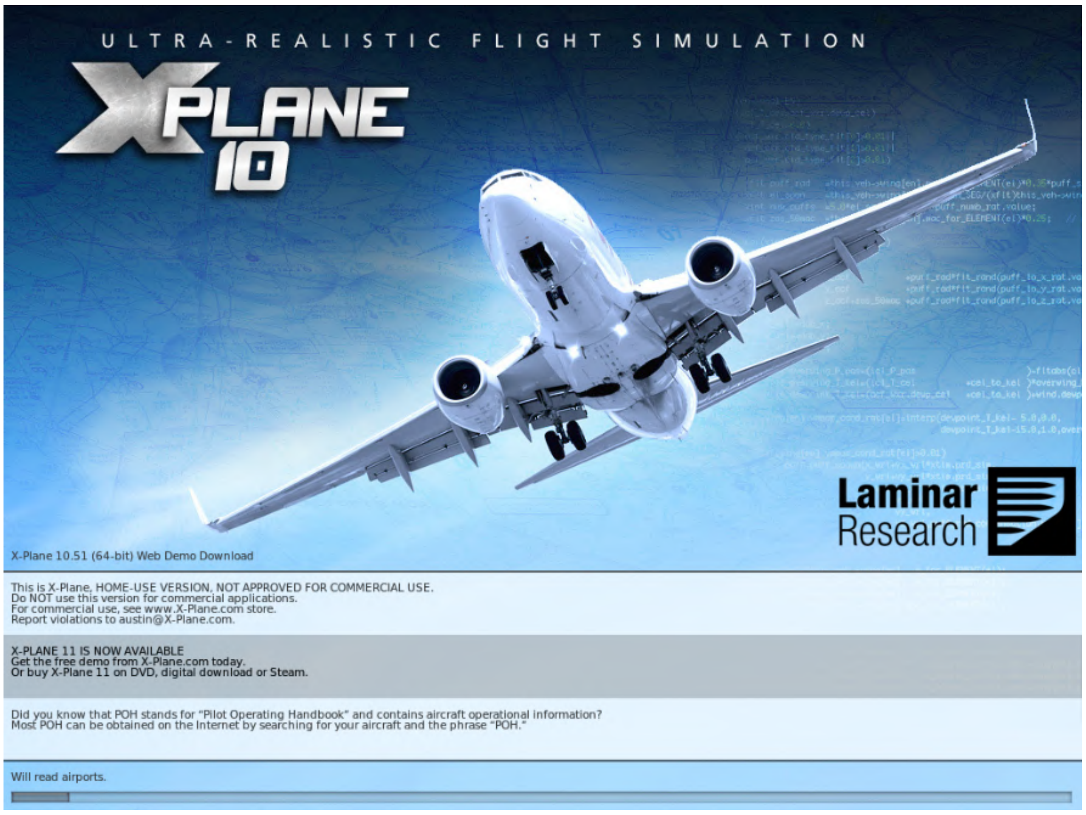
Figure 4.1 First selection.

After some time reading configuration, airports and airplanes from the folder X-Plane 10 as it is shown in Figure 4.2, a new window allows to choose the airport, aircraft, time and weather in Figure 4.3.

This version of X-Plane doesn't allow to charge a specific situation in this main window, so the aircraft appears in one of the runways by default of the selected airport. In the case of study with the demo, only the Seattle Tacoma Intl (KSEA) in Figure 4.4 (with its real version in Figure 4.5) is available.

Time selection:

- day
- sunset
- twilight
- night



X-Plane 10.51 (64-bit) Web Demo Download

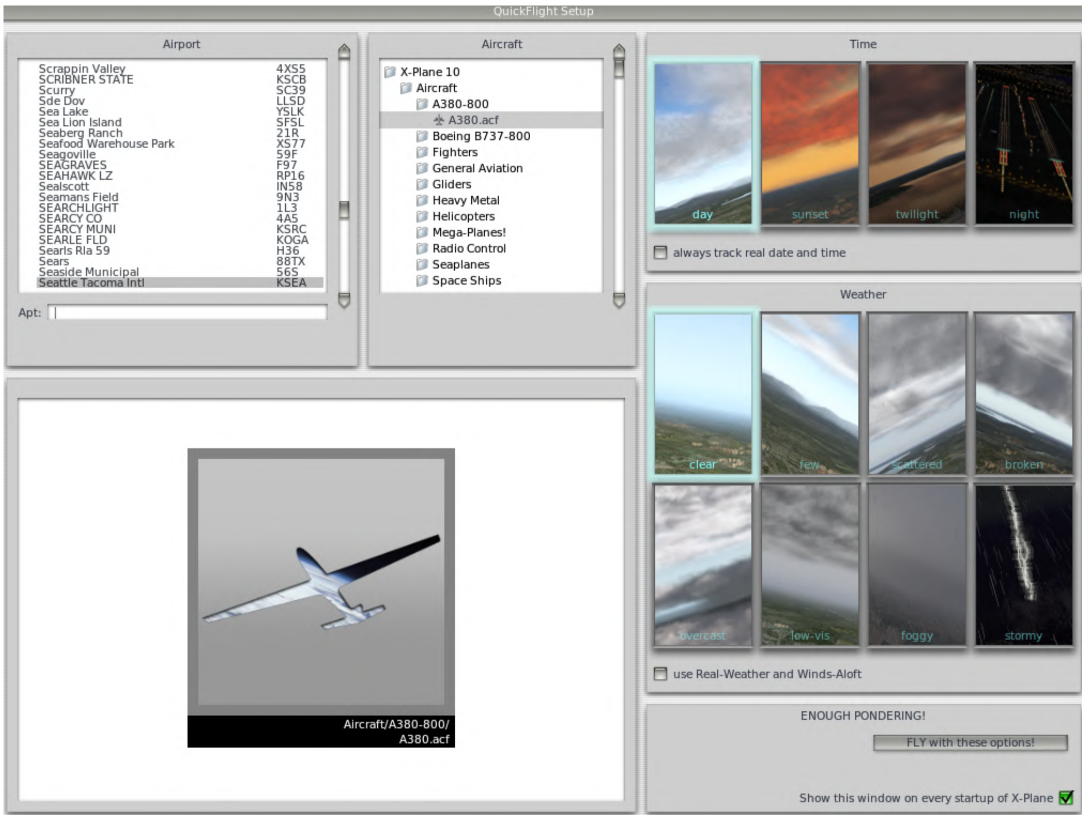
This is X-Plane, HOME-USE VERSION, NOT APPROVED FOR COMMERCIAL USE.
Do NOT use this version for commercial applications.
For commercial use, see www.X-Plane.com store.
Report violations to austin@X-Plane.com.

X-PLANE 11 IS NOW AVAILABLE
Get the free demo from X-Plane.com today.
Or buy X-Plane 11 on DVD, digital download or Steam.

Did you know that POH stands for "Pilot Operating Handbook" and contains aircraft operational information?
Most POH can be obtained on the Internet by searching for your aircraft and the phrase "POH."

Will read airports.

Figure 4.2 Loading X-Plane.



Scrappin Valley	4XS5
SCRIBNER STATE	KSCB
Scurry	SC39
Sde Dov	LLSD
Sea Lake	YSLK
Sea Lion Island	SFSL
Seaberg Ranch	21R
Seafood Warehouse Park	XS77
Seagoville	59F
SEAGRAVES	F97
SEAHAWK LZ	RP16
Sealscott	IN58
Seamans Field	9N3
SEARCHLIGHT	1L3
SEARCY CO	4A5
SEARCY MUNI	KSRC
SEARLE FLD	KOGA
Searls Rla 59	H36
Sears	88TX
Seaside Municipal	56S
Seattle Tacoma Intl	KSEA

Apt:

X-Plane 10
Aircraft
A380-800
A380.acf
Boeing B737-800
Fighters
General Aviation
Gliders
Heavy Metal
Helicopters
Mega-Planes!
Radio Control
Seaplanes
Space Ships

Aircraft/A380-800/
A380.acf

always track real date and time

use Real-Weather and Winds-Aloft

Show this window on every startup of X-Plane ✓

Figure 4.3 QuickFlight Setup.

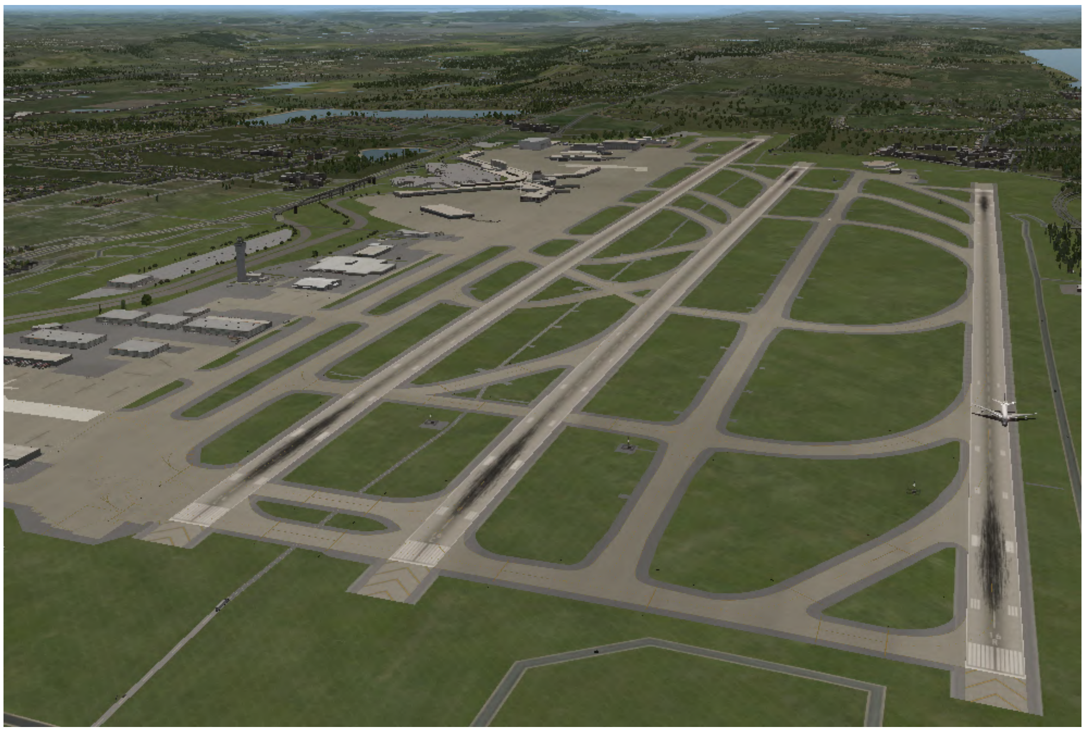


Figure 4.4 Aerial image of Seattle Tacoma airport in simulator.

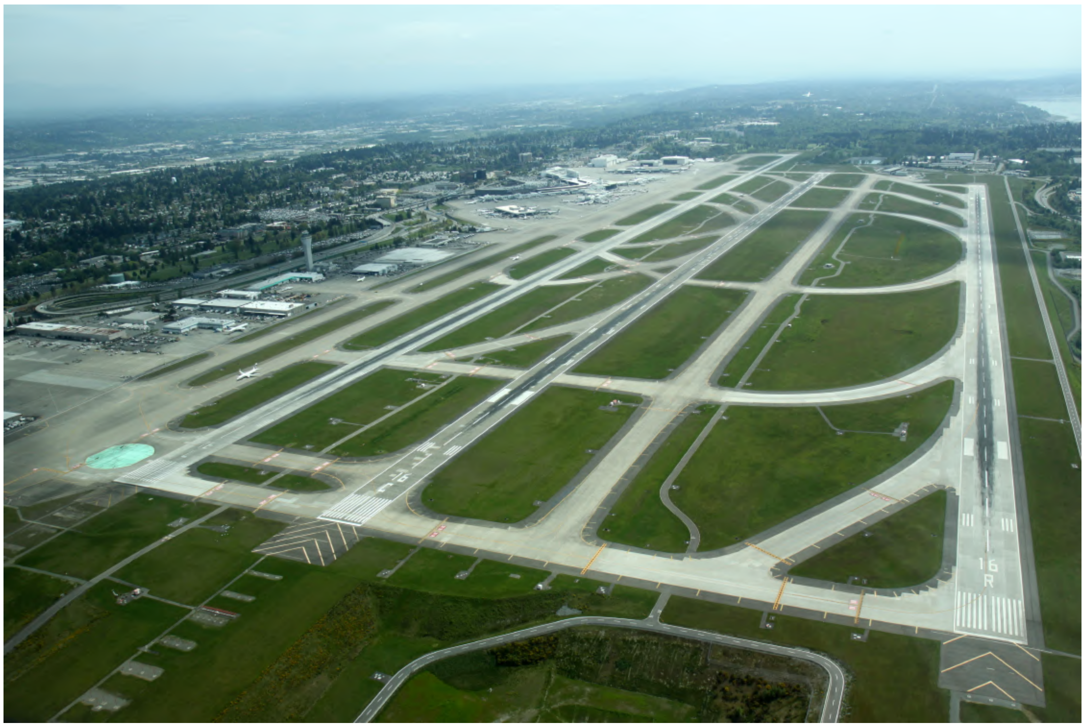


Figure 4.5 Aerial image of Seattle Tacoma airport [41].

Weather selection:

- clear
- few
- scattered

- broken
- overcast
- low-vis
- foggy
- stormy

Therefore, after choosing **Seattle Tacoma Intl KSEA** airport, an airplane, the time and weather and clicking in the box at the right bottom: **Fly with these options**, the system loads all the scenario with its characteristics.

4.1.1 Airport: KSEA

As it has been said, due to the demo version, the only option is KSEA, Seattle Tacoma International airport. It counts with an elevation of 432.3 ft (131.8 m) and three runaways.

The dimensions of the smallest available runaway, the 16R-34L, are 8500 ft x 150 ft. This is the one that most interest implies to the project, because of being the unique of KSEA that presents ILS category III, which has been described in Section 1.3. That permits to study the case with the worst atmospheric and visual conditions. It can be seen on the left side of the Figure 4.6.

According to [42], airplanes can land in both directions using Visual Approaches or Dependent Instrument Approaches. Finally, the chosen one was 16R (heading 161.3°) as the arrival runway (South Flow). It is the less dangerous to the buildings, which are higher near the 34L (heading 341.3°). Besides, the glideslope provided by default is the one of the 16R, as it is at the bottom of Figure 4.15. In order to take data from different situations, landings in 34L have also been performed.

4.1.2 Aircraft: A380

Once the runaway has been chosen to satisfy the established requirements by the project, the aircraft must fulfil the possibilities offered by that runway. Besides, it needs to own autopilot and ILS communication to take the data using this approach aid. In order to apply the results to other situations without problem, the extreme case was taken, the largest commercial plane: A380.

Some of its characteristics related to the landing phase are the final approach speed, the indicated airspeed at threshold in the landing, of 138 knots (71 m/s) and the landing field length between 5200 ft (1600 m, minimum gross weight and 0 ft of airfield elevation) and 8800 ft (2700 m, for maximum gross weight and 8000 ft of airfield elevation) [44]. For KSEA elevation, the landing field length reaches 7200 ft, what assures, enough to land the A380, although it doesn't operate in the airport due to its huge wingspan [45]. However, it could be a potential additional alternate accordingly to airport compatibility of Airbus [46]

More information about this airplane is found in [47], with the classification as regards several organisations such as ICAO and depending on the criterion: wingspan, approach category...

It has to be pointed that the A380 is not one of the defaults in the simulator. Nevertheless, in the official page of the software, several airplanes, among which A380 is, are offered in a free way. It was added copying the downloaded plane data to the folder Aircraft inside X-Plane folder, as it is shown in Figure 4.7. The design of the plane resembles the real one, what can be appreciated in Figure 4.9.

4.1.3 Settings of a flight

The simulator places the airplane on the runway 16 L by default, and the second window about the demo or complete version appears (Figure 4.10). After selecting **Understood**, 15 minutes are available to use the demo unless it is paused pressing letter "P" in the keyboard.

The screen in that moment is as depicted in Figure 4.11 or, if the program fills only half screen, in Figure 4.12. This is the one used during all the project because of developing codes in MATLAB or elaborating this memory simultaneously. It is one of the main reasons why X-Plane 10 has been chosen, with this possibility leading to have other processes at the same time in the screen.

At the top of the cockpit, the menu bar can be displayed if the mouse is moved towards that part of the screen. The first drop-down menu or pop-up menu, **About**, consists on two submenus:

- **About X-Plane**, where some important information about X-Plane can be found in Figure 4.13, such as:

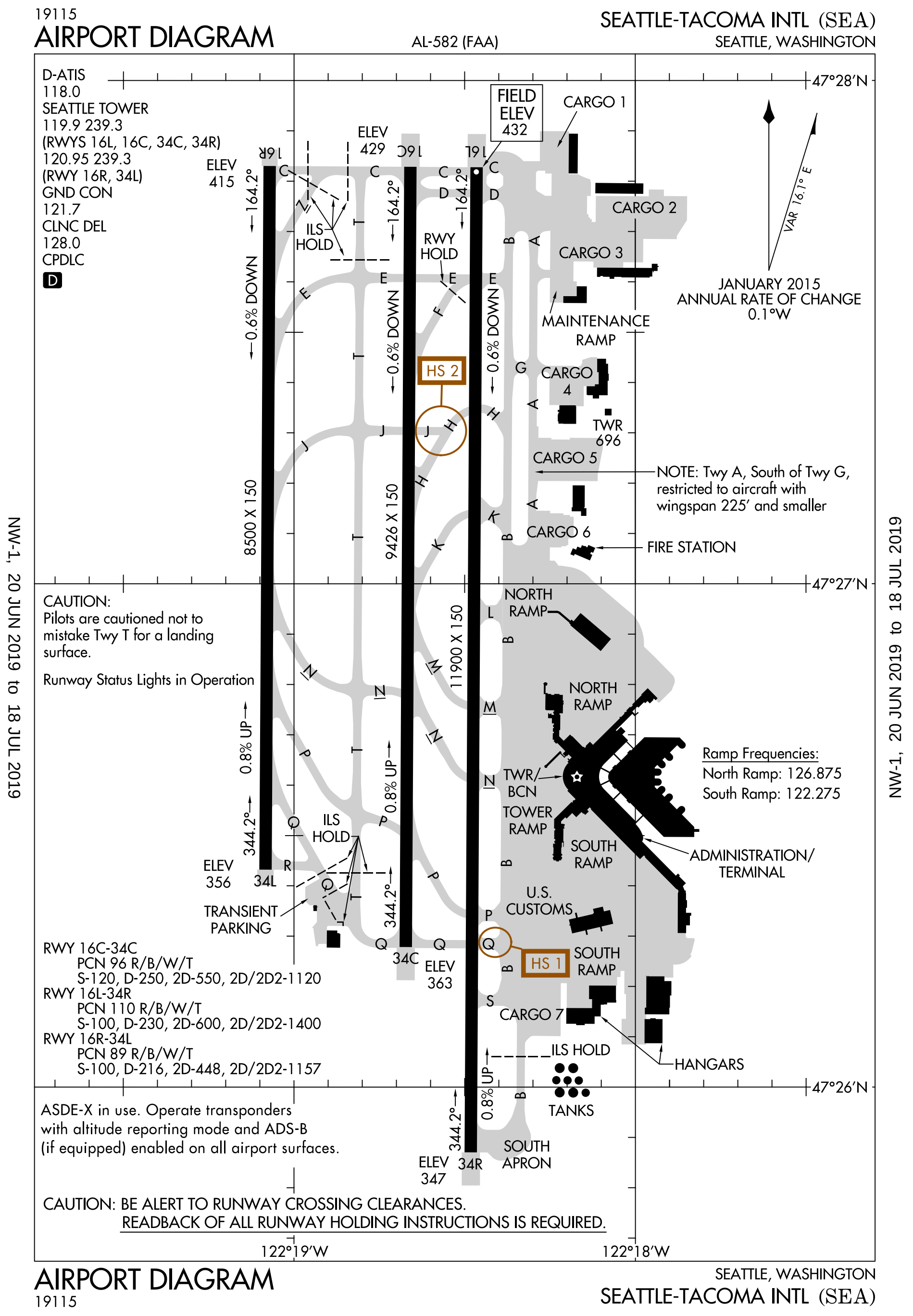


Figure 4.6 Diagram of Seattle Tacoma airport [43].

- version
- author of scenery
- author of algorithms
- author of rendering
- author of airplanes
- author of airports

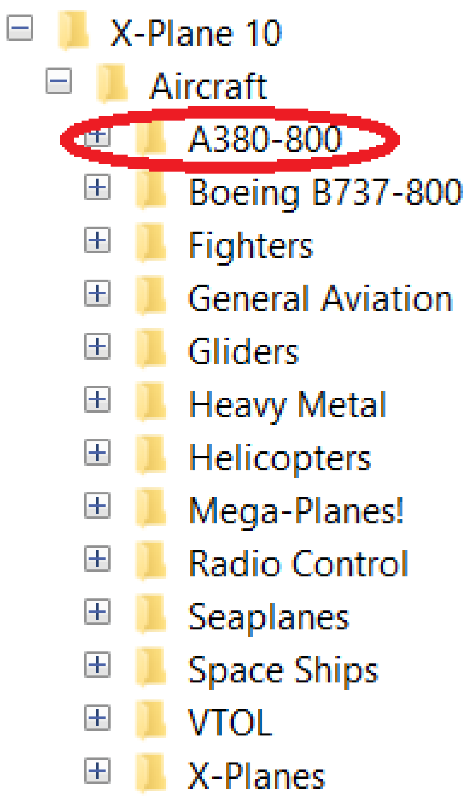


Figure 4.7 Adding A380.

Figure 4.8 Airbus A380 [48].

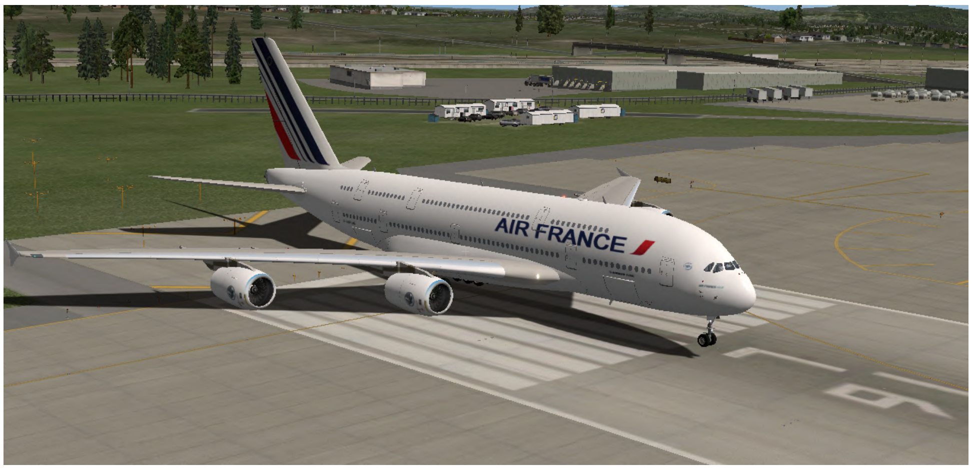
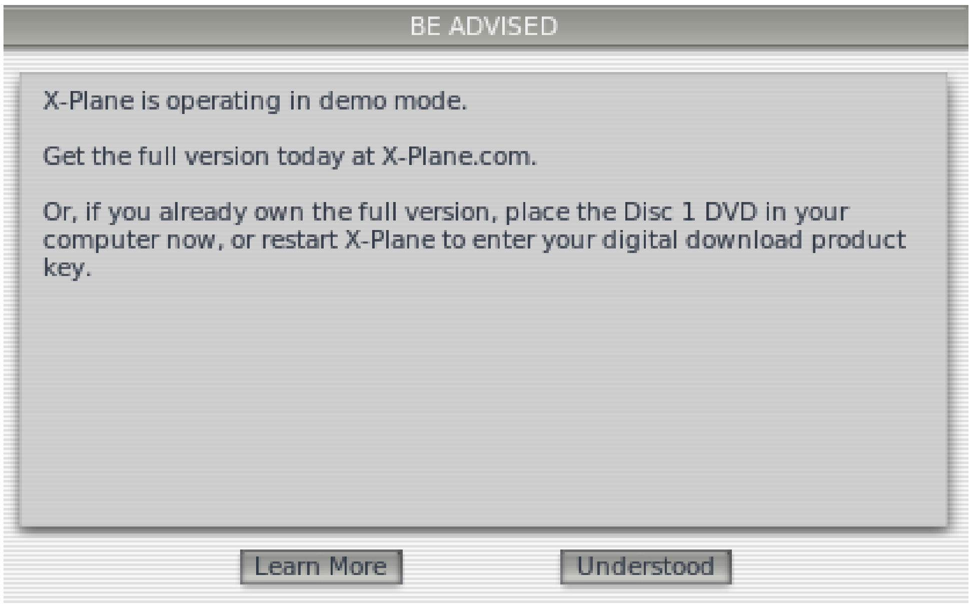


Figure 4.9 Airbus A380 from X-Plane.



BE ADVISED

X-Plane is operating in demo mode.

Get the full version today at X-Plane.com.

Or, if you already own the full version, place the Disc 1 DVD in your computer now, or restart X-Plane to enter your digital download product key.

Figure 4.10 Second selection.

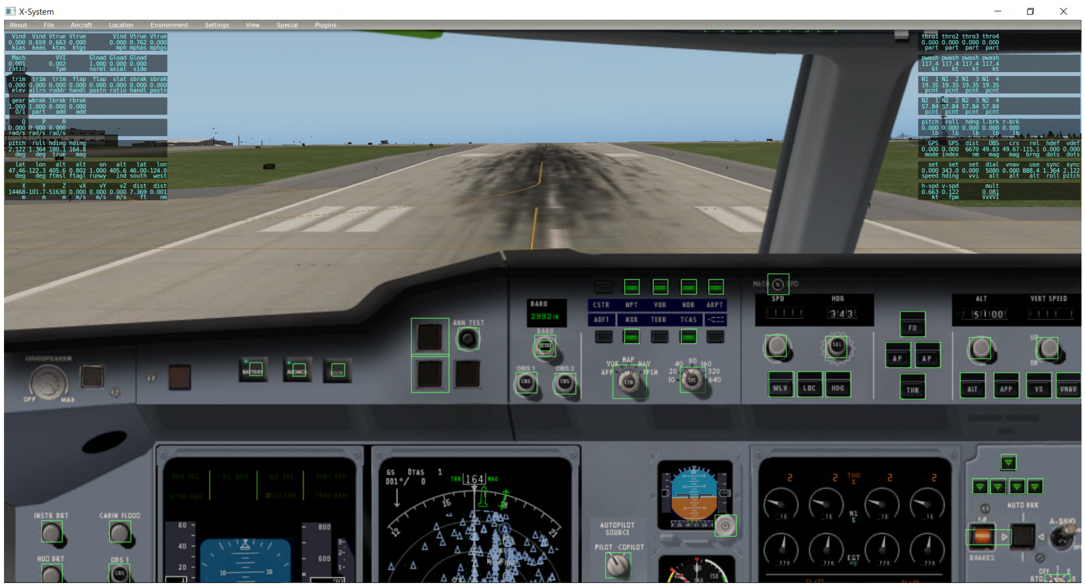
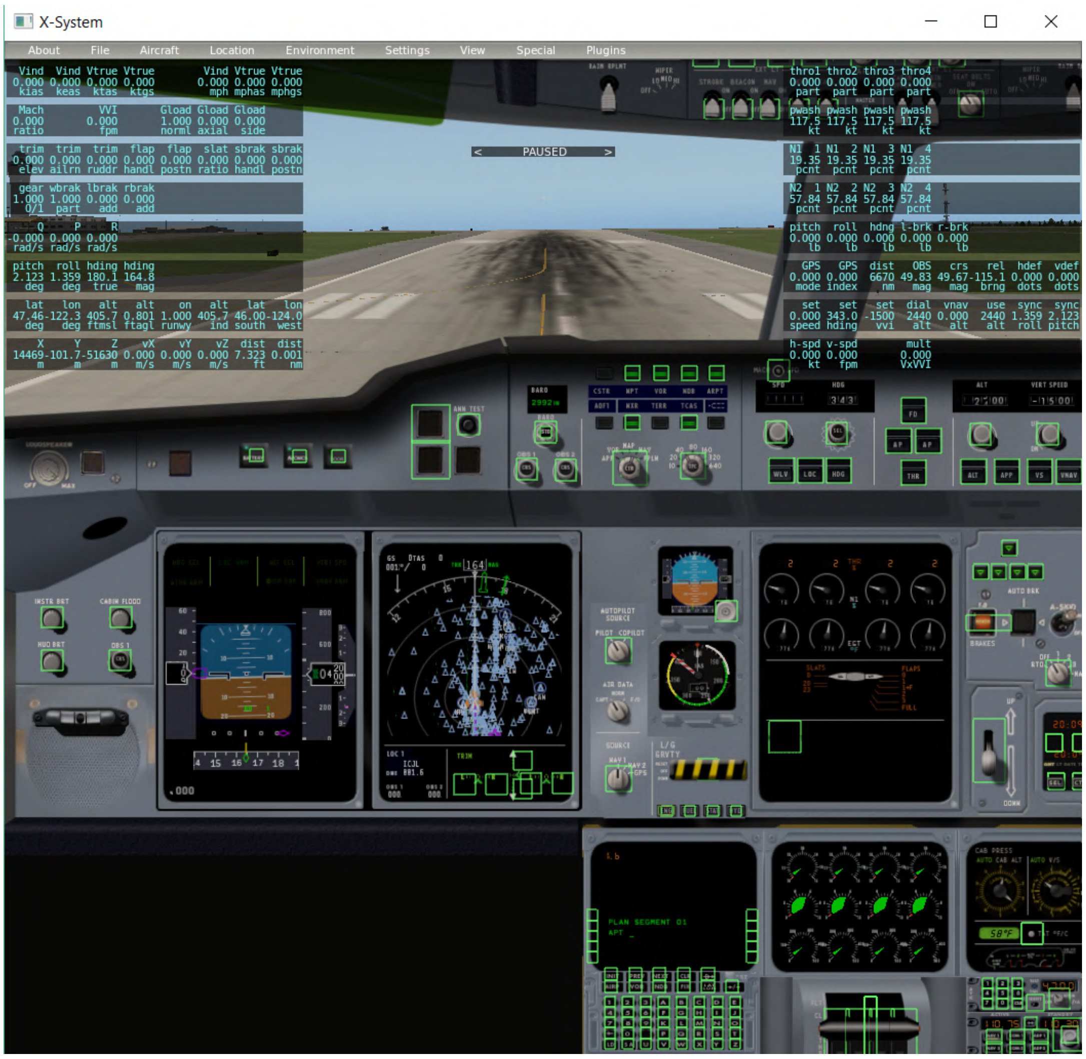


Figure 4.11 Cockpit in full screen.

  - author of Air Traffic Control
  - source of airspace data
  - source of Instrument Approach Data: Lufthansa Systems, important to this project because of making possible the autopilot can be ordered and the data is taken.
- **Instructions**, in Figure 4.14, with several tabs that provide information about:
  - flight controls
  - cockpit control
  - keyboard
  - ATC
  - tech support

Besides, the following options should be selected:



```
 Vind  Vind Vtrue Vtrue       Vind Vtrue Vtrue
0.000 0.000 0.000 0.000      0.000 0.000 0.000
 kias  keas  ktas  ktgs        mph mphas mphgs

 Mach         VVI       Gload Gload Gload
0.000       0.000       1.000 0.000 0.000
ratio         fpm       norml axial  side

 trim  trim  trim  flap  flap  slat sbrak sbrak
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
 elev ailrn ruddr handl postn ratio handl postn

 gear wbrak lbrak rbrak
1.000 1.000 0.000 0.000
  0/1  part   add   add

    Q     P     R
-0.000 0.000 0.000
rad/s rad/s rad/s

pitch  roll hding hding
2.123 1.359 180.1 164.8
  deg   deg  true   mag

  lat   lon   alt   alt    on   alt   lat   lon
47.46-122.3 405.7 0.801 1.000 405.7 46.00-124.0
  deg   deg ftmsl ftagl runwy   ind south  west

    X     Y     Z    vX    vY    vZ  dist  dist
14469-101.7-51630 0.000 0.000 0.000 7.323 0.001
    m     m     m   m/s   m/s   m/s    ft    nm
```

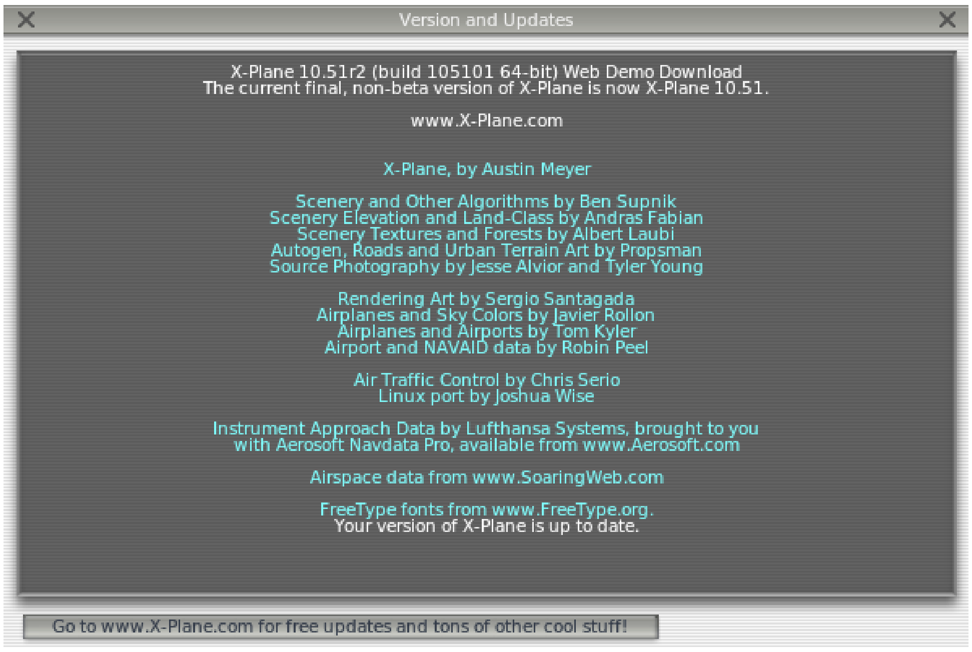
```
thro1 thro2 thro3 thro4
0.000 0.000 0.000 0.000
 part  part  part  part

pwash pwash pwash pwash
117.5 117.5 117.5 117.5
   kt    kt    kt    kt

N1  1 N1  2 N1  3 N1  4
19.35 19.35 19.35 19.35
 pcnt  pcnt  pcnt  pcnt

N2  1 N2  2 N2  3 N2  4
57.84 57.84 57.84 57.84
 pcnt  pcnt  pcnt  pcnt

pitch  roll  hdng l-brk r-brk
0.000 0.000 0.000 0.000 0.000
   lb    lb    lb    lb    lb

  GPS   GPS  dist   OBS   crs   rel  hdef  vdef
0.000 0.000  6670 49.83 49.67-115.1 0.000 0.000
 mode index    nm   mag   mag  brng  dots  dots

  set   set   set  dial  vnav   use  sync  sync
0.000 343.0 -1500  2440 0.000  2440 1.359 2.123
speed hding   vvi   alt   alt   alt  roll pitch

h-spd v-spd        mult
0.000 0.000       0.000
   kt   fpm       VxVVI
```

Figure 4.12 Cockpit in half screen.

- **Show mouse click-regions in the cockpit**, to get the green lines that point to the zones of the screen that can be changed
- **Show instrument instructions in the cockpit** in Figure 4.14 to obtain the information about them, passing the mouse over those areas

The submenu **File** has the options of:

- **QuickFlight Setup**, going back to the initial screen to choose airport, airplane, time and weather
- **Save/Load situation**
- **Save/Load replay**
- **Load Xavion Flight**, to see flights performed with Xavion in iPads or iPhones
- **Load Flight Data Recorder File**, to set atmospheric conditions and the state of the aircraft
- **Take Screenshot**
- **Toogle AVI Movie**, to start recording
- **AVI Movie Specs**, to specify the characteristics of the recording
- **Logbook**, with some information about every flight performed in X-Plane
- **Quit** the simulation



X-Plane 10.51r2 (build 105101 64-bit) Web Demo Download
The current final, non-beta version of X-Plane is now X-Plane 10.51.

www.X-Plane.com

X-Plane, by Austin Meyer

Scenery and Other Algorithms by Ben Supnik
Scenery Elevation and Land-Class by Andras Fabian
Scenery Textures and Forests by Albert Laubi
Autogen, Roads and Urban Terrain Art by Propsman
Source Photography by Jesse Alvior and Tyler Young

Rendering Art by Sergio Santagada
Airplanes and Sky Colors by Javier Rollon
Airplanes and Airports by Tom Kyler
Airport and NAVAID data by Robin Peel

Air Traffic Control by Chris Serio
Linux port by Joshua Wise

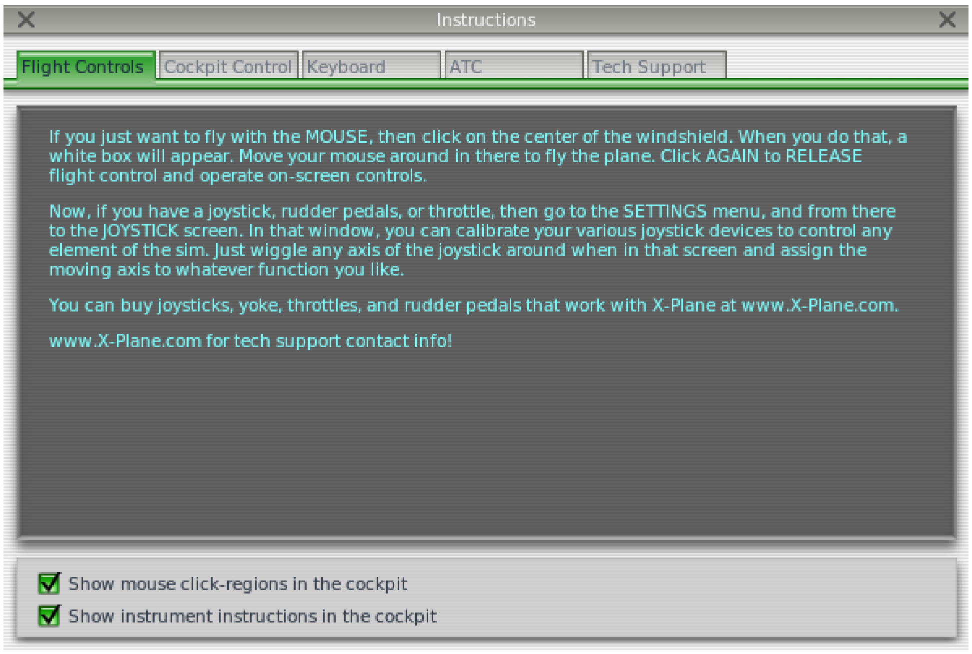
Instrument Approach Data by Lufthansa Systems, brought to you
with Aerosoft Navdata Pro, available from www.Aerosoft.com

Airspace data from www.SoaringWeb.com

FreeType fonts from www.FreeType.org.
Your version of X-Plane is up to date.

Go to www.X-Plane.com for free updates and tons of other cool stuff!

Figure 4.13 X-Plane information.



If you just want to fly with the MOUSE, then click on the center of the windshield. When you do that, a white box will appear. Move your mouse around in there to fly the plane. Click AGAIN to RELEASE flight control and operate on-screen controls.

Now, if you have a joystick, rudder pedals, or throttle, then go to the SETTINGS menu, and from there to the JOYSTICK screen. In that window, you can calibrate your various joystick devices to control any element of the sim. Just wiggle any axis of the joystick around when in that screen and assign the moving axis to whatever function you like.

You can buy joysticks, yoke, throttles, and rudder pedals that work with X-Plane at www.X-Plane.com.

www.X-Plane.com for tech support contact info!

✓ Show mouse click-regions in the cockpit
✓ Show instrument instructions in the cockpit

Figure 4.14 Aids about instructions in cockpit.

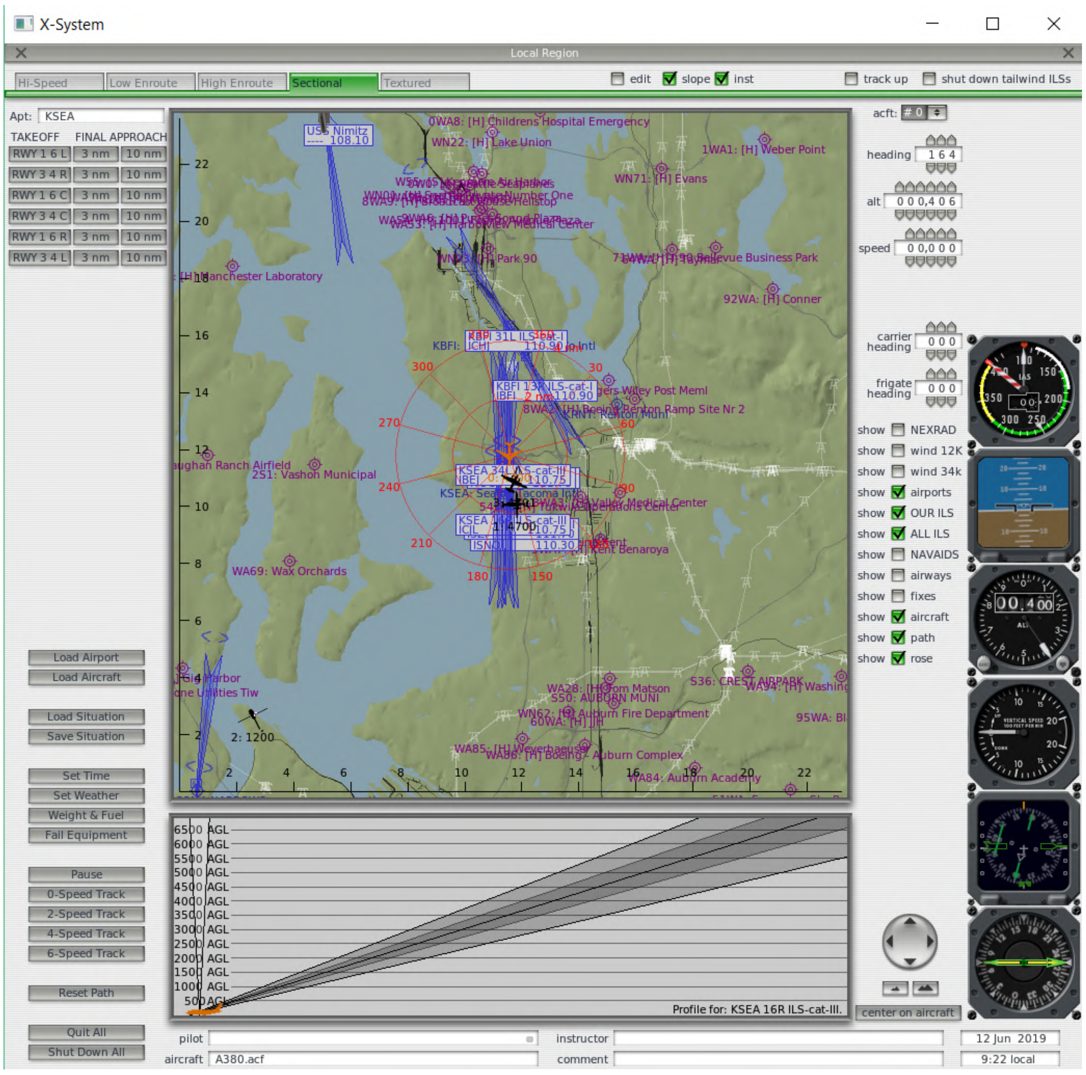
In the **Aircraft** module, it can be defined all its characteristics:

- **Open Aircraft**
- **Weight and Fuel**
- **Equipment Failures**
- **Aircraft and Situations**
- **Others**

If an aircraft is not included in the program but it is wanted to be tested, there are several options, buying it in X-Plane or searching it in forums (free version developed by users and not the official ones) and copying its data inside Aircraft in X-Plane folder, or it could be designed in a special tool related to X-Plane, the Plane-Maker.

Regarding the **Location**, four options are displayed:

- **Select Global Airport**, which allows to choose the aircraft location as a ramp, or a runway in order to perform a take off or a final approach from two possible points (3 or 10 nautical miles)
- **Local map** in Figure 4.15 offers different views of the navigation maps, aids and a vertical cut of the glideslope. Besides, it indicates in a label the name of the runway and its frequency associated. It is employed to tune and start the ILS communication when the aircraft enters in the corresponding zone. On the left part, the different approaches are offered to send the airplane to those locations, as substitute of the submenu **Select Global Airport**.
- **Planet Map**, an overview of the planets and the airports
- **Get Me Lost**, to go to an arbitrary airport



Apt: KSEA

TAKEOFF	FINAL APPROACH	
RWY 1 6 L	3 nm	10 nm
RWY 3 4 R	3 nm	10 nm
RWY 1 6 C	3 nm	10 nm
RWY 3 4 C	3 nm	10 nm
RWY 1 6 R	3 nm	10 nm
RWY 3 4 L	3 nm	10 nm

acft: # 0
heading 1 6 4
alt 0 0 0,4 0 6
speed 0 0,0 0 0
carrier heading 0 0 0
frigate heading 0 0 0

show NAVRAD ☐
show wind 12K ☐
show wind 34k ☐
show airports ✓
show OUR ILS ✓
show ALL ILS ✓
show NAVAIDS ☐
show airways ☐
show fixes ☐
show aircraft ✓
show path ✓
show rose ✓

Profile for: KSEA 16R ILS-cat-III.

pilot
aircraft A380.acf
instructor
comment
12 Jun 2019
9:22 local

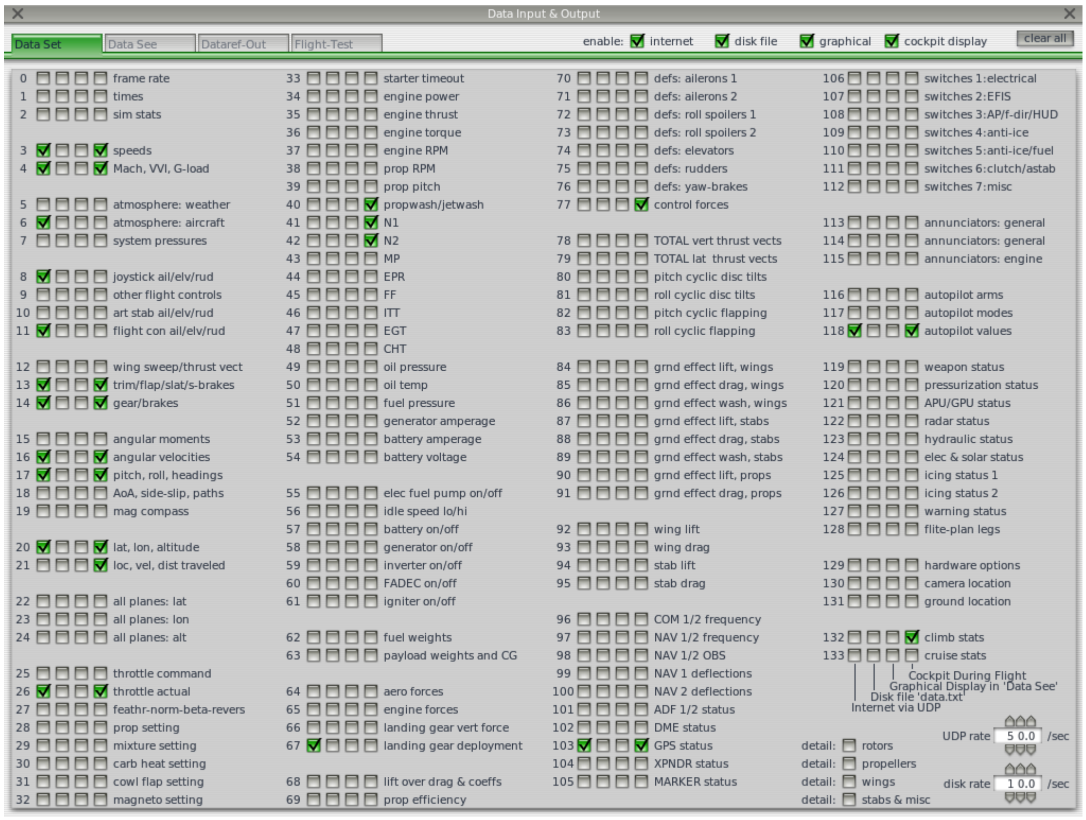
Figure 4.15 Local map.

The **Environment** controls:

- **Weather**, with more possibilities than those offered in Figure 4.3
- **Date and time**

Settings are another remarkable piece.

- It starts with the **Data Input and Output**. Data can be shown in cockpit during flight if it is selected or sent and received via UDP with an Internet connection, or to a disk file 'data.txt' or in Data See with a Graphical Display. However not all the data are interesting to the purpose of this project and other options are required to optimize the data recovery. This is one of the reasons why the X-Plane Connect from NASA is being employed. The important aspects are UDP and disk rates which determine the transfer of information.
- **Net Connections**, with computer IPs for multiplayers, instructors,...
- **Joystick and Equipment**. If external devices (joystics, yokes, pedals, handles...) are going to be used instead of the default controls, they have to be connected to the computer before launching X-Plane. Their configuration is done during the loading of the program and in this submenu, the characteristics can be changed.



enable: ✓ internet ✓ disk file ✓ graphical ✓ cockpit display

0	☐	☐	☐	☐	frame rate
1	☐	☐	☐	☐	times
2	☐	☐	☐	☐	sim stats
3	✓	☐	☐	✓	speeds
4	✓	☐	☐	✓	Mach, VVI, G-load
5	☐	☐	☐	☐	atmosphere: weather
6	✓	☐	☐	☐	atmosphere: aircraft
7	☐	☐	☐	☐	system pressures
8	✓	☐	☐	☐	joystick ail/elv/rud
9	☐	☐	☐	☐	other flight controls
10	☐	☐	☐	☐	art stab ail/elv/rud
11	✓	☐	☐	☐	flight con ail/elv/rud
12	☐	☐	☐	☐	wing sweep/thrust vect
13	✓	☐	☐	✓	trim/flap/slat/s-brakes
14	✓	☐	☐	✓	gear/brakes
15	☐	☐	☐	☐	angular moments
16	✓	☐	☐	✓	angular velocities
17	✓	☐	☐	✓	pitch, roll, headings
18	☐	☐	☐	☐	AoA, side-slip, paths
19	☐	☐	☐	☐	mag compass
20	✓	☐	☐	✓	lat, lon, altitude
21	☐	☐	☐	✓	loc, vel, dist traveled
22	☐	☐	☐	☐	all planes: lat
23	☐	☐	☐	☐	all planes: lon
24	☐	☐	☐	☐	all planes: alt
25	☐	☐	☐	☐	throttle command
26	✓	☐	☐	✓	throttle actual
27	☐	☐	☐	☐	feathr-norm-beta-revers
28	☐	☐	☐	☐	prop setting
29	☐	☐	☐	☐	mixture setting
30	☐	☐	☐	☐	carb heat setting
31	☐	☐	☐	☐	cowl flap setting
32	☐	☐	☐	☐	magneto setting
33	☐	☐	☐	☐	starter timeout
34	☐	☐	☐	☐	engine power
35	☐	☐	☐	☐	engine thrust
36	☐	☐	☐	☐	engine torque
37	☐	☐	☐	☐	engine RPM
38	☐	☐	☐	☐	prop RPM
39	☐	☐	☐	☐	prop pitch
40	☐	☐	☐	✓	propwash/jetwash
41	☐	☐	☐	✓	N1
42	☐	☐	☐	✓	N2
43	☐	☐	☐	☐	MP
44	☐	☐	☐	☐	EPR
45	☐	☐	☐	☐	FF
46	☐	☐	☐	☐	ITT
47	☐	☐	☐	☐	EGT
48	☐	☐	☐	☐	CHT
49	☐	☐	☐	☐	oil pressure
50	☐	☐	☐	☐	oil temp
51	☐	☐	☐	☐	fuel pressure
52	☐	☐	☐	☐	generator amperage
53	☐	☐	☐	☐	battery amperage
54	☐	☐	☐	☐	battery voltage
55	☐	☐	☐	☐	elec fuel pump on/off
56	☐	☐	☐	☐	idle speed lo/hi
57	☐	☐	☐	☐	battery on/off
58	☐	☐	☐	☐	generator on/off
59	☐	☐	☐	☐	inverter on/off
60	☐	☐	☐	☐	FADEC on/off
61	☐	☐	☐	☐	igniter on/off
62	☐	☐	☐	☐	fuel weights
63	☐	☐	☐	☐	payload weights and CG
64	☐	☐	☐	☐	aero forces
65	☐	☐	☐	☐	engine forces
66	☐	☐	☐	☐	landing gear vert force
67	✓	☐	☐	☐	landing gear deployment
68	☐	☐	☐	☐	lift over drag & coeffs
69	☐	☐	☐	☐	prop efficiency
70	☐	☐	☐	☐	defs: ailerons 1
71	☐	☐	☐	☐	defs: ailerons 2
72	☐	☐	☐	☐	defs: roll spoilers 1
73	☐	☐	☐	☐	defs: roll spoilers 2
74	☐	☐	☐	☐	defs: elevators
75	☐	☐	☐	☐	defs: rudders
76	☐	☐	☐	☐	defs: yaw-brakes
77	☐	☐	☐	✓	control forces
78	☐	☐	☐	☐	TOTAL vert thrust vects
79	☐	☐	☐	☐	TOTAL lat thrust vects
80	☐	☐	☐	☐	pitch cyclic disc tilts
81	☐	☐	☐	☐	roll cyclic disc tilts
82	☐	☐	☐	☐	pitch cyclic flapping
83	☐	☐	☐	☐	roll cyclic flapping
84	☐	☐	☐	☐	grnd effect lift, wings
85	☐	☐	☐	☐	grnd effect drag, wings
86	☐	☐	☐	☐	grnd effect wash, wings
87	☐	☐	☐	☐	grnd effect lift, stabs
88	☐	☐	☐	☐	grnd effect drag, stabs
89	☐	☐	☐	☐	grnd effect wash, stabs
90	☐	☐	☐	☐	grnd effect lift, props
91	☐	☐	☐	☐	grnd effect drag, props
92	☐	☐	☐	☐	wing lift
93	☐	☐	☐	☐	wing drag
94	☐	☐	☐	☐	stab lift
95	☐	☐	☐	☐	stab drag
96	☐	☐	☐	☐	COM 1/2 frequency
97	☐	☐	☐	☐	NAV 1/2 frequency
98	☐	☐	☐	☐	NAV 1/2 OBS
99	☐	☐	☐	☐	NAV 1 deflections
100	☐	☐	☐	☐	NAV 2 deflections
101	☐	☐	☐	☐	ADF 1/2 status
102	☐	☐	☐	☐	DME status
103	✓	☐	☐	✓	GPS status
104	☐	☐	☐	☐	XPNDR status
105	☐	☐	☐	☐	MARKER status
106	☐	☐	☐	☐	switches 1:electrical
107	☐	☐	☐	☐	switches 2:EFIS
108	☐	☐	☐	☐	switches 3:AP/f-dir/HUD
109	☐	☐	☐	☐	switches 4:anti-ice
110	☐	☐	☐	☐	switches 5:anti-ice/fuel
111	☐	☐	☐	☐	switches 6:clutch/astab
112	☐	☐	☐	☐	switches 7:misc
113	☐	☐	☐	☐	annunciators: general
114	☐	☐	☐	☐	annunciators: general
115	☐	☐	☐	☐	annunciators: engine
116	☐	☐	☐	☐	autopilot arms
117	☐	☐	☐	☐	autopilot modes
118	✓	☐	☐	✓	autopilot values
119	☐	☐	☐	☐	weapon status
120	☐	☐	☐	☐	pressurization status
121	☐	☐	☐	☐	APU/GPU status
122	☐	☐	☐	☐	radar status
123	☐	☐	☐	☐	hydraulic status
124	☐	☐	☐	☐	elec & solar status
125	☐	☐	☐	☐	icing status 1
126	☐	☐	☐	☐	icing status 2
127	☐	☐	☐	☐	warning status
128	☐	☐	☐	☐	flite-plan legs
129	☐	☐	☐	☐	hardware options
130	☐	☐	☐	☐	camera location
131	☐	☐	☐	☐	ground location
132	☐	☐	☐	✓	climb stats
133	☐	☐	☐	☐	cruise stats

Cockpit During Flight
Graphical Display in 'Data See'
Disk file 'data.txt'
Internet via UDP

detail: ☐ rotors
detail: ☐ propellers
detail: ☐ wings
detail: ☐ stabs & misc

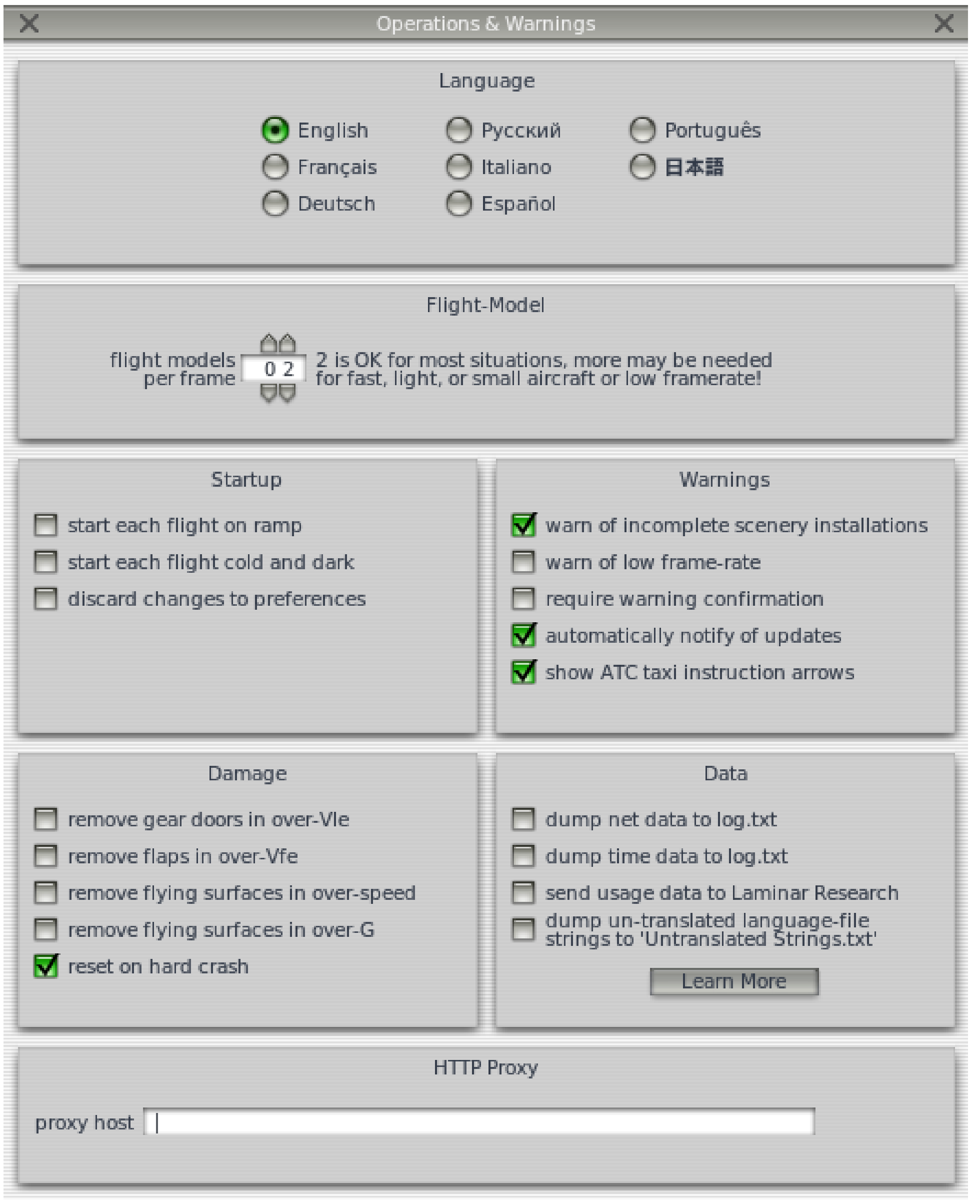
UDP rate 5 0.0 /sec
disk rate 1 0.0 /sec

Figure 4.16 X-Plane dataset.

- With **Rendering Options**, the quality is adapted according to the desired speed.
- The configuration of **Sound** allows to set the volume of engines, props, ground contact, weather, warning systems, com radio and avionics fan.
- At last, **Operations and Warnings**, in Figure 4.17 allows to set the language of preference among eight of them as well as some configuration about startup, warnings, damage and data.

All the control of visualization and cameras, can be chosen in **View**. The most used along the project have been **Forwards with panel** to control the aircraft indicators in cockpit, and **View is Ridealong** in order to watch the extraction of the landing gear and contact with the ground, which is wanted to be as smooth as possible for the comfort of the passengers. Another example could be Figure 4.18, with the 3D view of the cabin and all the controls and indicators in a more realistic disposition. This can be compared with the real one of Figure 4.19, from a 360° view in [49]. The different options classified in groups are:

- **Forwards with Panel/HUD/Nothing**
- **3-D Cockpit Command-Look**


Operations & Warnings

Language

[✓ English] Русский Português
Français Italiano 日本語
Deutsch Español

Flight-Model

flight models per frame 0 2 2 is OK for most situations, more may be needed for fast, light, or small aircraft or low framerate!

Startup

- [ ] start each flight on ramp
- [ ] start each flight cold and dark
- [ ] discard changes to preferences

Warnings

- [✓] warn of incomplete scenery installations
- [ ] warn of low frame-rate
- [ ] require warning confirmation
- [✓] automatically notify of updates
- [✓] show ATC taxi instruction arrows

Damage

- [ ] remove gear doors in over-Vle
- [ ] remove flaps in over-Vfe
- [ ] remove flying surfaces in over-speed
- [ ] remove flying surfaces in over-G
- [✓] reset on hard crash

Data

- [ ] dump net data to log.txt
- [ ] dump time data to log.txt
- [ ] send usage data to Laminar Research
- [ ] dump un-translated language-file strings to 'Untranslated Strings.txt'

HTTP Proxy

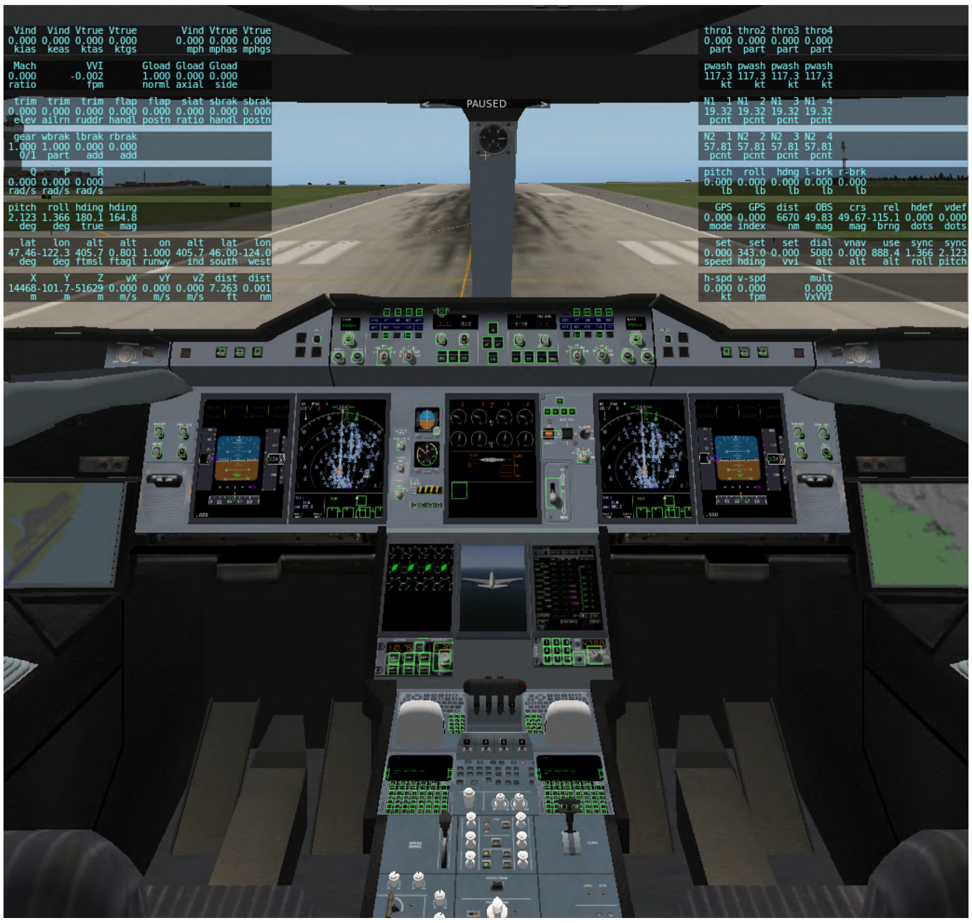
proxy host

Figure 4.17 Operations and Warnings.

- **View is Ridealong/Chase/Circle/Still Spot/Linear Spot**
- **View is Beacon Tower/on Runway/on Weapon/Free-Camera**
- **Translate Left/Right/Up/Down/Fore/Aft**
- **Rotate Left/Right/Up/Down**
- **Zoom In/Out**
- **Toogle Cinema Verite/Sunglasses/Night Vision Goggles/Aviation Flashlight**

In **Special**, extra properties are found such as:

- **Find Pitch/Yaw Stability Derivative**
- **Show/Output Flight Model**
- **Open/Toggle Checklist for Use**
- **Open/Toggle Text File for Viewing**
- **Toggle Nearby Air Traffic Controllers in Window**



```
 Vind  Vind Vtrue Vtrue        Vind Vtrue Vtrue
0.000 0.000 0.000 0.000       0.000 0.000 0.000
 kias  keas  ktas  ktgs         mph mphas mphgs

 Mach         VVI       Gload Gload Gload
0.000      -0.002       1.000 0.000 0.000
ratio         fpm       norml axial  side

 trim  trim  trim  flap  flap  slat sbrak sbrak
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
 elev ailrn ruddr handl postn ratio handl postn

 gear wbrak lbrak rbrak
1.000 1.000 0.000 0.000
  0/1  part   add   add

    Q     P     R
0.000 0.000 0.000
rad/s rad/s rad/s

pitch  roll hding hding
2.123 1.366 180.1 164.8
  deg   deg  true   mag

  lat   lon   alt   alt    on   alt   lat   lon
47.46-122.3 405.7 0.801 1.000 405.7 46.00-124.0
  deg   deg ftmsl ftagl runwy   ind south  west

    X     Y     Z    vX    vY    vZ  dist  dist
14468-101.7-51629 0.000 0.000 0.000 7.263 0.001
    m     m     m   m/s   m/s   m/s    ft    nm
```

```
thro1 thro2 thro3 thro4
0.000 0.000 0.000 0.000
 part  part  part  part

pwash pwash pwash pwash
117.3 117.3 117.3 117.3
   kt    kt    kt    kt

N1  1 N1  2 N1  3 N1  4
19.32 19.32 19.32 19.32
 pcnt  pcnt  pcnt  pcnt

N2  1 N2  2 N2  3 N2  4
57.81 57.81 57.81 57.81
 pcnt  pcnt  pcnt  pcnt

pitch  roll  hdng l-brk r-brk
0.000 0.000 0.000 0.000 0.000
   lb    lb    lb    lb    lb

  GPS   GPS  dist   OBS   crs   rel  hdef  vdef
0.000 0.000  6670 49.83 49.67-115.1 0.000 0.000
 mode index    nm   mag   mag  brng  dots  dots

  set   set   set  dial  vnav   use  sync  sync
0.000 343.0 0.000  5080 0.000 888.4 1.366 2.123
speed hding   vvi   alt   alt   alt  roll pitch

h-spd v-spd        mult
0.000 0.000       0.000
   kt   fpm       VxVVI
```

Figure 4.18 3D cockpit.

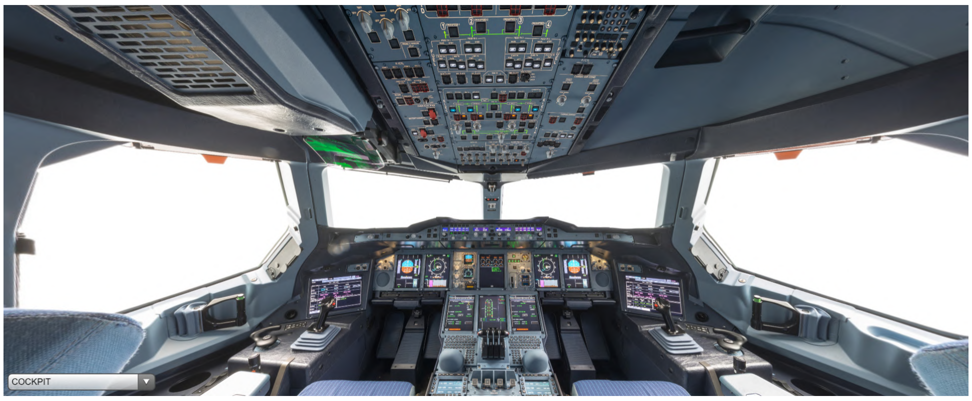


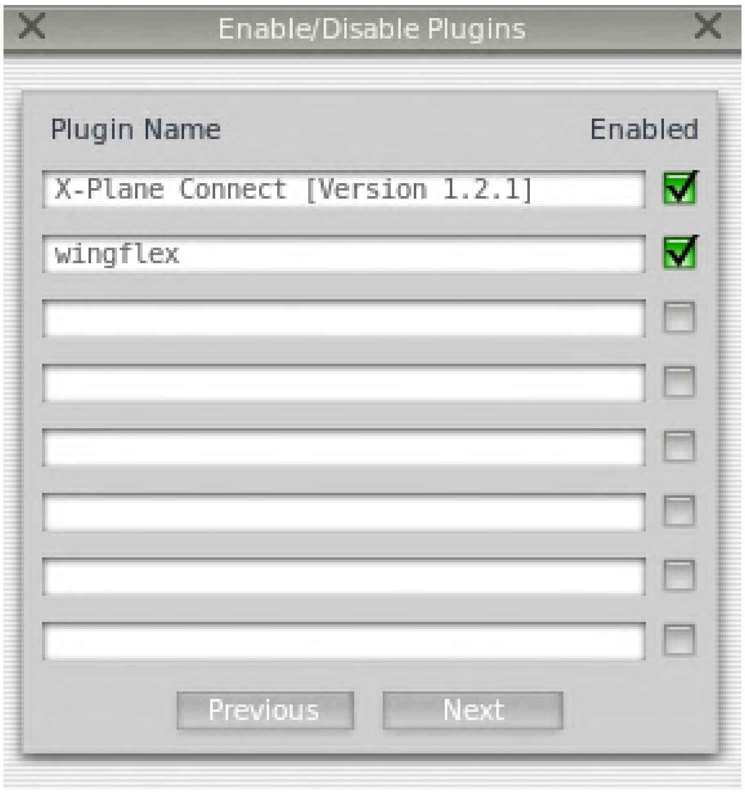
Figure 4.19 Real A380 cockpit from a 360° view in [49].

- **Set Environment Properties/Artificial Stab Constants/Autopilot Constants/FADEC Constants**
- **Show Control Deflections/Bouncers/Weapons Guidance/Projector Setup/Sky Colors/Dev Console**

Finally, **Plugins** is the other important submenu. It allows the connections with MATLAB.

- After selecting **Enable/Disable**, a window is opened, showing all the plugins available. In order to see them, they have to be previously in the folder plugins inside of Resources, X-Plane 10 subfolder, like in Figure 5.1.
- **Plugin Information**

- **HotKey Admin**


Enable/Disable Plugins

Plugin Name	Enabled
X-Plane Connect [Version 1.2.1]	✓
wingflex	✓

Figure 4.20 Enable/Disable Plugins.

Once this is all set, the following step is to learn some controls.

4.2 Controls and displays

It is important to understand the purpose of the different controls and indicators in the cabin. A real cockpit with its main parts is in Figure 4.21 to compare with the presented in the simulator. The main difference is that the pedals are substituted by a stick to control the rudder and by a bottom to brake the plane. From left to right and from the top to the bottom of the panel in Figure 4.11, the most important zones of the cockpit are described.

Autopilot (Figure 4.22) can be found at the top right part of the cockpit, and it consists on controls and modes to make the plane fly in.

The autopilot controls: speed, heading, altitude and vertical speed settings in the black rectangles can be changed to indicate the autopilot's targets (to reach and hold them).

The autopilot modes are those which aircraft can be configured to behave in a specific way. From left to right:

- WLV: autopilot engage attitude-hold. The autopilot will hold the plane's wings level to maintain the current heading. This is the simplest autopilot mode, often found on general aviation aircraft.
- LOC: autopilot engage VOR/localizer. The autopilot will fly the selected VOR bearing (if it has a plan with segments between established points) or follow the approach localizer whose frequency is tuned in the selected Nav receiver, at the right bottom of Figure 4.6. Nav receiver and OBS must be set up before engaging this mode and the aircraft have to fly a path that crosses the selected course to let the autopilot lock on.
- HDG: autopilot engage heading. The autopilot will fly the heading indicated in the heading selected bug on the directional gyro or HSI and in the heading display. The desired heading must be set in the described control before engaging this mode.

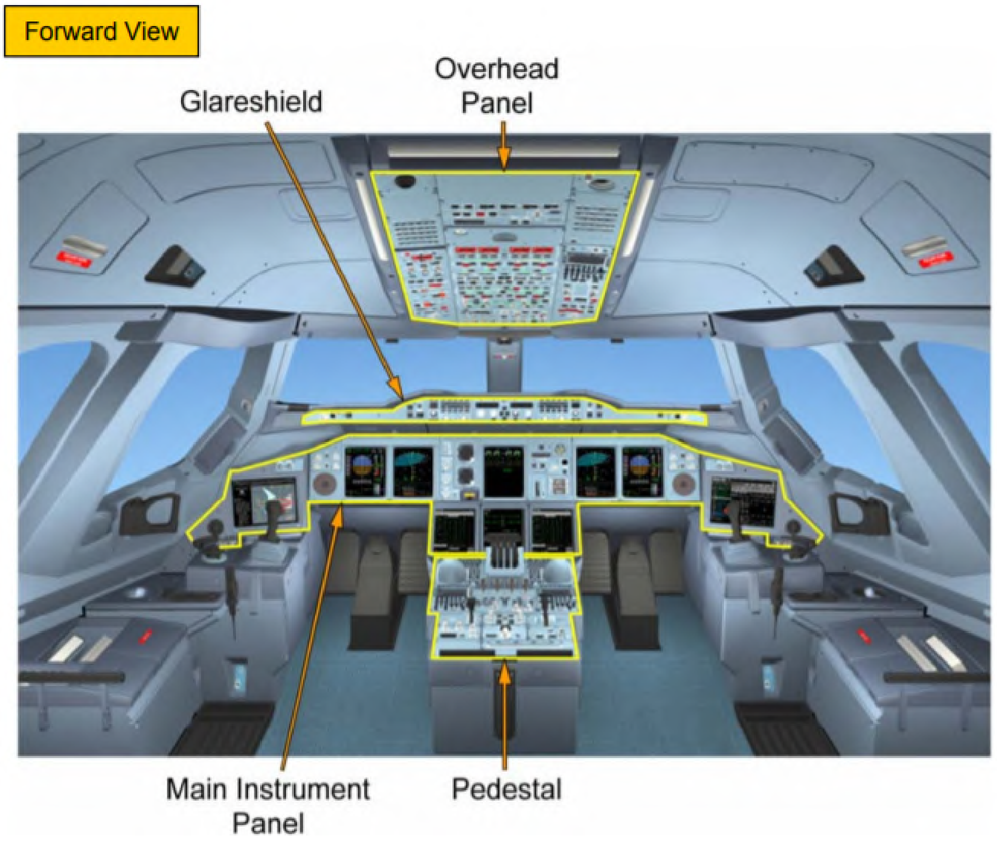


Figure 4.21 Parts of the cockpit [50].

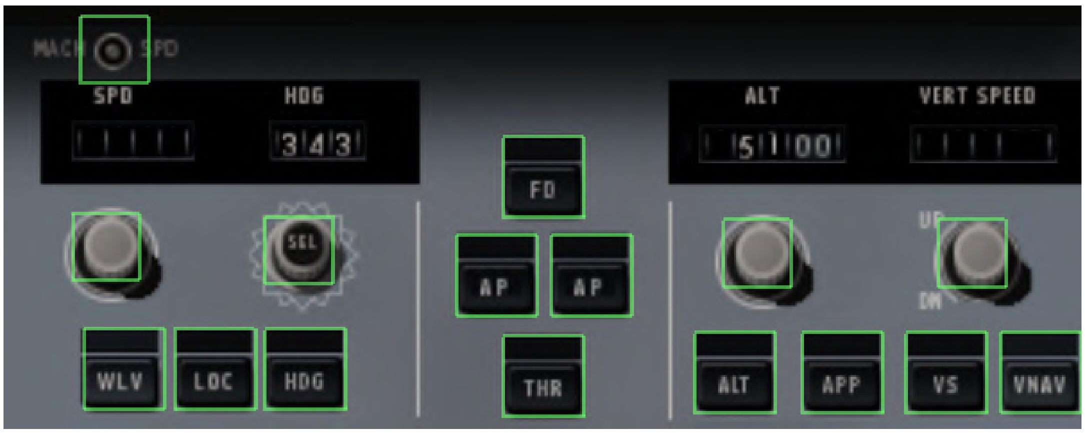


Figure 4.22 Autopilots controls and modes.

- FD: autopilot flight-director
- AP: autopilot servo
- THR: autopilot engage auto-throttle. The autopilot will adjust the throttle to maintain the set speed (it has to be chosen before engaging this mode).
- ALT: autopilot engage altitude hold. The autopilot will lock onto and maintain the plane's current altitude.
- APP: autopilot approach mode. This primes the autopilot to grab both a localizer and glideslope.
- VS: autopilot engage vertical speed. The autopilot will fly the plane at the climb rate selected in the vertical speed setting (positive value to climb, negative to descend).

- VNAV: autopilot engage Flight Management System (FMS) Vertical Navigation (VNAV). The autopilot will fly the plane at the altitude programmed into the flight management computer.

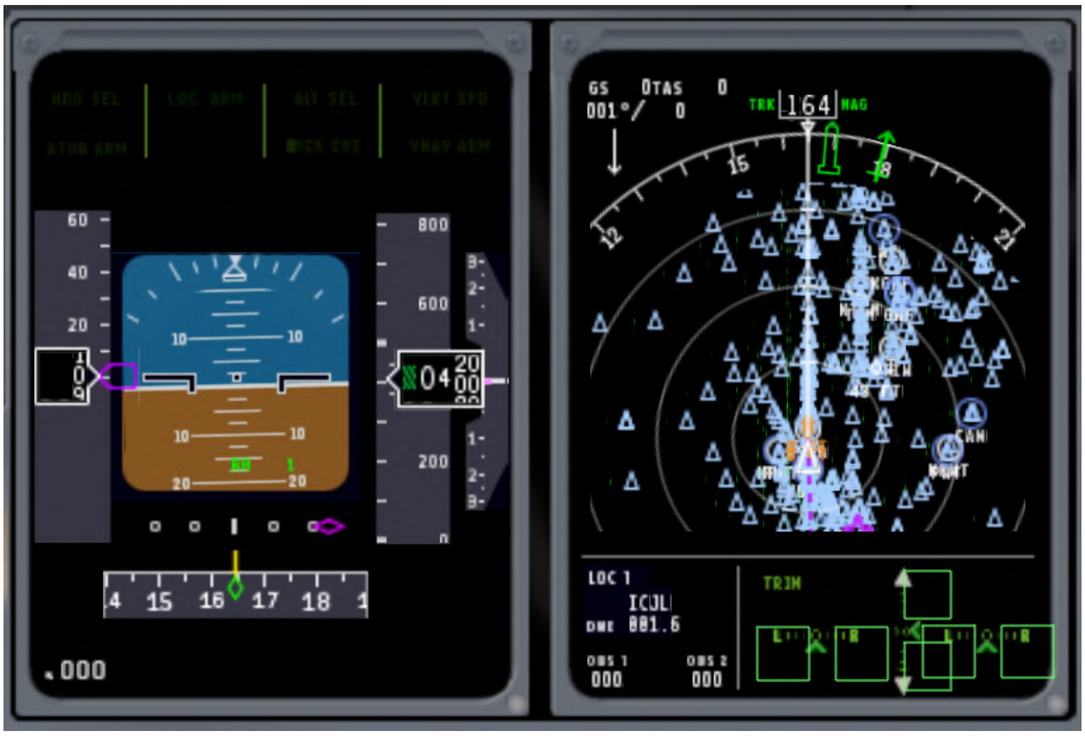


Figure 4.23 Navigation screens.

In Figure 4.23, two screens are displayed.

- The left one includes:
  - active aircraft modes
  - airspeed, artificial horizon
  - altitude above sea level
  - rate of climb
  - horizontal situation indicator (HSI).
- The right one is subdivided in:
  - the EFIS map, a moving map that shows airports, navaids, weather and other air traffic in the area near the plane.
  - the top part, which presents data such as true airspeed, the speed over the ground and the relative wind direction with an arrow and its compass direction.
  - the bottom part, with the distance to the navaid tuned to and the course selected with the OBS indicators are found; as well as different trim options are offered for aileron, rudder, elevator...

The main screen of Figure 4.24 is divided in three parts:

- Engine information: engine N1 (the speed at which the first stage (outer) compressor is turning in a turbine engine, as a percent of the original maximum design speed) and exhaust gas temperature (the temperature of the exhaust gases leaving the engine, measured at the exhaust manifold of a piston engine and at the exhaust port of a turbine engine)
- Flaps and slats indicator, which points the current position of them.
- Warning System: inverter warning for electronics, autopilot disconnection, thrust reverse indicator, slats out, engine fire, oil pressure, fuel pressure, hydraulic pressure and fuel quantity.

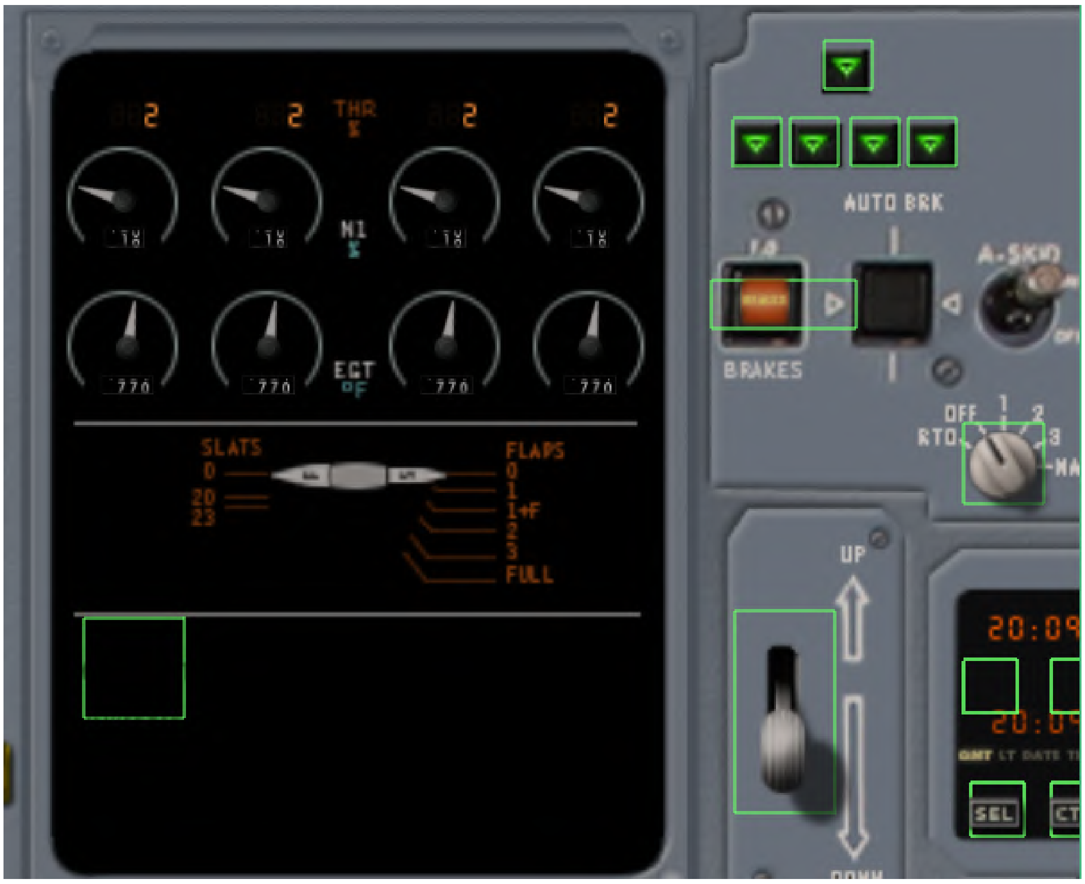


Figure 4.24 General characteristics and landing gear indicators.

Regarding the landing gear, on the right of Figure 4.24:

- There is an indicator for each gear (green when it is down and locked, unlit when the gear is up and locked and red when it is in transit.
- At the bottom, a lever raises and lowers the landing gear, with its wheel-shaped end not to be confused with something else.
- The brakes control the turn of wheels to slow or stop the plane when it is on the ground. In real planes, they are not an indicator but operated by pedals. The autobrake selector enables automatic braking on touchdown and controls how hard the brakes will come on.

If the projection of the present document had been in a larger scale, the Flight Management Computer (FMC) of Figure 4.25 would have allow to program a route through multiple waypoints, but the last phase of a flight, the landing, is the one been studied. The other parts are:

- The screen above the knob that controls throttle, which indicates:
  - engine pressure between the intake and exhaust
  - engine N, the speed at which the second stage (inner) compressor is turning in a turbine engine, expressed as a percent of the original maximum design speed
  - the fuel flow rate
- On the right, some atmospheric data are displayed as:
  - the cabin pressure with the cabin altitude indicator
  - the rate of change of cabin pressure as an equivalent climb rate (vertical velocity indicator)
  - the temperature

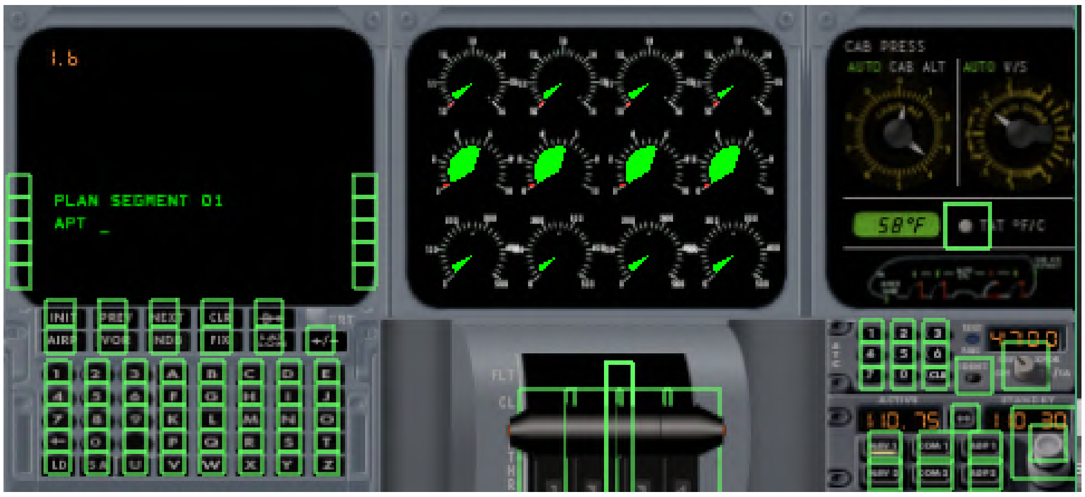


Figure 4.25 FMC, engine, throttle and communications.

- Just under this, the communications are settled with the transponder or the Nav/comm/ADF receiver, a combination receiver for communications, VOR/ILS, and non-directional beacons (all of them described in Chapter 1). First, the receiver must be selected, then with the knobs the standby frequency is set and flipping the switch it is activated. As the selected runway is 16R-34L, and its frequency, 110.75 Hz, is shown in Figure 4.26, it is what must be dialled.

Additionally, extending this image, the glideslope installation can be appreciated in Figure 4.27 with the letters GS surrounded by the blue square. Its location is defined by 793 feet (247 meters) from the extreme of the runway and 175 feet from its longitudinal axis (or 100 feet -31 meters- from the lateral of the runway), which resembles what has been exposed in Figure 1.4. As regards to the localizer, it is situated at the extreme of the runway in the direction of the landing, with its center 800 feet (243.84 meters) away from the runway in its axis.

4.2.1 Using joysticks or similar

A joystick was borrowed and configured, establishing its limits as regards its degrees of freedom: pitch and roll but not yaw or thrust control. The camera visualization was also connected. After some trials, it was seen that this external device did not provide the needed accuracy, and the author of the project hasn't had the enough knowledge to land a plane by hand. Therefore, the idea of using the autopilot and the ILS to make the most optimal number of landings came out. To interact with the simulator, the mouse and keyboard was preferred at the end.

4.2.2 Using keyboard and mouse

The controls can be configured associating some commands to keys that the user chooses and the movement of pitch, roll and yaw or the camera, to the mouse. This is done inside of **Joystick and Equipment/Keys**, in Figure 4.28.

4.3 Setting a landing manually

After this general overview, the steps to follow in order to perform a landing using X-Plane would be:

1. Starting X-Plane from X-Plane.exe
2. Clicking in the little windows in **Run Demo** (Figure 4.1) and **Understood** (Figure 4.10)
3. Choosing airport (KSEA in demo), aircraft, time and weather in Figure 4.3
4. Pausing the simulation pressing letter "P"
5. Selecting starting point (3 nmi is preferred) for the approach in **Location**, **Select Global Airport**

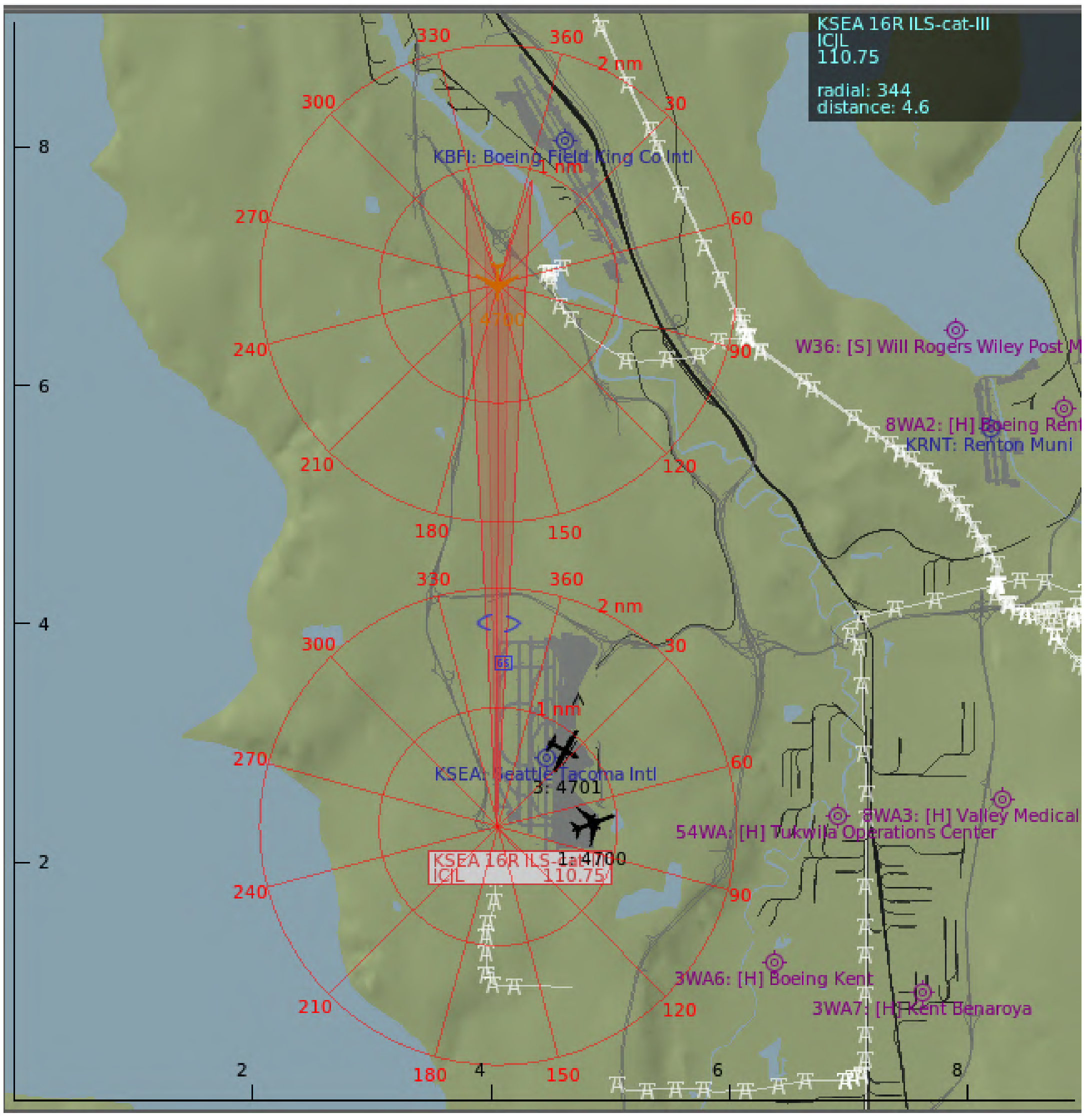


Figure 4.26 ILS of 16R runway.

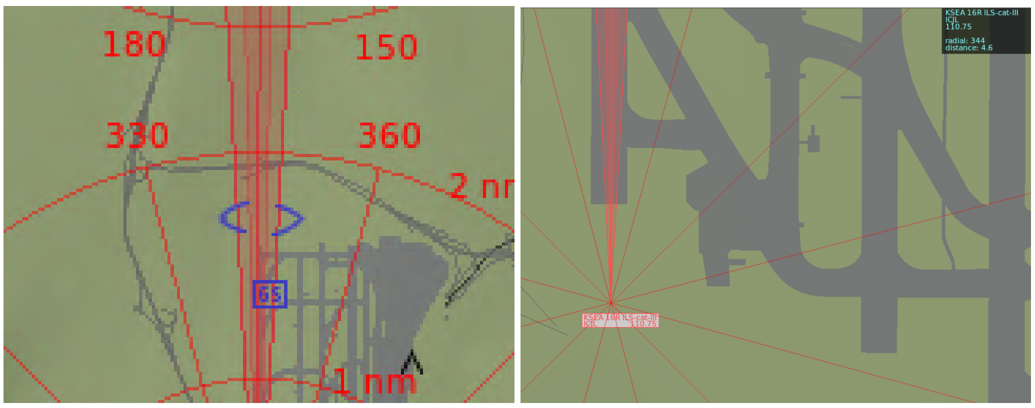
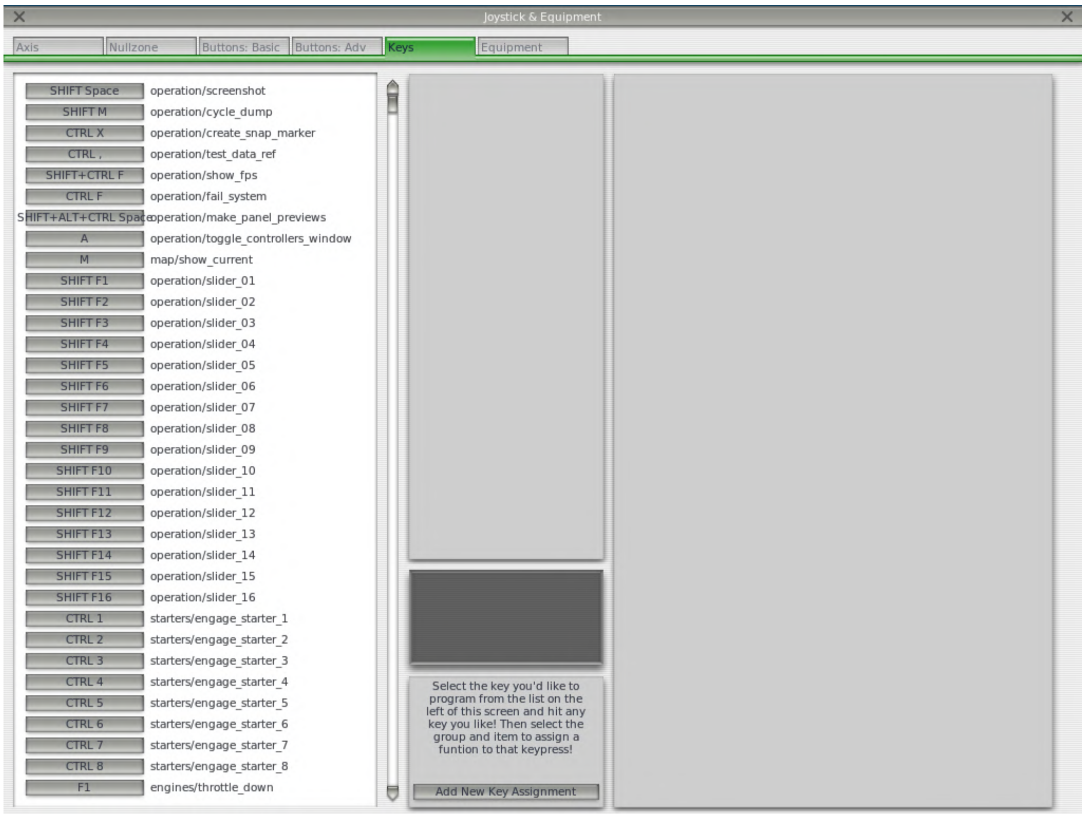


Figure 4.27 Glideslope and localizer installations 16R.



SHIFT Space	operation/screenshot
SHIFT M	operation/cycle_dump
CTRL X	operation/create_snap_marker
CTRL ,	operation/test_data_ref
SHIFT+CTRL F	operation/show_fps
CTRL F	operation/fail_system
SHIFT+ALT+CTRL Space	operation/make_panel_previews
A	operation/toggle_controllers_window
M	map/show_current
SHIFT F1	operation/slider_01
SHIFT F2	operation/slider_02
SHIFT F3	operation/slider_03
SHIFT F4	operation/slider_04
SHIFT F5	operation/slider_05
SHIFT F6	operation/slider_06
SHIFT F7	operation/slider_07
SHIFT F8	operation/slider_08
SHIFT F9	operation/slider_09
SHIFT F10	operation/slider_10
SHIFT F11	operation/slider_11
SHIFT F12	operation/slider_12
SHIFT F13	operation/slider_13
SHIFT F14	operation/slider_14
SHIFT F15	operation/slider_15
SHIFT F16	operation/slider_16
CTRL 1	starters/engage_starter_1
CTRL 2	starters/engage_starter_2
CTRL 3	starters/engage_starter_3
CTRL 4	starters/engage_starter_4
CTRL 5	starters/engage_starter_5
CTRL 6	starters/engage_starter_6
CTRL 7	starters/engage_starter_7
CTRL 8	starters/engage_starter_8
F1	engines/throttle_down

Select the key you'd like to program from the list on the left of this screen and hit any key you like! Then select the group and item to assign a funtion to that keypress!

Figure 4.28 Key association.

6. Setting the NAV receiver with the corresponding frequency. If the selected runway is 16R-34L, its frequency, 110.75 Hz (that can be obtained from **Location**, **Local Map**, as it is in Figure 4.26), it is what must be dialled.
7. Clicking THR (autothrottle), LOC (localizer) and APP (approach mode) in cockpit (FD (flight director) and AP (autopilot) must switch on themselves applying this order)
8. Pulling down the lever to deploy landing gear
9. Setting autobrake turning the knob to point at the maximum option
10. Pressing "P" in order to continue the simulation and watch how the autopilot lands the aircraft.

Now that X-Plane is controlled, the software to send orders and extract parameters, without having to repeat all this procedure each time, is going to be addressed.

5 X-Plane Connect

This chapter deals with the toolbox that connects MATLAB with X-Plane.

5.1 Installing plugin in X-Plane

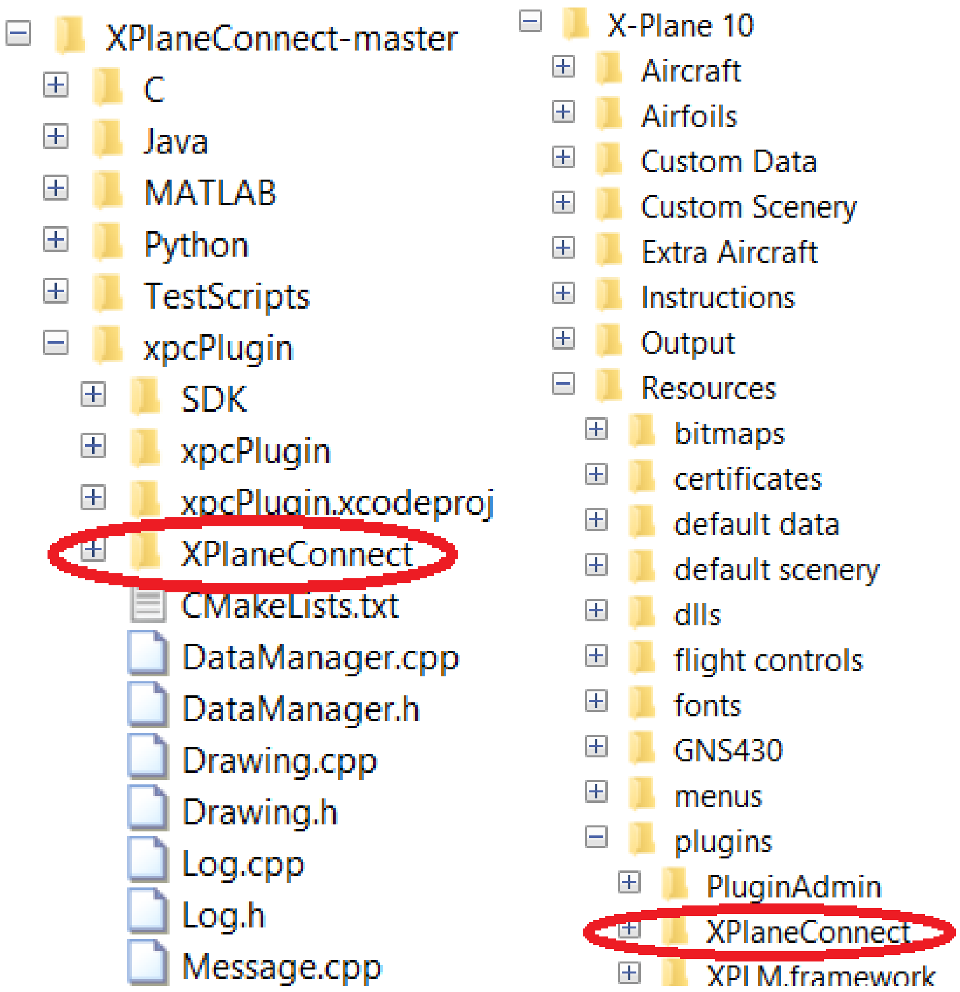
First of all, the repository [36] allows the download of the toolbox X-Plane Connect (XPC), which must be associated in X-Plane to be recognised. For that, the X-Plane Plugin folder from (xpcPlugin/XPlaneConnect of Figure 5.1, or with backslash (\) in Windows directory - it is automatically corrected) has to be copied to the plugin directory (X-Plane 10/Resources/plugins on the right of Figure 5.1). Then, X-Plane 10 can be launched and in the plugins, the XPC must be shown, as it is indicated in Figure 4.20.

5.2 Inner functions of X-Plane Connect

In the first place, getting to know the toolbox required some time because it is more complex to understand than the two previous libraries. A helpful resource has been the page where all the functions and their usage is explained [51].

Although it provides modules for C, Java, MATLAB and Python, the one to be implemented along the document is the related to MATLAB. So the functions provided in Figure 5.2, where the ones used in this project are circled in red, can be described as follows:

- *openUDP*: sets up a new connection to an X-Plane Connect pluging running in X-Plane through Internet User Datagram Protocol (UDP)
- *closeUDP*: closes a connection to X-Plane Connect
- *setCONN*: sets the port on which the client sends and receives data
- *pauseSim*: pauses or activates the X-Plane physics simulation. 1 or true to pause the simulation; 0 or false to activate and 2 to switch the pause state.
- *getDREF*: gets an X-Plane dataref
- *getDREFs*: gets one or more X-Plane datarefs
- *sendDREF*: sets the value of an X-Plane dataref
- *sendDREFs*: sets the value of one or more X-Plane datarefs. The size of value has to match the size (number) of datarefs.
- *readDATA*: reads data exported by X-Plane
- *sendDATA*: sends raw data to X-Plane
- *selectDATA*: sets which data rows X-Plane will export
- *getCTRL*: gets control surfaces information for a plane: position of latitudinal stick, longitudinal stick, pedal and throttle between -1 and 1; gear configuration (up/down), flaps deflection betweeen 0 and 1; speed brakes between -0.5 and 1.5.
- *sendCTRL*: sets control surfaces for a plane



```
⊟ XPlaneConnect-master
   ⊞ C
   ⊞ Java
   ⊞ MATLAB
   ⊞ Python
   ⊞ TestScripts
   ⊟ xpcPlugin
      ⊞ SDK
      ⊞ xpcPlugin
      ⊞ xpcPlugin.xcodeproj
      ⊞ XPlaneConnect
        CMakeLists.txt
        DataManager.cpp
        DataManager.h
        Drawing.cpp
        Drawing.h
        Log.cpp
        Log.h
        Message.cpp

⊟ X-Plane 10
   ⊞ Aircraft
   ⊞ Airfoils
   ⊞ Custom Data
   ⊞ Custom Scenery
   ⊞ Extra Aircraft
   ⊞ Instructions
   ⊞ Output
   ⊟ Resources
      ⊞ bitmaps
      ⊞ certificates
      ⊞ default data
      ⊞ default scenery
      ⊞ dlls
      ⊞ flight controls
      ⊞ fonts
      ⊞ GNS430
      ⊞ menus
      ⊟ plugins
         ⊞ PluginAdmin
         ⊞ XPlaneConnect
         ⊞ XPLM.framework
```

Figure 5.1 Folder to be copied and where to place it.

- *getPOSI*: gets position and orientation of a plane: latitude, longitude, roll, pitch, true heading (in degrees), altitude above mean sea level (in meters).
- *sendPOSI*: sets position and orientation of an airplane
- *sendTEXT*: draws text on the screen
- *sendVIEW*: sets the X-Plane cameras
- *sendWYPT*: draws waypoints in the simulated world

The difference between data and datarefs is that, while the first one is defined according the corresponding values in the X-Plane UDP Data screen (this isn't enough information for the project), the second one is chosen in MATLAB independently of the screen of X-Plane.

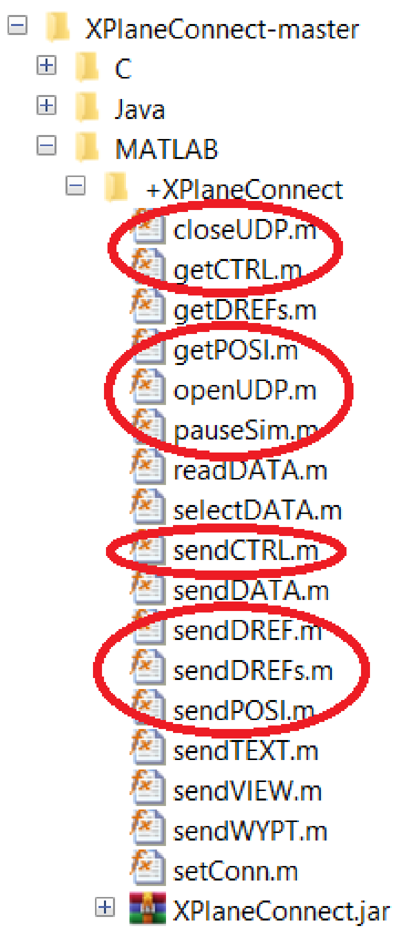
From now on, the term "data" will be used as a shortcut of datarefs throughout the document.

5.3 Data

In [52], a detailed list of all the possible data or commands to extract/send is available. Some of them have been deprecated and substituted by others. Also as the version of the toolbox and X-Plane does not coincide, others appear to be no longer active.

The main parameters of interest for this project are the following:

- sampling time for the data to be retrieved between specific moments where the data is taken
- speed of the simulation, which also activates the simulation when it is sent to X-Plane if it was paused



```
⊟ XPlaneConnect-master
  ⊞ C
  ⊞ Java
  ⊟ MATLAB
    ⊟ +XPlaneConnect
        closeUDP.m
        getCTRL.m
        getDREFs.m
        getPOSI.m
        openUDP.m
        pauseSim.m
        readDATA.m
        selectDATA.m
        sendCTRL.m
        sendDATA.m
        sendDREF.m
        sendDREFs.m
        sendPOSI.m
        sendTEXT.m
        sendVIEW.m
        sendWYPT.m
        setConn.m
    ⊞ XPlaneConnect.jar
```

Figure 5.2 Inner functions, those relevant to the project circled in red.

- autothrottle
- localizer, to set the localizer communication
- approach mode, to set the glideslope communication
- gear status (down or up)
- autobrake ratio, to brake just when the touchdown is produced
- indicated airspeed
- groundspeed, the condition to stop taking data
- latitude
- longitude
- altitude in feet above mean sea level
- altitude in meters measured by radioaltimeter above ground level
- pitch
- roll
- heading
- pitch ratio
- roll ratio
- heading ratio
- throttle ratio

- flap ratio
- vertical velocity

5.3.1 Notes

The deprecated data that could have been used in the project are:

- airspeed_mode
- heading_mode
- altitude_mode
- mode_hnav
- mode_gls

All the autopilot modes have been implemented as flags in autopilot_state [53]. They have three states: off, arm and engage. The difference between arm and engage is that the mode is loaded and it will be activated when certain conditions are reached, passing from arm to engage. For example altitude hold arm will engage altitude hold when the plane reaches the target altitude dialled into the autopilot dash.

Some modes can't be overwritten and others must have a specific order to set them correctly. For instance, to set up an approach, only the heading mode has to be set first, then the localizer arm, otherwise, there could happen that the implementation of the localizer comes first if they are set at the same time. The autopilot must have always one lateral and one vertical mode. And from that, some modes cancel other automatically.

Editing a variable that has associated several fields, -998 can be written in those that aren't wanted to overwrite them. That is, the parameter will stay at the current X-Plane value if -998 is assigned to it.

Reading of several parameters from aircraft sensors tends to lag by several second, like the indicator of the vertical velocity: *vvi_fpm_pilot*; and their exact measure have to be obtained using other methods. It has been decided to use discretised difference quotient, being *h_i* the value of the altitude at the previous instant of time, *h_f* the one of the current altitude and *interval* the step of time. If *h_i* is empty, in the first instant of time, it has been considered $h\_i = h\_f$ and therefore, the vertical velocity is null, $vvi = 0$. For the rest of instants, the vertical velocity is calculated as:

$$vvi = \frac{h\_f - h\_i}{interval} \cdot 60 \tag{5.1}$$

Code 5.1 SERP3nmiapp.

```
if cont==1
    h_i = posi(3)*3.28084; % set h_i to calculate vertical velocity
    %initial altitude in feet in order to calculate the vertical
    %velocity by deriving the altitude with incremental cocients
    %due to the lag of the vvi of the airplane
end
h_f = posi(3)*3.28084; %new altitude in feet
vvi = (h_f-h_i)/(interval)*60; % vertical velocity feet/min
h_i = h_f;
```

5.3.2 Equipment of the aircraft

In this project, the simulated aircraft will be equipped with the following sensors, available as normal instrumentation in modern airplanes, to allow the extraction of the data described in the previous subsection.

- Radar altimeter to get the altitude above ground level.
- Instruments of attitude: gyroscopes and accelerometers of Air Data and Inertial Reference System (ADIRS) with pitch, roll and heading and their derivatives and accelerations.
- Sensor of airspeed.

- Multi-Function Probe to provide the pressure from which obtain altitude measured from mean sea level.
- GPS, that provides the location in latitude and longitude mainly.
- Sensor of gear state.
- Weight On Wheels (WOW) sensor, to start braking as soon as the airplane touches ground, when this sensor activates.
- Throttle sensor.

These assumptions are based on the document from [50]. In this, several images of the real cockpits as well as a detailed description of all the components and systems of A380-800 can be appreciated.

5.4 Own code in X-Plane Connect

Using the provided X-Plane Connect code as a starting point, a number of customized pieces of code were developed in order to fulfil the requirements of the work at hand. The code is named *SERP3nmiapp* and can be found in section Subsection 9.1.1.

It is worth remarking that this code has been extended and adapted to be applied to all kind of airports, aircraft, and situations related to landing operations. This versatility requirement has increased the difficulty of the development of codes, but will be quite useful to future works.

5.4.1 Start X-Plane

Before the program is executed, the simulator has to be running with the situation 3nmi, at 16R runway of KSEA, paused since the beginning (letter "P" must be pressed when being at the runaway or during the loading of the approaching). Several seconds have to pass before running the file.

If it is preferred, X-Plane can be executed from MATLAB with the orders of changing folder to the directory which contains X-Plane.exe and running it preceded with the bang operator (!). The instructions must be executed from the Command Window, where they can be implemented.

5.4.2 Import X-Plane Connect

The path is set to contain the inner functions described in Section 5.2 and then the connection is established with X-Plane importing the plugin XPlaneConnect.

5.4.3 Setup

Each iteration (landing) is saved in two files and in order to generate automatically a new file, persistent variable *num* is created. The path of the files is added and then they are opened to write in them.

- File of extension .txt, where in the first row some info about the landing is saved:
  - *plane*, A380 by default in this project
  - *airport*, KSEA by default in this project
  - *weather*, with the possible options of Section 4.1
  - *time*, with the possible options of Section 4.1
  - *distance*, starting point around that distance to the airport
  - *land_way*, landing runway in 16R or in 34L
  - *start_point*, starting point from the default point ('default') or a sent point ('s')
  - *direction*, landing from north to south ('NS', 16R) or from south to north ('SN', 34L)

  The second row contains the name of each variable and its units (if it's a dimensional one) are presented, with all the data by columns and indicating at the end the total number of requested connections, the successful ones and the percentage of connectivity. Some samples aren't taken due to UDP timeouts, originated by bad WIFI/Internet connection, or a low efficiency of the computer that causes the delay of the simulation.

- File of extension .mat, with the name of the variables and their data as vectors of the same number of components, to be used with the IRL algorithm in Chapter 7. It can't have all the extra information that the previous document allows.

The characteristics of the connections are fixed opening the socket with the inner function *openUDP*, indicating the interval of data acquisition, 0.2 s or 5 Hz, 20 samples per second. Then the simulation speed can be selected as 1 for real time, 2, for accelerating time to double the real one or 0 to pause it.

5.4.4 Set the initial position

The software provides two points (3 nmi and 10 nmi) from the chosen landing strip or runway. However if the aim is to study and take data of different casuistry, this fixed point does not provide a complete point of view.

To ensure that, the initial point is given around 3nmi, the maximum distance the glideslope and localizer operate. The general characteristics considered are:

- A normal altitude for the considered distance to the airport
- The throttle and controls required to land correctly that have been left to the autopilot
- The landing gear deployment in a certain distance from the airport given as the latitude coordinate. Since the starting point is around the 3 nmi from the runway, this is an adequate distance to deploy it. With this, the indicated airspeed at threshold in the landing reaches the required 138 knots for this aircraft, A380.

It is important to point out that in order to send a specific location with latitude, longitude and altitude, all the rest of parameters must have been configured. The easiest way of doing that is pausing the simulation, and then sending the plane to the default 3 nmi point from X-Plane. Once there, the position can be set to another point from MATLAB, for example, A, maintaining the pitch, roll, heading and gear the same as they were in the default position of 3 nmi, putting -998 in their respective values. A remarkable aspect is the direction of the aircraft, if the landing is performed flying to the north, the chosen runway to take the point of approach must be the 34L one. If the 16R was selected, the plane would continue to south from the point A when it was sent.

Besides, the given point has to be really near of the default points 3 nmi of 16R or 34L, according to the direction the landing is wanted; however it doesn't limit the first default point of the simulator. In fact, being in the point of 3 nmi of 34Rit, it is possible to send A near 34L due to their same direction. This two paragraphs could be more understandable observing the Figure 4.6.

5.4.5 Set initial conditions: autothrottle, localizer, approach

After selecting the datarefs to work with: *autopilot_state*, *gear_handle_status*, *auto_brake_settings* and *sim_speed*; they are defined with *sendDREF*. The order to perform a good approach using ILS is putting the flag of autopilot_state: 1 - autothrottle, 256 - localizer, 1024 - approach mode.

At last, the gear is extracted and the autobrakes set to 4 the maximum, to start their function when the plane touches the ground.

It is important to know that if the starting point is far from the airport, the approach mode will be armed but will not engage until it reaches the 3 nmi from the airport and even it will avoid it if the aircraft changes its course excessively. In the meantime, another mode is armed, such as autopilot engage attitude-hold or the altitude hold. However, if the dialled altitude is reached, the autopilot will try to maintain it and will not land. To avoid that, a high value of altitude must be dialled. It happens the same with the vertical velocity and the heading. This last one has a huge influence on the aircraft when it is dialled the opposite to the one being followed. The aircraft tries to reach it too quickly and ends falling.

5.4.6 Parameter extraction

The parameters that are required to perform the subsequent data analysis have their datarefs written in a vector, *DREF*.

A variable, *cont* keeps track of the requested connections every 0.2 s and another registers if a connection is perfectly performed of *conn*. It doesn't update its value if any failure has been produced.

A loop is initialized with an inner stop condition in its interior. Every step time, a new iteration is done and the code tries to obtain the data. However, when for some reason the connection can not be executed, the code shows an error in Command Window (instead of the data of that step of time) and counts a fail using the catch part, but doesn't stops running the program and tries the next iteration. Hence, to end a iteration, the stop condition must be accomplished or the user has to use in Command window the keys of the keyboard: "Ctrl"+"C".

DREF parameters are sent as request to X-Plane to receive their values with *getDREFs*. They along with some of the data from *getPOSI* and *getCTRL* (saved in *posi* and *ctrl* respectively) are shown in the Command Window each step time and recorded in the corresponding file. It is remembered the exception of the vertical velocity indicator that is calculated deriving the altitude.

The process of selecting the adequate variables to extract is determined by the subsequent study in the algorithm. A plane is a source of parameters and the efficient way is not to extract all of them. This would not be correct due to the quantity of time required and the limitations of the processing capacity. On the other hand, an insufficient number of data would result in a inexact control. Therefore, the following parameters have been selected:

- *conn*, number of successful connection and the component of the vectors
- *lat*, latitude in degrees from the first component of *posi*
- *long*, longitude in degrees from the second component of *posi*
- *altmsl* altitude in meters measured from the mean sea level, converted to feet
- *altagl*, altitude in meters measured from above ground level, converted to feet (third component of *posi*)
- *pitch*, pitch of the aircraft,fifth component of *posi*
- *roll*, roll of the aircraft, fourth component of *posi*
- *head*, heading of the aircraft, sixth component of *posi*
- *pitchr*, pitch ratio flying with autopilot, deflection of elevator
- *rollr*, roll ratio flying with autopilot, deflection of ailerons
- *headr*, heading ratio flying with autopilot, deflection of rudder
- *Tr*, throttle ratio of all the engines
- *vvel*, vertical velocity in feet per minute differentiating the altitude with incremental quotients (the decrement divided the interval)
- *Vg*, ground speed in m/s, as the condition to stop taking data
- *V*, indicated airspeed in knots, to proof the threshold speed (138 knots in A380)
- *flaps*, flap ratio
- *gear*, gear status, down or up
- *autob*, autobrake selector, set to the maximum to brake just as the touchdown is performed

The stop condition is used to finish saving data when the aircraft is not moving in the runway after landing. It is described by having an altitude under than 10 feet above ground level, with an ground speed inferior to 0.001. The last thing added to the file of extension .txt (and printed to the Command Window) is the number of successful connections and the percentage of connectivity. Finally, the connection between X-Plane and MATLAB is ended with *closeUDP*.

One of the important aspects to consider is that due to the inexperience of the student in charge, with this kind of software, it has been decided to take the information of the performance of the autopilot employing the ILS described in Section 1.3.

5.5 Considered cases

The situations vary in time of day, the meteo (with the implied turbulences) and the starting location. However all of them have in common the aircraft (A380 and the airport (KSEA).

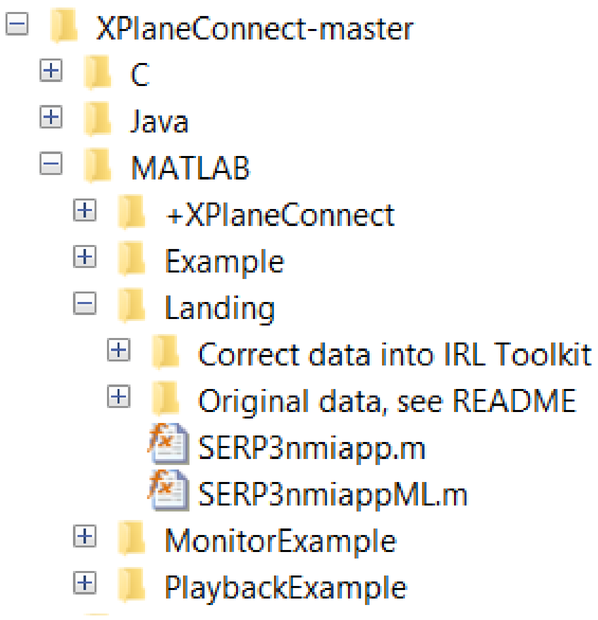
As it has been said, in spite the fact that the unique plane considered has been A380, any model with autopilot and the required modes can be chosen to extract data from it.

After hundreds of failed iterations while testing and modifying the code to include more data or information until its current version, a number of ten landings have been correctly and completely performed. In the Table 5.1 is shown their information.

Table 5.1 First iterations.

Number	Time	Weather	Runway	Starting point
1	day	clear	34L	default
2	day	clear	34L	sent
3	night	clear	16R	default
4	sunset	foggy	34L	default
5	sunset	stormy	16R	default
6	sunset	stormy	34L	sent
7	twilight	foggy	16R	sent
8	twilight	foggy	16R	default
9	sunset	broken	16R	sent
10	sunset	broken	34L	sent

These can be found in the created folder Landing with the code that has generated them, as it is shown in Figure 5.3.



```
⊟ XPlaneConnect-master
  ⊞ C
  ⊞ Java
  ⊟ MATLAB
    ⊞ +XPlaneConnect
    ⊞ Example
    ⊟ Landing
      ⊞ Correct data into IRL Toolkit
      ⊞ Original data, see README
        SERP3nmiapp.m
        SERP3nmiappML.m
    ⊞ MonitorExample
    ⊞ PlaybackExample
```

Figure 5.3 Developed code and folder of iterations.

As a first approach to the problem, a limited number of scenarios have been considered. The small set is nonetheless sufficient to test the performance of the proposed method in the IRL Toolkit of Chapter 7, and can be extended in future reviews of the work. That is because it will be necessary to obtain more robust results training the machine, before letting it to determine the optimal action to follow in X-Plane.

6 Machine Learning (Inverse Reinforcement Learning)

As it has been described in Section 1.4, Machine Learning (ML) is going to be employed to make the simulator learn from recorded data. In that way the automatic system that is been developed will be able to land the plane without human supervision. To this goal, lots of tests should be done to certify the system.

The branch of ML to be used in this project is going to be based is Inverse Reinforcement Learning (IRL). It has the inverse setting of Reinforcement Learning (RL). What distinguishes them is needed to be understood in order to clarify what is given and what is searched in each one. [54]

On the one hand, in RL the agent (machine) tries to find the optimal actions that have to be implemented in given states to maximize the reward in a environment.

On the other, in IRL, the aim is to find a reward function that explains the behaviour given a policy or a set of trajectories performed by an expert.

As reported by [26], it is characterized informally as follows:

Given:

- measurements of an agent's behaviour over time, in a variety of circumstances
- if needed, measurements of the sensory inputs to that agent (environment states)
- if available, a model of the environment (transition probabilities)

Determine: reward function to optimize

The two given last points appear in any reinforcement learning, so the first one is which determines the definition of an IRL problem. In Table 6.1 their differences are exposed.

Table 6.1 Differences between RL and IRL.

	RL	**IRL**
given	reward function R	policy π or set of trajectories
determine	optimal policy π	reward function R

IRL and Inverse Optimal Control (IOC) are often used as synonyms. The most important difference to be aware of is that optimal control talks about costs while IRL talks about rewards. Rewards can be turned into costs and vice versa by adding a negative sign.

The biggest motivation for IRL is the difficulty in designing manually a reward function for a task, which becomes much more complex as more parameters have to be considered in the definition of the state.

According to [26], the reward function, rather than the policy, supposes the best definition of the task, because of being:

- succint, with the desired states, instead of the policy, which needs the description of what the behaviour should look like

- robust, adapting to any variations of the environment in order to accomplish the objective
- transferable to a different set of actions, for example, from humans to robots

Therefore, it is better to learn the underlying reward function than to copy a behaviour. That's why Behaviour Cloning isn't applied. In spite of that, this has been combined with Reinforcement Learning in [24] to show how they are powerful enough to learn how to fly an aircraft through different points in space and different turbulence conditions.

With the aim of making decisions, Markov Decision Processes provide a mathematical framework in situations where outcomes are partly random and partly under the control of a decision maker.

6.1 Markov Decision Processes

It consists on a tuple $(S, A, R, P_{sa}, \gamma)$ [26], where:

- S is the set of n environment states
- $A = \{a_1,...,a_k\}$ is the set of k actions
- R is the real reward function which maps each state to a real valued reward $S \to \mathbb{R}$. It is only known by the demonstrator or expert, while the estimated one is E, also called $\hat{R}$. Another term related with this is the expected reward, $R(s,a)$, for state-action pairs with SxA domain instead of $R(s)$, defined as a function of states. The conversion from one to another is explained in [55], and can't be found in papers generally although it is mentioned.
- T contains the transition probabilities, the probability of getting state $s' \in S$ after performing action $a \in A$ in state $s \in S$
- $\gamma \in [0,1)$ is the discount factor for future rewards. In each step, the reward is updated multiplying by the discount once
  - Sooner rewards probably do have higher utility than later rewards
  - Also helps the algorithm to converge

A policy π maps states to actions. It can be:

- stationary deterministic, which outputs one deterministic action for each state: $\pi : S \to A$
- stochastic, containing probabilities of choosing each action for each state: $\pi : SxA \to [0,1]$.

The observed behaviour can be provided as a set of trajectories $\xi = \{(s_0,a_0),(s_1,a_1),...\} = \xi_1,\xi_2,...$, a ordered in time list of states and actions performed by an expert or sampled from some policy, instead of the policy itself.

The value of a state s under policy π is denoted as $V^\pi(s)$. By the first theorem of Bellman Equations, this value is defined as:

$$V^\pi(s) = R(s) + \gamma \sum_{s' \in S} T(s'|s,\pi(s))V^\pi(s') \tag{6.1}$$

Besides, the value of state-action pairs under some police π is denoted as

$$Q^\pi(s,a) = R(s) + \gamma \sum_{s' \in S} T(s'|s,a)V^\pi(s') \tag{6.2}$$

The second theorem determines the optimal police in and only if, for all $s \in S$,

$$\pi(s) \in argmax_{a \in A} Q^\pi(s,a) \tag{6.3}$$

The optimal policy π^* maximizes the expected discounted sum of rewards $E[\sum_{t=0}^{\infty} \gamma^t R|\pi^*]$.

6.2 IRL algorithms

The different algorithms depend on the definition of the problem. In a basic approach to the problem, three cases can be found according to [26], although lately new methods have been developed.

- Optimal policy π is known, in a small state space
- Optimal policy π is known, in a large or infinite state space
- Optimal policy π is unknown, but behaviour trajectories ξ are given

The third one is the most realistic case and the one to be dealt with in the present project. All of them are extensively develop in [26].

6.2.1 IRL from sampled trajectories

This method does not require an explicit model of the MDP, though it is assumed the ability to find an optimal policy under any reward. The algorithm is also presented with features of the form $f : S \to \mathbb{R}$ that can be used to represent the unknown reward R.

The initial state distribution $\xi = \xi_1, \xi_2, ..., \xi_d$ is fixed and it can be assumed that for the (unknown) policy π the goal is to find R such that π maximizes $E[V(\xi)]$. To simplify notation, only one fixed start state s_0 is considered (to generalize this fact, s_0 can be a "dummy" state whose next-state distribution under any action is ξ).

R is estimated as $\hat{R}$ using linear approximation.

$$\hat{R}(s) = \alpha_1 \phi_1(s) + \alpha_2 \phi_2(s) + ... \alpha_d \phi_d(s) \tag{6.4}$$

With $\phi_1, ..., \phi_d$ are fixed, known, bounded basis functions mapping from S into $\mathbb{R}$ and the $\alpha_i s$ are the unknown parameters wanted to "fit". This turn the difficult function search into a much easier linear combination which enables to use linear programming. The aim is to calculate the empirical value of a trajectory not the expected value of a policy, in other words, how much reward did some trajectory actually yield.

It is needed to estimate $V(\xi)$. After registering m trajectories, for each $i = 1, ..., d$, $\hat{V}_i(\xi)$ is defined to be what the average empirical return would have been on these m trajectories if the reward had been $R = \phi_i$. For instance, with only $m = 1$ and if that trajectory visited the sequence of states (s_0, s_1, ...), then:

$$\hat{V}_i(\xi) = \phi_i(s_0) + \gamma \phi_i(s_1) + \gamma^2 \phi_i(s_2) \tag{6.5}$$

$$\hat{V}_i(\xi) = \sum_{s_j \in \xi} \gamma^j \phi_i(s_j) \tag{6.6}$$

In general, $\hat{V}_i(\xi)$ would be the average over the empirical returns of m such trajectories (in practice, trajectories are also truncated after a large but finite number of steps). Then, for any setting of the $\alpha_i s$, a natural estimate of $V(\xi)$ is:

$$\hat{V}(\xi) = \alpha_1 \hat{V}_1(\xi) + \alpha_2 \hat{V}_2(\xi) + ... + \alpha_d \hat{V}_d(\xi) \tag{6.7}$$

This is justified by the fact that:

$$V(\xi) = \alpha_1 V_1(\xi) + \alpha_2 V_2(\xi) + ... + \alpha_d V_d(\xi) \tag{6.8}$$

The trajectory generated by the expert is called ξ_{π^*}. In some cases, there might be several trajectories the same expert (from different initial states for example). The weighted mean of their respective empirical values can be calculate. From now on, it is assumed that only one expert trajectory exists.

The goal is to find parameters $\alpha_i s$ such that the given expert trajectory (assumed optimal) ξ_{π^*} yields a higher empirical reward than other trajectories generated by different policies. So for the "base case" trajectory ξ_{π^1}, which is a randomly chosen policy:

$$\hat{V}(\xi_{\pi^*}) \geqslant \hat{V}(\xi_{\pi^1}) \tag{6.9}$$

However, just comparing the expert trajectory to one arbitrary random trajectory will most likely not be suffice. Optimally, it is wanted to compare to many more trajectories and especially trajectories that are more "competitive" than random policies. The "inductive step" of the algorithm is as follows. Having some set (m)

of trajectories associated to policies $\pi_1,...,\pi_k$, and wanting to find a setting of the $\alpha_i s$ so that the resulting reward function (hopefully) satisfies:

$$\forall \xi_{\pi^{\neg *}} \in \{\xi_{\pi^1}, \xi_{\pi^2}, ..., \xi_{\pi^k}\} : \hat{V}(\xi_{\pi^*}) \geqslant \hat{V}(\xi_{\pi^{\neg *}}) \tag{6.10}$$

The optimization becomes:

$$\begin{aligned} &\mathit{maximize} && \sum_{i=1}^{k} p(\hat{V}(\xi_{\pi^*}) - \hat{V}(\xi_{\pi_i})) \\ &\mathit{such\ that} && |\alpha_i| \leqslant 1, \quad i = 1,...,d \end{aligned} \tag{6.11}$$

Where, p, the penalization function penalizes violated constraints more heavily than satisfied constraints are rewarded. That has to be taken into account because in some cases the reward function cannot be expressed as linear combination of the fixed basis functions. p can be defined as follows:

$$\begin{aligned} p(x) &= x \quad if \quad x \leqslant 0 \\ p(x) &= 2x \quad if \quad x > 0 \end{aligned} \tag{6.12}$$

Here 2 is a heuristically chosen parameter, to which the results aren't extremely sensitive in general. $\hat{V}(\xi_{\pi^*})$ and $\hat{V}(\xi_{\pi^i})$ are just (implicit) linear functions of the $\alpha_i s$ as given in (6.8) and hence the problem is solved via linear programming.

The optimization gives a new setting of the $\alpha_i s$ and a new reward function $\hat{R} = \alpha_1\phi_1 + ... + \alpha_d\phi_d$. Then this trajectory $\xi_{\pi_{k+1}}$, maximizing $V(\xi_\pi)$ under $\hat{R}$, is added to the current set of policies and the process is repeated until a satisfactory $\hat{R}$ is found.

This development follows the one of [26] and [54]. It has to be said that due to the the author's inexperience in this kind of algorithms, it was decided to use a toolbox that implements them.

The first options were:

- One of Reinforcement Learning that was exactly the opposite of the situation to apply [37]
- Two of MDP solvers [38] and [39], which doesn't provide an environment, IRL algorithm or examples

The tool of choice in this case was the IRL toolkit. It is provided in [40], with three examples and lots of algorithms. Its weak points are the complexity and the scarce description available.

Continuing the mathematical description of this chapter, the paper associated to it, [28], is characterised by taking the real human demonstration as suboptimal trajectories using a probabilistic model of the expert's behaviour, as the maximum entropy IRL model (MaxEnt). This is closely related to linearly-solvable MDPs, and has been used extensively to learn from human demonstrations. Under MaxEnt, the probability of taking a path ξ_i is proportional to the exponential of the rewards encountered along that path. The model is convenient for IRL, because its likelihood is differentiable and a complete stochastic policy uniquely determines the reward function. Intuitively, such a stochastic policy is more deterministic when the stakes are high, and more random when all choices have similar value.

The structure and usage of the IRL Toolkit is presented in the next chapter.

7 IRL Toolkit

In this chapter, the toolkit used to obtain the reward and policy functions is described. It can be found in [40], based on the project in paper [28], which presents a probabilistic algorithm for nonlinear inverse reinforcement learning. While most prior inverse reinforcement learning algorithms represent the reward as a linear combination of a set of features, with Gaussian processes, GPIRL's aim is to learn the reward as a nonlinear function. At the same time it also determines the relevance of each feature to the expert's policy. This probabilistic algorithm allows complex behaviours to be captured from suboptimal stochastic demonstrations (performed by humans, or in this project by an autopilot), while automatically balancing the simplicity of the learned reward structure against its consistency with the observed actions.

One of the points that characterise this paper is that it includes most of the existent algorithms of IRL: MaxEnt, FIRL, MMP, MWAL, MMPBoost and LEARCH. The result is that the nonlinearities considered in GPIRL outperformed prior methods, especially when generalising the learned reward to new state spaces.

In the following sections, it is going to be presented a point of view of the tool focused in its use for this project. It is worth noticing that the IRL Toolbox is not a commercial product and has been developed by its authors as complement to help others to replicate their results. The package is provided as is. It is poorly documented, it's not regularly maintained and there is no support line available to handle potential errors or code bugs. Unfortunately in this project, a number of such bugs and errors have appeared, thus complicating and requiring a considerable amount of time to be dealt with.

7.1 Installation

After downloading (from [40]) and unzipping the toolkit, the file README.txt provides a succinct description of the structure and the organization of the programs.

This toolkit was built with MATLAB R2009b, and therefore, its implementation in earlier versions might need some changes in the codes, like the instruction *matlabpool* (General\runtestseries.m), which has been deprecated by *parpool* as of MATLAB R2013b.

Besides that, other bugs that have been solved are those related to the addition of the corresponding files to the path in order to execute a test. It has been done in the scripts, uncommenting *addpaths*, a function with no inputs or outputs that updates MATLAB path with all the files required. For the project, the created folder *Landing* and its subfolders have been included within this function, which allows MATLAB to access them.

The unique file that requires installation is CVX (convex optimization implemented by [56]). It can be run from Utilities\cvx_setup in Figure 7.1 as the first step of every session or added to path and executed if it is the first time, with the following lines proposed by the author of the present project:

Code 7.1 Setup of cvx.

```
global i %global and not persistent if it is not a function

if isempty(i) %running cvx_setup only the first time
    addpath Utilities/cvx
```

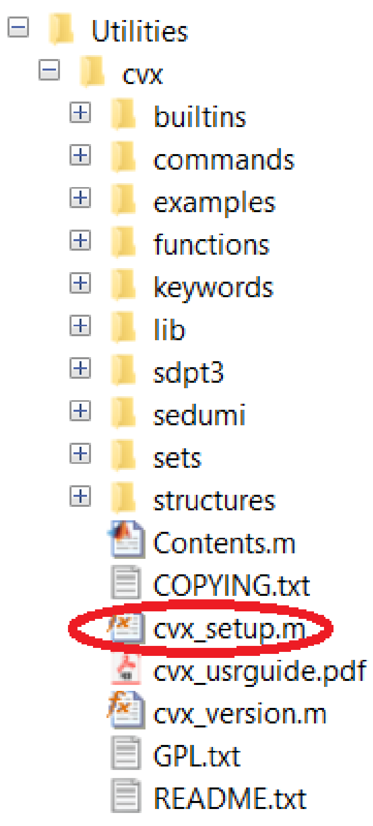


Figure 7.1 cvx_setup localization.

```
    cvx_setup
    i=1;
end

addpaths; % IMPORTANT, as it comes in the code, it is commented. It activates
%the necessary files to MATLAB
```

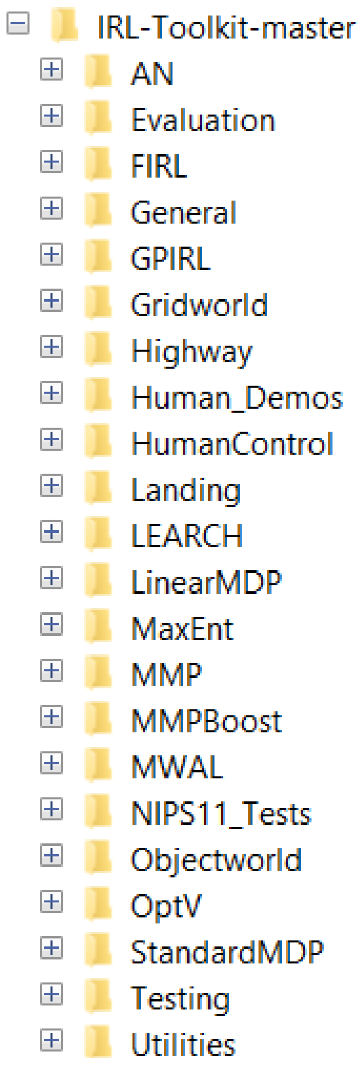
7.2 Directory overview

The main categories of the directory in Figure 7.2 are IRL algorithms, MDP solvers, example domains and miscellaneous.

7.2.1 IRL algorithms

The implemented algorithms are described and it is indicated the related paper of each one.

- *AN*: Abbeel & Ng's projection algorithm for IRL. Since Abbeel & Ng's algorithm does not return a single reward, this implementation is somewhat less accurate than the original algorithm, because a single reward function must be selected to return for evaluation. [26]
- *FIRL*: Feature Construction for Inverse Reinforcement Learning algorithm. Only compatible with discrete features (the "continuous" parameter must be set to 0 for the current example environment) [57]
- *GPIRL*: Gaussian Process Inverse Reinforcement Learning algorithm. For continuous features, it is recommended to use the warped kernel by setting the "warp_x" parameter to 1. [28]
- *LEARCH*: a simplified implementation of the Learning to Search algorithm. The current version can run either log-linear mode or nonlinear, with decision trees of logistic regression to create nonlinear features. [58]



IRL-Toolkit-master
- AN
- Evaluation
- FIRL
- General
- GPIRL
- Gridworld
- Highway
- Human_Demos
- HumanControl
- Landing
- LEARCH
- LinearMDP
- MaxEnt
- MMP
- MMPBoost
- MWAL
- NIPS11_Tests
- Objectworld
- OptV
- StandardMDP
- Testing
- Utilities

Figure 7.2 Directory of IRL Toolkit.

- *MaxEnt*: Maximum Entropy IRL algorithm. [59]
- *MMP*: Maximum Margin Planning. This implementation uses the QP formulation of MMP rather than subgradient methods, since the QP version is already very fast on the examples in this package. [60]
- *MMPBoost*: MMP with feature boosting, using decision trees of a configurable depth. It has also been simplified. [61]
- *MWAL*: the game-theoretic MWAL algorithm. [62]
- *OptV*: a provisional implementation of the OptV algorithm, that learns reward functions instead of value functions due to the framework in which it has developed, producing poor results compared to other methods in the toolkit. [63]

All of them must implement two functions: *<name>run* and *<name>transfer*, where <name> is the name of the algorithm examples domains (*gridworld*, *highway*, *objectworld* or the developed *landing*) that will be used to identify it when calling runtest, central code to execute an experiment. Additionally, all current algorithms implement *<name>defaultparams*, but this is a convenience. *<name>run* function must accept this arguments:

- *algorithm_params*, the parameters of the algorithm. Some parameters may be missing, so it is advisable to implement a *<name>defaultparams* function to fill them in (for example, see *gpirldefaultparams.m*).

- *mdp_data*, a specification of the example domain. It includes the number of states, the number of actions, the transition function (specified by *sa_s* and *sa_p*), and so forth. For details, see (for example) *gridworldbuild.m*
- *mdp_model*, the name of the current MDP model (*standardmdp* or *linearmdp*). The final returned policy must be computed from the reward function using this model, but most algorithms will not use this parameter anywhere else.
- *feature_data*, information about the features associated to the example. The main parameter of this structure is *feature_data.splittable*, which is a matrix containing the value of each feature at each state.
- *example_samples*, an important part of the program, a cell array of the examples, where *example_-samples{i,t}* is time step t along trajectory i. Each entry in the cell array has two numbers: *example_-samples{i,t}(1)* is the state, and *example_samples{i,t}(2)* is the action. This one has been generated converting the data of X-Plane to this format with the program *XPC_IRL_converter.m*, in Section 9.3, which has been explained in Section 7.5.
- *true_features*, the true features that form a linear basis for the reward. Most algorithms (that are not cheating) will ignore this, but it may be useful for establishing a baseline comparison.
- *verbosity*, the desired amount of output. 0 means no output, 1 means "medium" output, and 2 means "verbose" output.

Additionally, IRL algorithms must return the structure *irl_result* containing the result of the computation. This has the following fields:

- *r*, the learned reward function, with *mdp_data.states* number of rows and *mdp_data.actions* number of columns
- *v*, the resulting value function, returned by <*mdp_model*>*solve*
- *p*, the corresponding policy
- *q*, the corresponding q function
- *r_itr*, the reward at each iteration of the algorithm, useful for debugging; if unused, set to *r*
- *model_itr*, the model at each iteration of the algorithm, to be used for transfer
- *model_r_itr*, a second optional reward at each iteration of the algorithm, useful for debugging; if unused, set to *r_itr*
- *p_itr*, the policy at each iteration; if unused, set to *p*
- *model_p_itr*, a second optional policy at each iteration of the algorithm, useful for debugging; if unused, set to *p_itr*
- *time*, the time the algorithm spent computing the reward function (optional, set to 0 if unused)

The transfer function must use the learned model to transfer the reward function to a different state space. The output of this function is identical to <*name*>*run*. The arguments are the following:

- *prev_result*, the *irl_result* structure returned by a <*name*>*run* call.
- *mdp_data*, MDP definition for the new state space.
- *mdp_model*, MDP model, as before.
- *feature_data*, feature data for the new state space.
- *true_feature_map*, true features for the new state space (again, not used except when "cheating").
- *verbosity*, as before

If both <*name*>*run* and <*name*>*transfer* are implemented, the algorithm should be usable in all tests without further modification.

7.2.2 MDP solvers

The package includes two MDP solvers.

- *standardMDP*: standard solver that uses value iteration to recover a function, which in turn specifies a deterministic optimal policy for the MDP.
- *linearMDP*: linear MPD solver that uses soft value iteration and corresponds to linearly-solvable MDPs. It produces stochastic examples, suboptimal under the prior solver.

Also additional MDP solvers can also be added to the framework, they hasn't been it. A MDP solver must implement 5 functions:

- *<name>solve* must find the solution to the specified MDP. It has two arguments: *mdp_data*, the MDP definition, constructed by the example building function; and the reward *r*. The output is a structure *mdp_solution*, which has fields *v*, *q* and *p*, corresponding to the value function, Q function, and policy. Note that these fields will only be used by other functions relating to the MDP solver implemented, so they can contain whatever information that is chosen.
- *<name>frequency* finds the visitation frequency of all states in the MDP. This function is used by many metrics. The inputs are *mdp_data* and *mdp_solution* (from a previous call to *<name>solve*), and the output is a column vector with 1 entry for each state, giving that state's expected visitation count under the policy in *mdp_solution*.
- *<name>action* samples an action at a particular state and takes as input *mdp_data*, *mdp_solution*, and a state *s*. The output is an action *a*, which is either the optimal action under a deterministic policy or a sampled action under a stochastic policy.
- *<name>step* executes the sampled action. It takes *mdp_data*, *mdp_solution*, a state *s*, and an action *a*, and returns the resulting state after taking action *a* in state *s*, which may be sampled if the MDP is nondeterministic.
- *<name>compare* compares two policies, given as arguments *p1* and *p2*, and returns the amount of discrepancy between them. In the case of a standard MDP, this is currently the number of states in which the policies disagree. In the case of the linear MDP, this is the sum over all states of the probability that *p1* and *p2* will take different actions. This function is only used when evaluating some metrics.

7.2.3 Example domains

Three example domains are included in the toolkit.

- *Gridworld*. This is an NxN gridworld, with 5 actions per state corresponding to moving in each direction and staying in place. The *determinism* parameter specifies the probability that each action will "succeed." If the action "fails," a different random action is taken instead. The reward function of the gridworld consists of BxB blocks of cells (forming a "super grid"). The features are indicators for x and y values being below various integer values. This environment does not support transfer experiments.
- *Objectworld*. This is a Gridworld populated with objects. Each object has one of C inner and outer colors, and the objects are placed at random. There are 2C continuous features, each giving the Euclidean distance to the nearest object with a specific inner or outer color. In the discrete feature case, there are 2CN binary features, each one an indicator for a corresponding continuous feature being less than some value D. The true reward is positive in states that are both within 3 cells of outer color 1 and 2 cells of outer color 2, negative within 3 cells of outer color 1, and zero otherwise. Inner colors and all other outer colors are distractors.
- *Highway*. This is the highway environment described in "Nonlinear Inverse Reinforcement Learning with Gaussian Processes." The number of cars on the highway must be specified manually. There is an algorithm for placing cars that avoids creating roadblocks. This algorithm will stall if it cannot be placed the specified number of cars, so too much can't be introduced. It is also possible to specify more than 2 categories or classes of cars, though this functionality is currently untested.

To this project, each domain has been observed and tried to understood. After that, a new environment has been created to represent the operation of landing a plane, following the structure that defines the default ones and that is described in the following section.

Functions that define the domain

The functions that are presented in all the domains and which have to be developed are:

- *<name>build*, with a single argument, the *mdp_params* structure passed to *runtest* that specifies the parameters of the desired example. Its outputs are:
  - *mdp_data*, a structure containing the definition of the MDP. It must specify the following fields:
    - *states*, the number of states
    - *actions*, the number of actions in each state
    - *discount*, the discount factor of the MDP (currently all examples are infinite horizon discounted reward)
    - *sa_s*, 3D matrix of size *states* x *actions* x *K*, where *K* is any number. The entry *sa_s(s,a,k)* specifies the state to transition to when taking action a in state *s*, and sampling destination *k* (see below)
    - *sa_p*, 3D matrix as above, where *sa_p(s,a,k)* gives the probability of transitioning to *sa_-s(s,a,k)* on action a in state s
  - *r*, a *states* x *actions* matrix specifying the true reward function of the example environment.
  - *feature_data*, a structure containing two fields regarding the features of the example:
  - *splittable*, a *states* x *features* matrix containing the value of each feature at each state
  - *stateadjacency*, a sparse *states* x *states* matrix with "1" for each pair of states that are "adjacent" (have an action to transition from one to the other); this is currently only used by FIRL.
  - *true_feature_map*, a (sparse) *states* x *features* matrix that gives the values of the "true" features that can be linearly combined to obtain *r*.
- *<name>draw*, which must draw some visualization of the specified example to the current axes. It has no output, and takes the following input arguments:
  - *r*, the reward function to draw.
  - *p*, the policy to draw (may be deterministic or stochastic).
  - *mdp_params*, the *mdp_params* structure passed to *<name>build*.
  - *mdp_data*, the *mdp_data* structure returned by *<name>build*.
  - *feature_data*, the *feature_data* structure returned by *<name>build*.
  - *model*, the MDP model (*standardmdp* or *linearmdp*).

Additionally to these, other functions may be created:

- *<name>defaultparameters* which contains the default parameters that define the example.
- others functions needed to reduce the extension of the main functions or to avoid repetitions, concentrating all the possible change of data into them

7.2.4 Miscellaneous

These are included for operations relative to the prior functions.

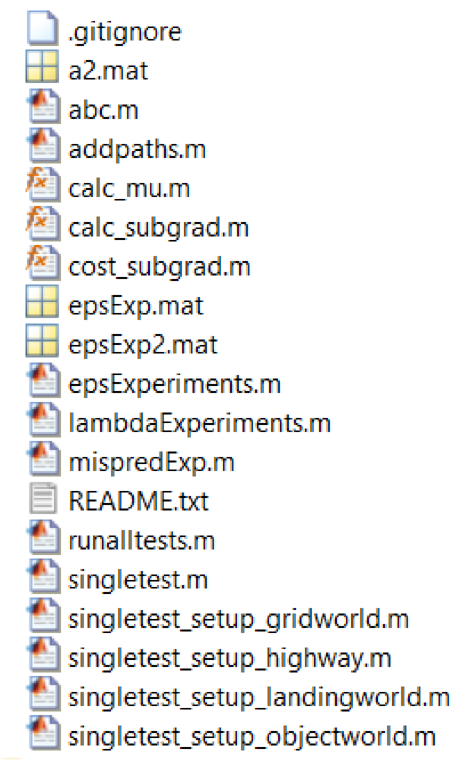
- *Evaluation*. This directory contains implementations of the various metrics that can be used to evaluate the IRL result. Each function is a particular metric.
- *General*. These are general testing scripts, including *runtest*, *runtransfertest*, etc.
- *HumanControl*. Interface for generating human demonstrations.
- *NIPS11_Tests*. Scripts for reproducing test results from "Nonlinear Inverse Reinforcement Learning with Gaussian Processes."
- *Testing*. Scripts for graphing, saving, and visualizing test results from series tests.
- *Utilities*. External utilities, including *minFunc* and *CVX*, where *cvx_setup* is to be executed.

7.3 Usage

After knowing an overview of the directories, it is possible to understand how to execute some example. It can be running a single test, a transfer test (to test how well a learned reward function generalizes to another state space), a series of test or series of transfer tests or using human data, which can be obtained for the three default examples in *Human_Demos*.

It has been implemented two extra files of single test, to make the execution of the examples *Gridworld* and *Highway* with their respective data, because the default one is expressed only on terms of *Objectworld*. Additionally, one single test for landing is available.

These and other files are shown in Figure 7.3.



```
.gitignore
a2.mat
abc.m
addpaths.m
calc_mu.m
calc_subgrad.m
cost_subgrad.m
epsExp.mat
epsExp2.mat
epsExperiments.m
lambdaExperiments.m
mispredExp.m
README.txt
runalltests.m
singletest.m
singletest_setup_gridworld.m
singletest_setup_highway.m
singletest_setup_landingworld.m
singletest_setup_objectworld.m
```

Figure 7.3 Extra files of IRL Toolkit directory.

7.4 Definition of the current problem

As it has been indicated, the extracted variables from the simulator are those from Section 5.3. However, not all of them are considered to define the state of the airplane:

- latitude, *lat*, in degrees
- longitude, *long*, in degrees
- altitude in meters measured from the mean sea level, *altmsl*, converted to feet
- altitude in meters measured from above ground level, *altagl*, converted to feet
- pitch ratio, *pitch_r*, which belongs to the maximum range of [-1.00,1.00], the minimum and maximum deflections of the yoke, controlling the elevator, where -1.00 is full down, and 1.00 is full up.
- roll ratio, *roll_r*, which belongs to the maximum range of [-1.00, 1.00], the minimum and maximum deflections of the yoke, controlling the ailerons, where -1.00 is full left, and 1.00 is full right.
- heading ratio, *head_r*, which belongs to the maximum range of [-1.00, 1.00], the minimum and maximum deflections of the yoke, controlling the rudder, where -1.00 is full left, and 1.00 is full right.

- throttle ratio, *T_r*, which is the position of the handle itself (ratio of all engines), from 0.0 (idle) to 1.0 (max normal).
- indicated airspeed, *V*, in knots

Flap deflection, gear status and autobrake aren't considered because with all these states, the definition is already complex and they don't change during the landing, being fixed before starting it at a distance of around 3nmi from the runway.

The actions that may be taken are related to increase, decrease any of these parameters or do nothing:

- *pitch_r*: ratio of pitch
- *roll_r*: ratio of roll
- *head_r*: ratio of heading
- *throttle_r*: ratio of throttle

The features employed to determine the reward in a more accurate way are:

- *pitch*, pitch of the aircraft in degrees
- *roll*, roll of the aircraft in degrees
- *head*, heading in degrees

They could be described as in [19], however due to lack of time and complexity of the codes, they haven't been implemented in the IRL Toolkit.

7.4.1 Airport data

The program is aimed to land without external aid. However, a data base must be on board to know some information about the runways in order to be able to calculate the optimal descending and touchdown.

airportdata.m Subsection 9.2.1 allows to edit the following information:

- information of the north extreme of the runway: name, latitude, longitude, elevation and the altitude (measured with mean sea level) of the north glideslope installation. This last one is needed when using the altitude MSL of the aircraft position (to avoid the interferences of buildings, hills and other things on the ground) before passing to the altitude AGL when reaching 15 ft of altitude (because the runway is considered of null variation due to its unknown profile)
- information of the south extreme of the runway:name, latitude, longitude, elevation and the altitude (measured with mean sea level) of the north glideslope installation
- width and length of the runway
- information about localizer: the relative distance (longitudinal and transversal) with the nearest extreme of the runway and the degrees of the interior and exterior cones and the increments of degree to define the limits of horizontal lanes (while more values, given as a vector, more lanes)
- information about glideslope: the relative distance (longitudinal and transversal) with the nearest extreme of the runway and the degrees of the interior and exterior cones and the increments of degree to define the limits of vertical lanes (while more values, given as a vector, more lanes)

In this case, the information about the runways 16R-34L, taken from X-Plane (it differs a bit regarding the reality), is in Table 7.1 and Table 7.2

Table 7.1 Information about 16R. Landing north $\rightarrow$ south.

	North extreme	**Glideslope (north)**	**Localizer (south)**
Latitude [degrees]	47.46384	47.46069	47.43774
Longitude [degrees]	-122.31785	-122.31683	-122.31809
Altitude [ft msl]	330.33	351.79	349.38

Table 7.2 Information about 34L. Landing south → north.

	South extreme	**Glideslope (south)**	**Localizer (north)**
Latitude [degrees]	47.44056	47.44304	47.46661
Longitude [degrees]	-122.31808	-122.31667	-122.31784
Altitude [ft msl]	353	359.86	308.4

7.4.2 Conversions

As it has been exposed, IRL Toolkit works with states: single values which represent the definition of the aircraft each instant of time. The easier way of setting up the state is to have all the variables that define it (vectors of values that depend on the instant of time) expressed like vectors of whole numbers starting from 1. These can be added up easily and allow to distinguish a state from another very intuitively with the inverse of the formula used to obtain it.

These vectors of whole numbers are going to be called coordinates, as they represent positions in matrices. Mainly, two different conversions are employed depending on the variable to deal with.

From degrees into coordinates

It is needed to convert from increment of degrees of latitude and longitude into feet, in order to represent distances which are much more easy to represent with plots in MATLAB. The associated program is the one in Subsection 9.3.2

It has to be said that two different conversion factors are searched, due to the decreasing distance between two meridians, that goes to zero when going to the poles, that varies the conversion factor of longitudinal degrees into feet depending on the area around where the wanted distance is.

At first, the method was using KSEA diagram of FAA in Figure 4.6. It was obtained a conversion factor of latitude into feet thanks to the division of the decimal degrees and the known measure of the runway.

$$1\ segment = 0.1' = 0.0017 \tag{7.1}$$

With the length of the runway of 8500 feet and 14 segments:

$$14 \cdot 0.0017 = 0.0233 \tag{7.2}$$

So the conversion could be:

$$8500\ feet/0.0233 = \mathbf{364290\ \ feet/degree} \tag{7.3}$$

However, it was rejected due to precision, above all with the conversion of longitudinal degrees into feet.

The other option proposed was taking the distance in the Equator that corresponds to 360 degrees and pass it to feet. The equatorial circumference consists on 40075161.2 meters: the Earth radius of the semi-major axis of the Earth at the equator is 6378137 meters, so it gives:

$$111319.9\ meters/degree \cdot 3.28084\ feet/meter = \mathbf{365222.78\ \ feet/degree} \tag{7.4}$$

The inconvenience of this one is the inaccuracy for the longitude conversion.

The solution offered is the Haversine formula. It calculates the great-circle distance between two points, the shortest distance over the Earth's surface.This is needed above all due to the decreasing distance between two meridians, that goes to zero when going to the poles, multiplied by the cosine of the latitude. The two previous conversions aren't used.

$$\begin{aligned} a &= (sin(\Delta\phi/2))^2 + cos(lat_1 \cdot \pi/180)cos(lat_2 \cdot \pi/180)(sin(\Delta\lambda/2))^2 \\ c &= 2 \cdot atan2(\sqrt{a}, \sqrt{1-a}) \\ d &= R_{Earth} \cdot c \end{aligned} \tag{7.5}$$

With Earth's mean radius $R_{Earth} = 6371 \cdot 10^3 meters$, passed to feet multiplying by 3.28084 feet/meters, and the increment in radians:

$$\begin{aligned} \Delta\phi &= (lat_2 - lat_1)\pi/180 \\ \Delta\lambda &= (long_2 - long_1)\pi/180 \end{aligned} \tag{7.6}$$

Applying (7.5) for (7.7) and (7.8) in degrees:

$$lat_2 = lat_1 + 1;\ long_2 = long_1 \tag{7.7}$$

$$lat_2 = lat_1;\ long_2 = long_1 + 1 \tag{7.8}$$

For degrees of latitude it is obtained *convlat* and for degrees of longitude it is obtained *convlong*. The respective results for the location of this project, the KSEA, are:

$$\begin{aligned} \boldsymbol{convlat} &= \mathbf{364812.76}\ \boldsymbol{feet/degree\ \ of\ \ latitude} \\ \boldsymbol{convlong} &= \mathbf{246631.93}\ \boldsymbol{feet/degree\ \ of\ \ longitude} \end{aligned} \tag{7.9}$$

Multiplying the degrees by the corresponding factors, the distances obtained are transformed into coordinates, which represent lanes. In the axis system defined in this project, the variable that shows the advance of time is the one defined in the direction of the runway, x (starting at 1 in the initial location of the aircraft), and the transversal to this one in the horizontal plane is used to create the horizontal lanes, as well as the one in the vertical plane, that defines the vertical lanes.

While the limits of the vertical lanes are established by lines generated from a glideslope installation (supposed in its optimal position if it doesn't exist in the runway), the limits of the horizontal ones are determined by lines generated from a localizer at the further extreme of the runway.

Ratios to coordinates

Applying a similar reasoning to the rest of parameters that define the state is obtained the code of Subsection 9.3.1.

It is important to note that *pitch_r*, *roll_r*, *head_r* and *T_r* are ratios, and therefore, a conversion to the "components of the possible number of each parameter is needed". For example, *pitchrs* could be 201 possible values, from -1 to 1 with a step of 0.01. In this case it is going to be applied a linear conversion, so its values have to be found. Before that, as the data extracted comes with several decimals from X-Plane, they must be discretised to adapt them to the possible values given by the indicated range (*ini* and *last*) and the number of possible values, *divisions*. Hence, the step is defined:

$$step = \frac{last - ini}{divisions - 1} \tag{7.10}$$

Associating each value from X-Plane to the possible discretised ones is like rounding to the nearest possible value.

Having that, the values of the linear conversion can be deduced proposing two equations, taking the first and the last possible values of pitch ratio, are the (7.11):

$$\begin{aligned} -1a + b &= 1 \\ 1a + b &= 201 \end{aligned} \tag{7.11}$$

Subtracting the first to the second one:

$$2a = 200 \rightarrow a = 100 \tag{7.12}$$

Substituting in one of (7.11)

$$b = 101 \tag{7.13}$$

Then, if *pitch_r* is 0.1, its conversion would be:

$$0.1 \cdot 100 + 101 = 112 \tag{7.14}$$

It is going to be considered the same number of possible ratios of all variables: $pitchrs = rollrs = headrs = Trs = divisions$. Generalizing for an unknown number of possible pitch ratios (*pitchrs*) and for a range [*ini,last*]:

$$\begin{aligned} ini \cdot a + b &= 1 \\ last \cdot a + b &= pitchrs \end{aligned} \tag{7.15}$$

Subtracting the first to the second one:

$$(last - ini)a = pitchrs - 1 \rightarrow a = \frac{pitchrs - 1}{last - ini} \tag{7.16}$$

Substituting in the first equation of (7.15)

$$b = 1 - a \cdot ini \rightarrow b = 1 - \frac{pitchrs - 1}{last - ini} \cdot ini \rightarrow b = \frac{last - ini \cdot pitchrs}{last - ini} \tag{7.17}$$

With that, the coordinates of the ratios are:

$$\begin{aligned} pitch\_r &= pitchr \cdot a + b \\ roll\_r &= rollr \cdot a + b \\ head\_r &= headr \cdot a + b \\ T\_r &= Tr \cdot a + b \end{aligned} \tag{7.18}$$

Regarding the indicated airspeed, its coordinates are calculated as a combination of the two previous methods. It is given the initial and last values of the range of the speed and the number of divisions (for this project, it has been considered the same as for the ratios, although it could be simply added as another parameter). The difference with the ratios is that the velocity is not discretised. Instead, it is directly converted into coordinates, checking if each component is inside two near possible vales and associating the respective coordinate.

These three methods can be edited or improved, modifying the codes and always having as outputs the coordinates to compute the state.

The inverse procedure is implemented in Subsection 9.4.3

7.4.3 landing

In this program (Section 9.2) a vertical and a horizontal view of the movement of the aircraft is represented accordingly to the glideslope and the localizer in the selected runway. With this, it is possible to know of a simple and quick way if the data taken from X-Plane is correct regarding the situation of the aircraft each step of time, before its conversion and implementation in IRL Toolkit.

A first section of data that can be changed allows to:

- load the data from a .mat file (for example from the added path: XPC original data copied folder)
- call to *airportdata.m* where the runway, localizer and glideslope data is found and can be edited
- introduce conversion factors to pass from latitude and longitude to feet, as it has been explained in Subsubsection 7.4.2

With that, the (x,y,z) of the runway and aircraft are obtained and all is calculated taking as origin point of the graphic representation the furthest extreme of the runway from the position of the aircraft.

It has to be said that to reproduce the graphics of X-Plane as best as possible, the landing will be represented from the right to the left, independently of the direction (north to south or south to north). This generalisation will also be useful to draw the landing environment in the Toolkit. To achieve that, some parameters have to be changed of sign depending on the direction and others even their value, such as the altitude of the glideslope related to the point of touchdown. For the altitude of the aircraft, because of unknowing the profile of the runway, it has been defined as a combination of the subtraction of the altitude measured from the mean sea level and the altitude of the corresponding point of glideslope (until the aircraft is 15 feet above ground), and the altitude above ground level when it is in the runway.

Then, on the one hand, the vertical plane includes the glideslope cone projected with two different zones: a more centred disposition around 3 degrees and the vertical movement of the aircraft.

On the other hand, the localizer cone with two areas is projected in a horizontal plane with the horizontal movement of the aircraft.

In both, which can be appreciated with an example in Figure 7.4, the following aspects can be found:

- runway views, in black and center axis in discontinuous line
- aircraft movement in red
- a description in a title indicating the view, name of the instrument related, name of the runway and the direction of landing.
- the labels of the axis and their units
- the location of glideslope, in dark blue, and localizer, in green
- aircraft movement in red
- the limits of the range, where it is recommended to fly in order to perform a soft touchdown, in the color corresponding to the localizer or glideslope.

The range of movement is fixed by Figure 7.5 of [8], because of not having very accurate measures from Figure 7.6. In spite of that, comparing the images, their similarity can be appreciated.

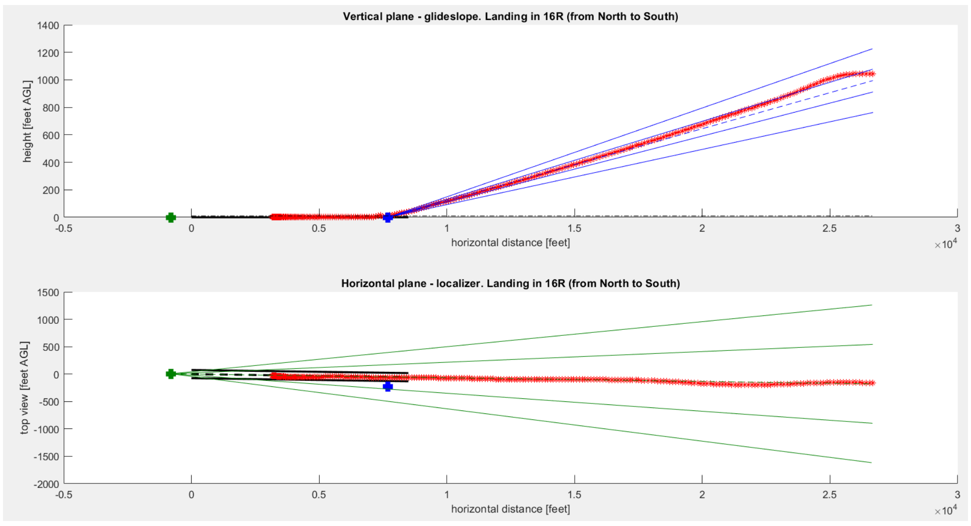


Figure 7.4 Vertical and horizontal views.

7.5 XPC IRL converter

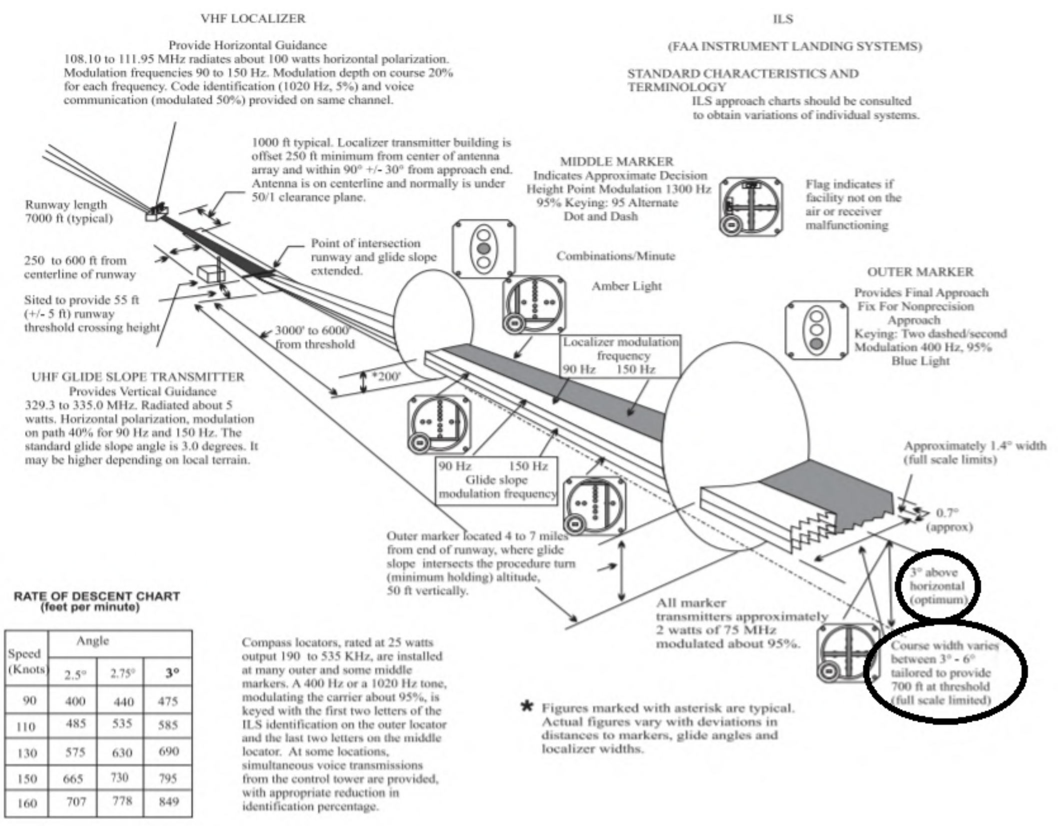
It is a program (Section 9.3) to convert data taken from X-Plane with X-Plane Connect (XPC) Toolbox into the kind of data managed by the IRL Toolkit. It produces files of .mat extension which contain a structure, *autopilot_result*, with two fields:

- *mdp_params*, a structure with the common data of every iteration
- *example_samples*, with a cell matrix that contains in each cell a state and an action, defined by:
  - columns with the number of steps considered (it doesn't have to be a complete iteration. It could be divided into several segments, each one in a .mat file.
  - rows with the different iterations (they have to have the same characteristics)

Before running the script, the files generated by XPC must be copied to the folders:

- *landing_demos*, which is going to be modified
- *XPC original data copied*, for doing tests and don't loose the data. If something goes wrong, the data in *landing_demos* can be browsed and restored from this folder or from the folder of XPC.

After executing the program, *landing_i.mat* files are generated with the segments of a set of iterations.



Runway length
7000 ft (typical)
Point of intersection runway and glide slope extended.
Combinations/Minute
Amber Light
Localizer modulation frequency
90 Hz 150 Hz
3000' to 6000' from threshold
*200'
90 Hz 150 Hz
Glide slope modulation frequency
Approximately 1.4° width (full scale limits)
0.7° (approx)
3° above horizontal (optimum)
Course width varies between 3° - 6° tailored to provide 700 ft at threshold (full scale limited)

VHF LOCALIZER

Provide Horizontal Guidance
108.10 to 111.95 MHz radiates about 100 watts horizontal polarization. Modulation frequencies 90 to 150 Hz. Modulation depth on course 20% for each frequency. Code identification (1020 Hz, 5%) and voice communication (modulated 50%) provided on same channel.

ILS

(FAA INSTRUMENT LANDING SYSTEMS)

STANDARD CHARACTERISTICS AND TERMINOLOGY
ILS approach charts should be consulted to obtain variations of individual systems.

1000 ft typical. Localizer transmitter building is offset 250 ft minimum from center of antenna array and within 90° +/- 30° from approach end. Antenna is on centerline and normally is under 50/1 clearance plane.

MIDDLE MARKER
Indicates Approximate Decision Height Point Modulation 1300 Hz 95% Keying: 95 Alternate Dot and Dash

Flag indicates if facility not on the air or receiver malfunctioning

250 to 600 ft from centerline of runway

Sited to provide 55 ft (+/- 5 ft) runway threshold crossing height

OUTER MARKER
Provides Final Approach Fix For Nonprecision Approach
Keying: Two dashed/second
Modulation 400 Hz, 95%
Blue Light

UHF GLIDE SLOPE TRANSMITTER
Provides Vertical Guidance
329.3 to 335.0 MHz. Radiated about 5 watts. Horizontal polarization, modulation on path 40% for 90 Hz and 150 Hz. The standard glide slope angle is 3.0 degrees. It may be higher depending on local terrain.

Outer marker located 4 to 7 miles from end of runway, where glide slope intersects the procedure turn (minimum holding) altitude, 50 ft vertically.

All marker transmitters approximately 2 watts of 75 MHz modulated about 95%.

RATE OF DESCENT CHART
(feet per minute)

Speed (Knots)	Angle		
	2.5°	2.75°	3°
90	400	440	475
110	485	535	585
130	575	630	690
150	665	730	795
160	707	778	849

Compass locators, rated at 25 watts output 190 to 535 KHz, are installed at many outer and some middle markers. A 400 Hz or a 1020 Hz tone, modulating the carrier about 95%, is keyed with the first two letters of the ILS identification on the outer locator and the last two letters on the middle locator. At some locations, simultaneous voice transmissions from the control tower are provided, with appropriate reduction in identification percentage.

* Figures marked with asterisk are typical. Actual figures vary with deviations in distances to markers, glide angles and localizer widths.

Figure 7.5 ILS with localizer and glideslope according to [8].

7.5.1 Options

Here are presented all the data that is susceptible to being changed, as values of parameters.

File configuration

It is started with file configuration:

- *tlen*, total length to standardise the iterations
- *len*, length of the segment. It is a parameter to have the landing discretised in segments of the same length (for example 50 instants of time, in which the connection has been successful), because in each iteration, the number of taken data is different. When it is in the last segment, every landing is different, and to solve that, those with a minor number of components are filled repeating their last value. The number of instants of time of a landing performed with 0.2 s as step time from 3 nmi of the runway oscillates between 255 and 281. If the discretised length is 50, taking a total extension of *tlen*, as 300 instants, it would result in 6 segments and their corresponding files.
- *ini*, initial point of the range of pitch, roll, heading and throttle ratios. It has been considered the same for all of them.
- *last*, last point of the range of pitch, roll, heading and throttle ratios. It has been considered the same for all of them.
- *divisions*, number of possible values in the range of pitch, roll, heading and throttle ratios. It has been considered the same for all of them. For example, 201 divisions belongs to a range [-1,1] and step of 0.01.
- *inis*, initial value of the range of indicated airspeed
- *lasts*, last value of the range of indicated airspeed

Airport information

Then the airport information is offered. If the instrument landing systems doesn't exist, their optimal coordinates should be set to generate the optimal cones of descending.

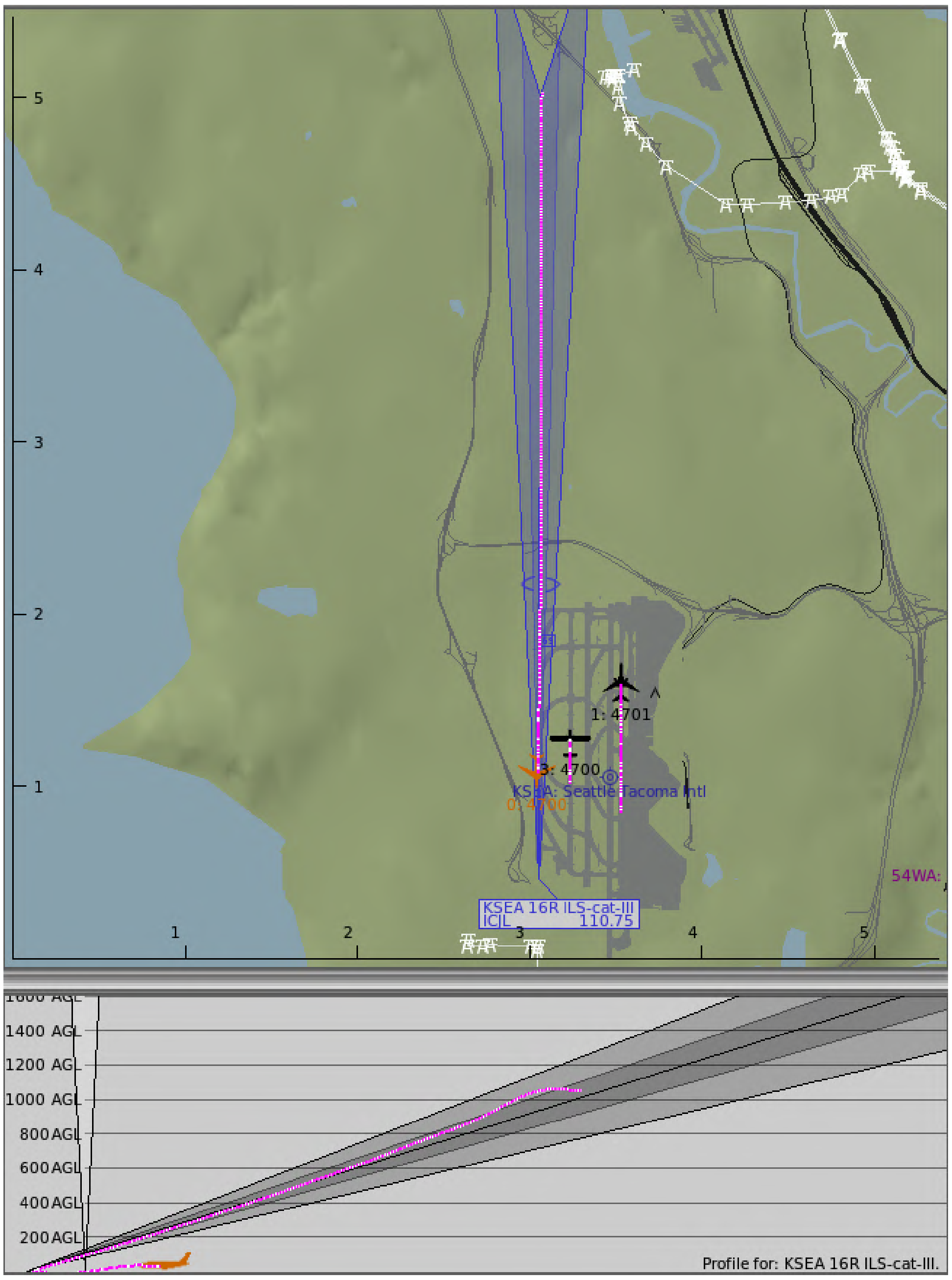


Figure 7.6 XPlane representation in Local Map of an aircraft landing.

It calls a file named *airportdata.m*, the one in Subsection 9.2.1, and explained in Subsection 7.4.1. In it the horizontal and vertical lanes (*Hlanes* and *Vlanes*) are calculated as regards to the number of values of increments of degrees that define the limits of the lanes.

IRL algorithm information

At last, some fixed information is needed relative to the IRL algorithm.

- *continuous*, if it is 1 the features are continuous, if not, they are discrete
- *determinism*, probability of correct transition
- *discount*, temporal discount factor to use

For deeper modifications, all the code should be completely read and understood before changing anything.

7.5.2 Processing

The folder *landing_demos* where the .mat files are located is added to path (it is needed because, in general, MATLAB only allows to work with files that are in the same folder that the executed file), and then a structure is obtained with information of all the .mat files of that folder (*landing_demos*).

A first loop is started to repeat the same in each iteration. There is a problem when the connection fails, the vertical velocity increases in the right next iteration after the bad connection(s). So, to solve this, unusual values are searched and substituted in a internal loop with the media of the previous (it has been considered 0 if it was the first component or the last one) and the next one.

Then, to get the files of the same length, the last value of each parameter is repeated to fill the number of components indicated by *tlen*.

The last thing in the first loop is to discretise the values. That is, to associate the extracted values of X-Plane (by default with several decimals) with the possible values given by the indicated range (*ini* and *last*) and *divisions* as well as converting all of the relevant data for the state into coordinates, with what has been explained in Subsection 7.4.2.

7.5.3 First structure, *mdp_params*

Once the data is adequate to work with, it is going to be adapted to the IRL Toolkit. The first structure needed is really simple, as it doesn't need any extra operations, and it doesn't depend on X-Plane. It consists on this information:

- *seed*, initialization for random seed
- *ini*, initial point of the range of pitch, roll, heading and throttle ratios. It has been considered the same for all of them.
- *last*, last point of the range of pitch, roll, heading and throttle ratios. It has been considered the same for all of them.
- *inis*, initial point oof the range of indicated airspeed.
- *lasts*, last point of the range of indicated airspeed.
- *length*, length of the segment.
- *Hlanes*, number of possible horizontal lanes (associated to longitude)
- *Vlanes*, number of possible horizontal lanes (associated to altitude)
- *pitchrs*, number of possible pitch ratios. It coincides with *divisions*.
- *rollrs*, number of possible roll ratios. It coincides with *divisions*
- *headrs*, number of possible heading ratios. It coincides with *divisions*
- *Trs*, number of possible throttle ratios. It coincides with *divisions*
- *airport*, a structure with all the information of Subsubsection 7.5.1
- *continuous*, parameter to set a discrete (0) or continuous (1) problem
- *determinism*, probability of correct transition
- *discount*, temporal discount factor to use

7.5.4 Second structure, *example_sample*

This one requires some extra manipulation. Three loops are contemplated. On the external one, the files that are going to be generated are created, and the structure, initialized as a cell matrix of zeros.

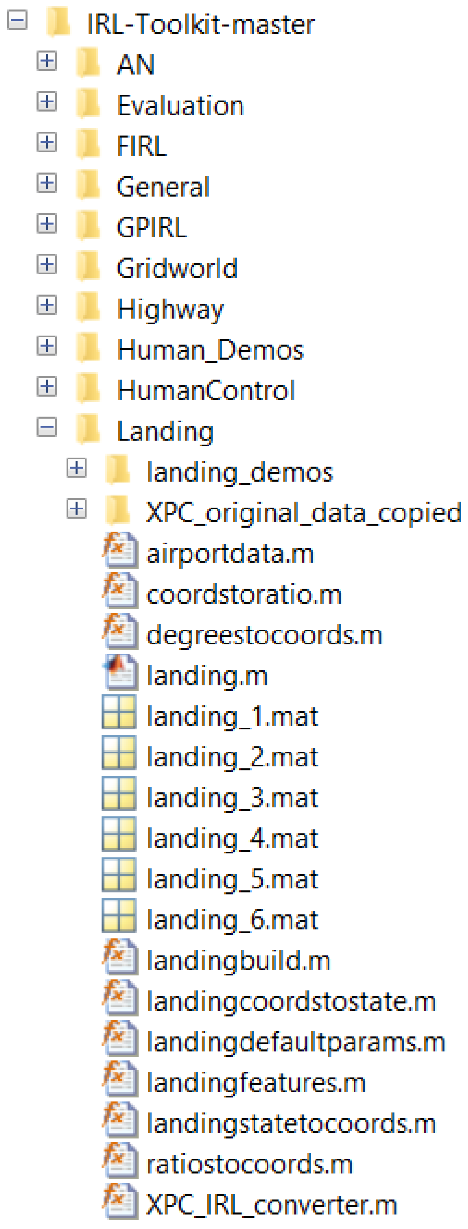
In the second loop, for each of the iterations, its data is loaded, and the vectors of pitch, roll, heading and throttle ratios are differentiated. To know the action to perform, the maximum difference indicates the variable to change.

The third loop is used to stablish the state and the action of increasing or decreasing the corresponding parameter for each instant of time in the segment. The stated is obtained as the result of an external function (*landingcoordstostate*, explained in Subsection 7.6.1) after passing to it the longitudinal position, the horizontal and vertical lanes; the pitch, roll, heading and throttle ratios; and the *mdp_params*, where the important parameters are the number of possible: horizontal and vertical lanes; pitch, roll, heading and throttle ratios; and indicated airspeed.

7.6 Own codes in IRL Toolkit

Just like the three world by default, *Gridworld*, *Objectworld* and *Highway*, were defined by several functions, another world has been created to satisfy the requirements of the project: *Landing*.

This section includes the explanation about the new environment developed, which is shown in Figure 7.7.



```
IRL-Toolkit-master
    AN
    Evaluation
    FIRL
    General
    GPIRL
    Gridworld
    Highway
    Human_Demos
    HumanControl
    Landing
        landing_demos
        XPC_original_data_copied
        airportdata.m
        coordstoratio.m
        degreestocoords.m
        landing.m
        landing_1.mat
        landing_2.mat
        landing_3.mat
        landing_4.mat
        landing_5.mat
        landing_6.mat
        landingbuild.m
        landingcoordstostate.m
        landingdefaultparams.m
        landingfeatures.m
        landingstatetocoords.m
        ratiostocoords.m
        XPC_IRL_converter.m
```

Figure 7.7 Landing directory.

7.6.1 landingcoordstostate

This function, which is presented in Subsection 9.3.3, converts the coordinates that define a point into the state that the algorithm needs to apply IRL.

For that, the following procedure gives the equations that are programmed there.

To make it simple, an example of a car in a highway is applied. The car is supposed to move forwards inside three lanes with four possible velocities. So in the first position, in the first lane, with the first velocity, it would be the first state. If it were in the same longitudinal position and lane, but with the next speed, it would represent the second state, and so forth, as it is in Figure 7.8. This situation, where the car has to avoid other vehicles and move under certain restrictions, is the mentioned *highway* example of the Toolkit and it is fully developed in [28].

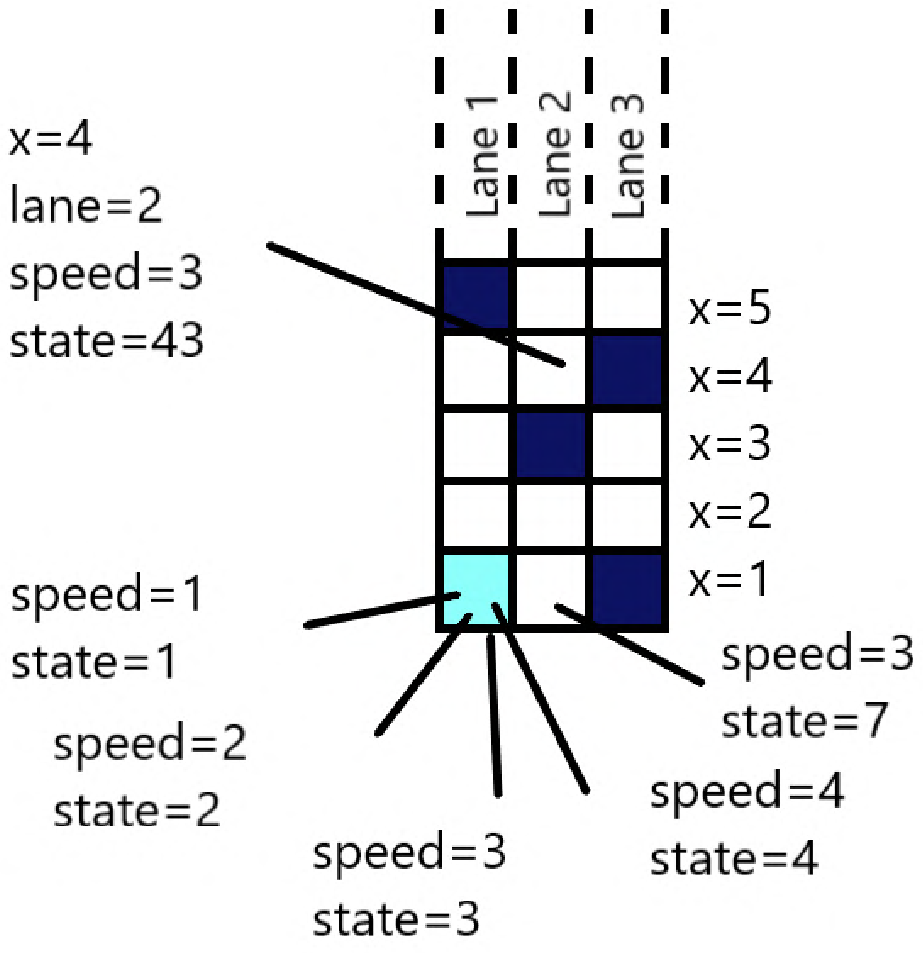


Figure 7.8 States in a highway.

Three variables are involved: longitudinal position, x; transversal position, *lane*; and *speed*. Forming a matrix with the number of lanes, nl, as columns and the number of files as the possible longitudinal positions, n, Figure 7.8 reflects that in each position, some number of speeds ns, are available (four in this case), defining the state. In a certain point, x, in a certain *lane* with a certain *speed*, its state, s, would be the sum of the previous positions multiplied for the number of possible speeds (four in this case) and adding up the speed that maintains in the moment.

$$s = ((x-1) \cdot nl + (lane-1)) \cdot ns + speed \tag{7.19}$$

Reordering for a future application in Subsection 7.6.2:

$$s = (x-1) \cdot nl \cdot ns + (lane-1) \cdot ns + speed \tag{7.20}$$

Therefore, from Figure 7.8, with the point defined by:

- $x = 4$
- $lane = 2$
- $speed = 3$
- $nl = 3$
- $ns = 5$

Its state as $s = 43$ is explained as:

$$s = (4-1)\cdot 3\cdot 4 + (2-1)\cdot 4 + 3 = 43 \tag{7.21}$$

$$\begin{aligned} s &= (x-1)\cdot Hlanes\cdot Vlanes\cdot pitchrs\cdot rollrs\cdot headrs\cdot Trs\cdot Vs \\ + & \quad (Hlane-1)\cdot Vlanes\cdot pitchrs\cdot rollrs\cdot headrs\cdot Trs\cdot Vs + \\ + & \quad (Vlane-1)\cdot pitchrs\cdot rollrs\cdot headrs\cdot Trs\cdot Vs + \\ + & \quad (pitch\_r-1)\cdot rollrs\cdot headrs\cdot Trs\cdot Vs + \\ + & \quad (roll\_r-1)\cdot headrs\cdot Trs\cdot Vs + \\ + & \quad (head\_r-1)\cdot Trs\cdot Vs + \\ + & \quad (T\_r-1)\cdot Vs + \\ + & \quad V\_ \end{aligned} \tag{7.22}$$

It must be said that *pitchrs*, *rollrs*, *headrs*, *Trs* and *Vs* are fields in a structure named *mdp_params*, and to access them, dot notation has to be used of the form *structName.fieldName* (*mdp_params.pitchrs* for instance)

Vlanev and *Hlanev* are vectors indicating the vertical and horizontal lane in which the aircraft is flying. In Figure 7.9, an example with 5 *Hlanes* and 5 *Vlanes* (indicating two values to *incrthetaL* and two values to *incrthetaG*) are shown as an equivalent to Figure 7.8. There the different lanes are indicated with their number in a circle while the limits determined by the lines are associated to a number inside a square.

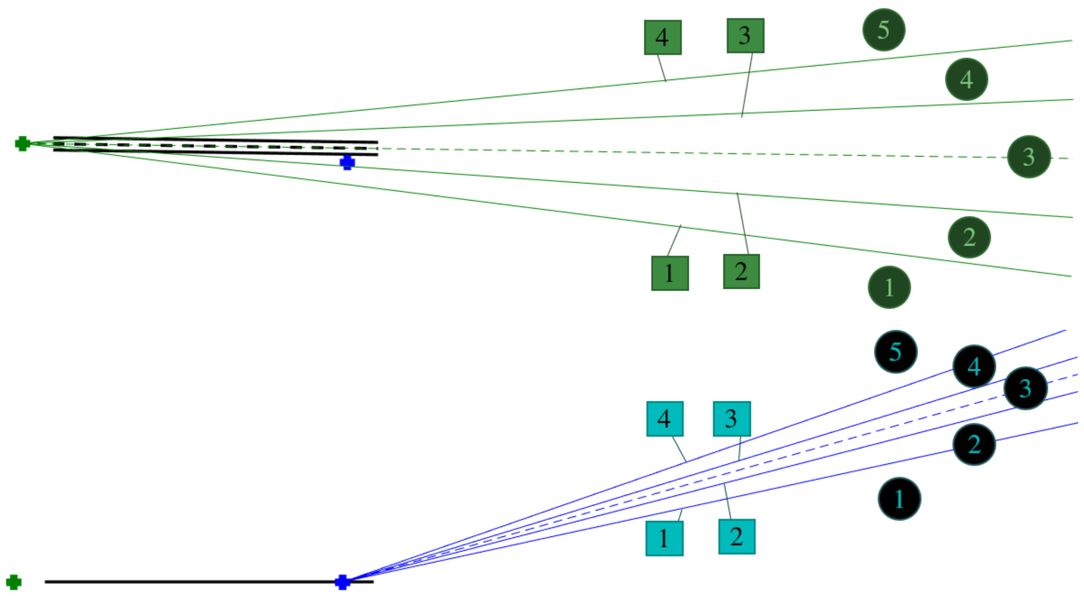


Figure 7.9 Horizontal and vertical lanes.

7.6.2 landingstatetocoords

This function, which is presented in Subsection 9.4.2, applies the inverse concept of landingcoordstostate, transforming the state into coordinates that define the characteristics of the aircraft in a more intuitive way.

For that, the following procedure provides the equations that are programmed there.

Continuing with the *highway* example, given the state, the coordinates are obtained dividing it by the highest coefficients of its definition in (7.20) and updating s subtracting that term for each parameter. It is shown the final transformation in (7.24).

$$\begin{aligned}
x &= ceil(s/(nl \cdot ns)) \\
s &= s - (x-1) \cdot (nl \cdot ns) \\
lane &= ceil(s/ns) \\
s &= s - (lane - 1) \cdot ns \\
speed &= s
\end{aligned} \tag{7.23}$$

Extrapolating to the current problem:

$$\begin{aligned}
x &= ceil(s/(Hlanes \cdot Vlanes \cdot pitchrs \cdot rollrs \cdot headrs \cdot Trs \cdot Vs)) \\
s &= s - (x-1) \cdot Hlanes \cdot Vlanes \cdot pitchrs \cdot rollrs \cdot headrs \cdot Trs \cdot Vs \\
Hlane &= ceil(s/(Vlanes \cdot pitchrs \cdot rollrs \cdot headrs \cdot Trs \cdot Vs)) \\
s &= s - (Hlane-1) \cdot Vlanes \cdot pitchrs \cdot rollrs \cdot headrs \cdot Trs \cdot Vs \\
Vlane &= ceil(s/(pitchrs \cdot rollrs \cdot headrs \cdot Trs \cdot Vs)) \\
s &= s - (Vlane-1) \cdot pitchrs \cdot rollrs \cdot headrs \cdot Trs \cdot Vs \\
pitch\_r &= ceil(s/(rollrs \cdot headrs \cdot Trs \cdot Vs)) \\
s &= s - (pitch\_r-1) \cdot rollrs \cdot headrs \cdot Trs \cdot Vs \\
roll\_r &= ceil(s/(headrs \cdot Trs \cdot Vs)) \\
s &= s - (roll\_r-1) \cdot headrs \cdot Trs \cdot Vs \\
head\_r &= ceil(s/(Trs \cdot Vs)) \\
s &= s - (head\_r-1) \cdot Trs \cdot Vs \\
T\_r &= ceil(s/(Vs)) \\
s &= s - (T\_r-1) \cdot Vs \\
V\_ &= s
\end{aligned} \tag{7.24}$$

As before, *pitchrs*, *rollrs*, *headrs*, *Trs* and *Vs* are fields in a structure named *mdp_params*, and to access them, dot notation has to be used of the form *structName.fieldName* (*mdp_params.pitchrs* for instance)

7.6.3 coordstoratios

This function in Subsection 9.4.3 is needed to convert the coordinates (defined by the state and action indicated by the IRL algorithm) to ratios that will be introduced in X-Plane. It is the inverse procedure of Subsection 7.4.2.

7.6.4 landingdefaultparams

This file Subsection 9.4.4 contains the default value of the default parameters. If a field is given a value by the user, it doesn't have to be filled with the default one.

7.6.5 Unfinished functions

Several functions haven't been finished. Their names and the explanation of why they aren't ready to be executed are:

- *landingbuild.m*, where the environment is defined. A single argument is needed as input, the *mdp_-params* structure passed to *runtest* that specifies the parameters of the desired example, and which is already defined in this case. Its outputs are:
  - *mdp_data*, a structure containing the definition of the MDP. It must specify the following fields, all of them already defined:
    - *states*, the number of states.
    - *actions*, the number of actions in each state.
    - *discount*, the discount factor of the MDP (currently all examples are infinite horizon discounted reward).
    - *sa_s*, 3D matrix of size *states* x *actions* x *K*, where *K* is any number. The entry *sa_s(s,a,k)* specifies the state to transition to when taking action a in state *s*, and sampling destination *k* (see below).

        - *sa_p*, 3D matrix as above, where *sa_p(s,a,k)* gives the probability of transitioning to *sa_-s(s,a,k)* on action a in state s.
    - *r*, a *states* x *actions* matrix specifying the true reward function of the example environment, the **unique parameter** of *landingbuild.m* that hasn't been defined. It can be created in this file or in *landingfeatures.m*.
    - *feature_data*, a structure containing two fields regarding the features of the example, from *landingfeatures.m*.
- *landingfeatures.m*, with the features associated to the problem. From its two (three - depending on the example) outputs, only a part of the first one is finished, **the others have to be added**:
    - *feature_data* with two fields:
        - *stateadjacency*, the only one which is the same as in the default examples, a sparse *states* x *states* matrix with "1" for each pair of states that are "adjacent" (have an action to transition from one to the other); although it is only used in FIRL, what has been said in Subsubsection 7.2.3.
        - *splittable*, a *states* x *features* matrix containing the value of each feature at each state. Despite the fact of having them defined as in [19], the difficult operation is expressing them in that format. This may also cause the addition of pitch, roll and heading (*pitch*, *roll* and *head*) to the parameters that define the state, which would become to depend on 11 variables.
    - *true_feature_map*, a (sparse) *states* x *features* matrix that has to be extrapolated from one of the default examples to the current case. It gives the values of the "true" features that can be linearly combined to obtain *r*.
    - *r*, the reward function, only if it isn't generated inside *landingbuild.m*.

7.7 Continuation

The ten iterations performed with the simulator X-Plane have been processed and converted to six .mat files with *XPC_IRL_converter.m*, ready to be inputs to IRL Toolkit.

Nevertheless, the training phase has presented unexpected problems which couldn't be solved in time. Mainly, as it has been announced previously, because of a subestimated development of the project before starting it. With the proximity of the deadline, it has been preferred closing the previous aspects of IRL Toolkit applying a logical and sequential order. This avoids omissions or unfinished intermediate parts whose lacking pieces would be much more difficult to fit together.

A specific description of what is still needed to develop is the following:

- finishing the previous functions of Subsection 7.6.5
- revising the files of *NIPS11_Tests* folder and adapting them to the problem of landing. That is to be able to running the series of tests with all the files and learn the reward and policy functions instead of running a single file with *singletest_setup_landingworld.m* of Figure 7.3. The function to develop could be named as *landingtest*. The created world differs from *highway* in that its examples can't be generated in IRL Toolkit what must be taken into account while calling to other functions in the toolbox.
- creating a function to use the results obtained after a first analysis of the ten iterations in IRL Toolkit. It may be finishing *SERP3nmiappML.m* in *XPlaneConnect-master* folder, sending the correct action to perform in X-Plane each instant of time, evaluating the current state in the policy and reward function generated by IRL Toolkit.
    - policy π that maps states to actions
    - reward function R, which maps each state to a real valued reward $S \to \mathbb{R}$
- repeating the process if robust results want to be achieved, with a bigger set of iterations that must be performed using X-Plane through *SERP3nmiapp.m* and whose data has to be processed with IRL Toolkit, generating better reward and policy functions.
- reaching the pertinent conclusions:

- taking as reference the autopilot landings and those performed by the developed software which must have similar patrons and justifying it
- comparing the airplane response between both set of iterations and noticing if with a broader data spectrum, it presents some improvements

The degree of development of the different items performed in order to obtain a final application of automatic landing integrable with X-Plane is in Table 7.3, expressed as percentage.

Table 7.3 Percentage of development of the different items in the project.

Item	**Degree of development [%]**
X-Plane	
downloading X-Plane, XPC and installing *xpcplugin*	100
SERP3nmiapp	100
performing several landings	100
IRL Toolkit	
XPC_IRL_converter	100
landings from X-Plane converted to IRL Toolkit format	100
airportdata	100
coordstoratios	100
ratiostocoords	100
degreestocoords	100
landing (drawing from X-Plane)	100
landingcoordstostate	100
landingstatetocoords	100
landingdefaultparams	100
landingbuild	25
landinfeatures	25
landingtest	0
SERP3nmiappML	75

8 Considerations and future perspective

Due to factors such as time and complexity (above all of the codes in the toolkit used to develop the IRL algorithm, which have lead to finally develop only a tool to automatize the data obtaining of X-Plane and converting them to the appropriate format of IRL Toolkit), some aspects have been simplified or not taken into account. Here some of them are exposed and future or possible solutions are presented.

The landings have been performed one after other taking the plane to the 3 nmi approach point, sending it from there to the starting point indicated by hand in MATLAB (as it has been explained in Subsection 5.4.4), and running the program. Each iteration has lasts on average around a minute. In future applications, a more complex code could make all the steps done by hand in X-Plane and integrate them in a loop to which indicate the number of landings desired. Besides the exploration of longer paths would be interesting, engaging the localizer and the approach mode when a certain point is reached and meanwhile, finding a correct way of maintaining an optimal flight.

As regards the stability, it isn't taken into account. The autopilot is supposed to stabilize the airplane. To assure the trim of the aircraft when the algorithm had the control, it would only be compared to the values provided by the autopilot and observed if they are similar. With the objective of representing a more realistic vision, the stability derivatives should be extracted and evaluated.

Regarding the Landing in IRL Toolkit, in order to generalize the codes, it could be developed a version in which the range and step of *pitch_r*, *roll_r*, *head_r*, *T_r* and *V_* vary for each parameter. This would implied less number of possible values and therefore, less number of possible states if the range is reduced to a more specific environment of values that are reached when performing a landing.

In *SERPnmi3app.m* if the distance to the runway were bigger, the landing gear would not be deployed since the start of the simulation and another variable to control would be when to extract the landing gear, select autobrake or extract more the flaps.

Another improvement would be generalising *XPC_IRL_converter.m* to perform the extension of vectors with a loop instead of doing with every field, being able to adapt to all number of fields. So if a field is added/eliminated, the code doesn't need to be changed, adding a line for the new field. Field Properties must be excluded (it only is created by default with command matlabfile, with the properties of the .mat file).

The profile of the runway could be obtained in order to use altitude MSL and get more accuracy changing the definition of the altitude in *landing.m*.

An important point of view in IRL Toolkit would be taking into account several actions at the same time, either with an index or changing the algorithm code, much more complicated than the simplification of one action related to a parameter each instant of time.

In a future work, this project could be resumed. After fulfilling all the points of Section 7.7, and processing the results, the relevance that IRL has in the field of aeronautics could be discussed in the corresponding conclusions.

9 Own codes

The codes that have been created by the author of this project are presented in this chapter. These and the rest of codes that are necessary in order to replicate the results obtained are available in the attached documents and they have been developed to be self-contained.

9.1 XPC Toolbox

In this section, the codes that are needed in a complementary way to the XPC Toolbox with the objective of extracting and manipulate data from X-Plane are found.

9.1.1 SERP3nmiapp.m

Send starting point, Extract data, Record data and Process data taken beginning at points near to 3 nautical miles from the selected runway performing an approach.

Code 9.1 SERP3nmiapp.m.

```
function SERP3nmiapp
%% X-Plane Connect MATLAB SERP3nmiapp
% Before running this script, ensure that the XPC plugin is installed in
% X-Plane, X-Plane is running with the situation 3 nmi, preferred, paused
% since the beginning (letter 'P' must be pulsed when being at the runaway
% or during the loading of the approaching.
% WAIT SEVERAL SECONDS BEFORE RUNNING THE FILE
% It is correct if the plane starts going down (a little)
% The first times, it usually goes wrong
% Send (not very far from the localizer)
% Extract
% Record
% KSEA airport, 34L-16R runway, cat III, frequency 110.75 Hz, 3nmi

%% To run XPlane from MATLAB, uncomment this two lines before the script
%cd 'C:\Users\loren\Google Drive\X-Plane 10' %The path where X-Plane.exe is
%!X-Plane.exe & % Executes X-Plane


%% Import XPC
addpath('../')
import XPlaneConnect.*

%% Setup
%Info for .txt
```

```
plane      = 'A380';
airport    = 'KSEA';
weather    = 'broken';
time       = 'sunset';
distance   = '3nmi';
land_way   = '34L';    %16R 34L
start_point = 'sent';  %default sent
direction  = 'SN';

% Nomenclature:
% -Weather options:clear,few,scattered,broken,overcast,low-vis,foggy,stormy
% -Time options:day,sunset,twilight,night (it doesn't affect)
% -Distance: starting point around that distance to the airport
% -Landing runway: in 16R or in 34L
% -Starting point: from the default point ('default') or a sent point ('s')
% -Direction: the direction of the landing from south to north ('SN') or
%  from north to south ('NS')

% Files to save the data in
persistent num %to generate different .txt and .mat files
if isempty(num)
   num = 0;
end
num  = num+1;
path = [direction,weather,distance,land_way,start_point,num2str(num)];
fd   = fopen([path,'.txt'], 'w');      % Open .txt file
m    = matfile([path,'.mat'], 'w', true); % Open .mat file

% Characteristics of the connection
Socket   = openUDP(); % Open connection to X-Plane
interval = 0.2; %5 Hz between one data and another
s_spd    = 1; %multiplier for real-time (1: real,2: 2 x real, 0: pause)

%% Set position
%THIS SECTION CAN BE COMMENTED TO USE THE APPROACH POINTS BY DEFAULT
%To send a starting ponint, the following two lines starting with POSI and
%sendPOSI must be uncommented.
%      Lat     Lon        Alt      Pitch    Roll    Heading   Gear
POSI = [47.39, -122.32,  1385/3.28084, -998,  -998,  -998,  -998];
sendPOSI(POSI, 0, Socket); % Set own aircraft position
% If -998 is sent, the parameter will stay at the current X-Plane value

%By default 16R 3nmi:
% POSI = [47.514, -122.317, 1395.7/3.28084, -998, -998, -998, -998];

%By default 34L 3nmi:
% POSI = [47.391, -122.318, 1395.7/3.28084, -998, -998, -998, -998];

%CHECKED:
% POSI = [47.39, -122.32,  1385/3.28084, -998,  -998,  -998,  -998];
% POSI = [47.514, -122.32, 1395/3.28084, -998,  -998,  -998,  -998];
% POSI = [47.514, -122.317, 1395.7/3.28084, -998, -998, -998, -998];
% POSI = [47.391, -122.318, 1395.7/3.28084, -998, -998, -998, -998];
% POSI = [47.39, -122.32,  1400/3.28084, -998,  -998,  -998,  -998];

%NOT IN FOGGY (depending on the conditions, the storm could be very strong):
% POSI = [47.514, -122.32, 1395/3.28084, -998,  -998,  -998,  -998];
```

```
%NOT VALID: two main reasons:
% POSI = [47.38, -122.32, 1396/3.28084, -998, -998, -998,     -998];
% POSI = [47.52, -122.32, 1450/3.28084, -998,  -998,  -998,  -998];
% Too far from the localizer, it crashes in the last meters

% Changing the direction. The plane just follows the initial and don't turn
% around

%% Set initial conditions: autothrottle,localizer,approach mode(glideslope)
disp('Setting initial conditions')

% Set datarefs to work with
Apilot = {'sim/cockpit/autopilot/autopilot_state'};
% autopilot modes

gearst = {'sim/cockpit/switches/gear_handle_status',...
         'sim/cockpit/switches/auto_brake_settings'};
% gear status and settings of autobrake

Avalue = {'sim/cockpit/autopilot/altitude',...
         'sim/cockpit/autopilot/vertical_velocity',...
         'sim/cockpit/autopilot/heading'};
% autopilot values of dialed variables

simspd = {'sim/time/sim_speed'};
% speed of the simulation, it also unpauses

sendDREF(simspd{1},s_spd, Socket); % set speed of the sim, unpauses

sendDREF(Apilot{1},  1, Socket); % autothrottle
sendDREF(Apilot{1}, 256, Socket); % localizer
sendDREF(Apilot{1}, 1024, Socket); % approach mode

sendDREF(gearst{1},  1, Socket); % gear down->1; gear up->0
sendDREF(gearst{2},  4, Socket); % autobrakes 0-4

%These values or similar has to be dialed into the autopilot to perform
%correctly and not engage the altitude or other modes.
% sendDREF(Avalue{1}, 5100, Socket); % set altitude dialed in autopilot
% % it is which is going to be maintained when it is reached, due to the
% % landing, it doesn't interest to maintain an altitude, an therefore, an
% % altitude that is not going to be reached is settled. In that way if this
% % mode is armed, it won't erase the Localizer or approach modes.
% sendDREF(Avalue{2},  0, Socket); % set VVI dialed in autopilot
% sendDREF(Avalue{3}, 343, Socket); % set heading dialed in autopilot
% % going from south to north


%% Extract parameters
DREFS = {...
    'sim/cockpit2/gauges/indicators/altitude_ft_pilot',... 1
    'sim/flightmodel/position/y_agl',...                 2
    'sim/flightmodel2/controls/pitch_ratio',...          3
    'sim/flightmodel2/controls/roll_ratio',...           4
    'sim/flightmodel2/controls/heading_ratio',...        5
    'sim/cockpit2/engine/actuators/throttle_ratio_all',... 6
    'sim/flightmodel/position/groundspeed',...         7
    'sim/cockpit2/gauges/indicators/airspeed_kts_pilot',... 8
    'sim/flightmodel/controls/flaprat',...             9
    'sim/cockpit/switches/auto_brake_settings',...    10
    };

% DREFS:
% 1 altitude (feet) MSL (mean sea level), barometric pressure 29.92 inHg
% 2 altitude (meters) AGL radioaltimeter altitude above ground level
% 3 actual pitch (ratio), deflection of elevator
% 4 actual roll  (ratio), deflection of aileron
% 5 actual heading (ratio), deflection of rudder
% 6 throttle ratio (ratio)
% 7 groundspeed  (m/s), STOP CONDITION
% 8 indicated airspeed in knots
% 9 flap ratio deflection (ratio)
%10 autobrake ratio    (ratio)

% Position with the inner function POSI directly

%'sim/cockpit2/gauges/indicators/vvi_fpm_pilot',...
%indicated vertical speed (ft/min), it tends to lag by several seconds
%'sim/cockpit/misc/radio_altimeter_minimum' %set to zero

cont = 0; %number of asked connections
conn = 0; %number of successful connections
stop = 0; %stop condition

%Initial info printed in the file fd
fprintf(fd,['Flying %s from %s (%s point) of %s airport ',...
    'in %s (%s) during %s with %s weather \n'],...
    plane,distance,start_point,airport,land_way,direction,time,weather);

%Initial description of the data printed in the file fd
fprintf(fd,['Connection Latitude[deg] Longitude[deg] Altitude msl[ft] ', ...
    'Altitude agl[ft] Pitch[deg] Roll[deg] Heading[deg] ', ...
    'Pitch_ratio Roll_ratio Heading_ratio Throttle_ratio ',...
    'Vertical_velocity[fpm] Vg[m/s] V[knots] Flap_ratio ',...
    'Gear_status Autobrake \n']);

%Depending on the position it is going to be needed some number of
%data recovery or other according the distance to the runway
while stop==0
    cont = cont+1;
    try

        % Get data
        posi   = getPOSI(0, Socket);
        ctrl   = getCTRL(0, Socket);
        result = getDREFs(DREFS,Socket);

        %POSI
        %          1. Latitude (deg)
        %          2. Longitude (deg)
        %          3. Altitude (m above MSL)
        %          4. Roll (deg)
        %          5. Pitch (deg)
        %           6. True Heading (deg)
        %           7. Gear (0=up, 1=down)

        %CTRL
        %           1. Latitudinal Stick [-1,1]
        %           2. Longitudinal Stick [-1,1]
        %           3. Pedal [-1, 1]
        %           4. Throttle [-1, 1]
        %           5. Gear (0=up, 1=down)
        %           6. Flaps [0, 1]
        %           7. Speed Brakes [-0.5, 1.5]

        % Manipulate data
        if cont==1
            h_i = posi(3)*3.28084; % set h_i to calculate vertical velocity
            %initial altitude in feet in order to calculate the vertical
            %velocity by deriving the altitude with incremental cocients
            %due to the lag of the vvi of the airplane
        end
        h_f = posi(3)*3.28084; %new altitude in feet
        vvi = (h_f-h_i)/(interval)*60; % vertical velocity feet/min
        h_i = h_f;

        lat    = posi(1);
        long   = posi(2);
        altmsl = posi(3)*3.28084; %altitude MSL in feet ídem result(1)
        altagl = result(2)*3.28084; %altitude AGL in feet
        pitch  = posi(5);
        roll   = posi(4);
        head   = posi(6);
        pitchr = result(3);
        rollr  = result(4);
        headr  = result(5);
        Tr     = result(6);
        vvel   = vvi;
        Vg     = result(7);
        V      = result(8);
        flaps  = result(9);
        gear   = round(ctrl(5));
        autob  = result(10);

        % Print and save data in Command Window, .txt file and .mat file
        fprintf(['%3d Lat:%5.3f Long:%6.3f Alt_MSL[ft]:%6.2f ',...
            'Alt_AGL[ft]:%5.2f Pitch:%5.2f Roll:%5.2f Heading:%5.2f ',...
            'Pitch_ratio:%5.2f Roll_ratio:%5.2f Heading_ratio:%5.2f ',...
            'T_ratio:%3.2f Vz[ft/min]:%7.2f Vg[m/s]:%5.2f ',...
            'V[knots]:%5.2f Flaps:%3.2f Gear:%d Autob:%d \n'],...
            [cont, lat, long, altmsl,...
            altagl, pitch, roll, head,...
            pitchr, rollr, headr,...
            Tr, vvel, Vg,...
            V, flaps, gear, autob]);

        fprintf(fd,['%3d %5.3f %6.3f %6.2f %5.2f %5.2f %5.2f %5.2f ',...
            '%5.2f %5.2f %5.2f %3.2f %7.2f %5.2f %5.2f %3.2f %d %d \n'],...
            [cont, lat, long, altmsl, altagl, pitch, roll, head,...
            pitchr, rollr, headr, Tr, vvel, Vg, V, flaps, gear, autob]);
```

```
        conn = conn+1;

        m.cont(1,conn) = cont;
        m.lat(1,conn)  = lat;
        m.long(1,conn) = long;
        m.altmsl(1,conn) = altmsl;
        m.altagl(1,conn) = altagl;
        m.pitch(1,conn) = pitch;
        m.roll(1,conn) = roll;
        m.head(1,conn) = head;
        m.pitchr(1,conn) = pitchr;
        m.rollr(1,conn) = rollr;
        m.headr(1,conn) = headr;
        m.Tr(1,conn)   = Tr;
        m.vvel(1,conn) = vvel;
        m.Vg(1,conn)   = Vg;
        m.V(1,conn)    = V;
        m.flaps(1,conn) = flaps;
        m.gear(1,conn) = gear;
        m.autob(1,conn) = autob;

        if altagl<=10 && Vg<0.001 %ATTENTION: stop condition
            % altagl = result(2) = altitude in ft above ground level
            % Vg     = result(7) = groundspeed in m/s
            fclose(fd); %Close file. Returns 0 if it is correctly closed
            fd=fopen([path,'.txt'],'a'); %Add information in the file
            fprintf(fd,['Time: %3.2f s. \n%d successful connections ', ...
                'from a total of %d. Connectivity: %2.2f %%\n'],...
                cont*interval,conn,cont,conn/cont*100);
            fprintf(['Time:%3.2f s. \n%d successful connections ', ...
                'from a total of %d. Connectivity: %2.2f %%\n'],...
                cont*interval,conn,cont,conn/cont*100);
            fclose(fd); %Close file.
            stop = 1;
        end
        pause(interval); % to record exactly when it is wanted

    catch ME
        disp(ME.identifier)
        if strcmp(ME.identifier,'MATLAB:Java:GenericException') ...
                || strcmp(ME.identifier,'MATLAB:badsubscript')
        end
    end
end

closeUDP(Socket);
```

9.2 landing.m

It generates a figure such as Figure 7.4, representing the X-Plane data in vertical and horizontal views. The inner function *airportdata* contains airport information that can be changed in Subsection 9.2.1.

Code 9.2 landing.m.

```
%Utility to draw the vertical and horizontal views of the glidepath
%Important, the x and y to draw should be transformed from degrees of
%latitude and longitude to feet.
%In the extract data code the less calculation is done the better, to take
%data in 0.2s, so the transformation has to be done here, from the latitude
%and longitude, accordding with the altitude.
%UNITS IN IMPERIAL SYSTEM: feet and degrees

clear all, clc, close all

%% DATA information (CAN BE CHANGED)
addpath XPC_original_data_copied

% load('NSclear3nmi16Rdefault1.mat')
% load('SNclear3nmi34Ldefault1.mat')
load('NSfoggy3nmi16Rdefault1.mat')
% load('SNfoggy3nmi34ldefault1.mat')

%If in the airport there aren't glideslope or localizer, set the data
%of where they would be optimally: the point to touchdown and the point
%where to locate the origin point of horizontal guide
airport = airportdata; %Change in this file any of the airport information

%Information of the north extreme
rWNS = airport.rWNS; %name of runway from North to South
latN = airport.latN; %latitude
longN = airport.longN; %longitude
elevN = airport.elevN; %altitude MSL, it isn't used
zGSaN = airport.zGSaN; %altitude MSL glideslope in north

%Information of the south extreme
rwSN = airport.rwSN; %name of runway from South to North
latS = airport.latS; %latitude
longS = airport.longS; %longitude
elevS = airport.elevS; %altitude MSL, it isn't used
zGSaS = airport.zGSaS; %altitude MSL glideslope in south

%Runway given by two points in its central line:
lr = airport.lr; %the longitudinal length of runway
wr = airport.wr; %width of runway

%Localizer
xLOCr     = airport.xLOCr;    %distance between the extreme of runway
                              %and the LOC installation
yLOCr     = airport.yLOCr;    %distance between the axis of runway and
                              %the LOC installation
zLOC      = airport.zLOC;     %altitude AGL
incrthetaL = airport.incrthetaL; %degrees from the central line of the cone
```

```
%Glideslope
xGSr      = airport.xGSr; %distance between the extreme of runway and the
                          %GS installation
yGSr      = airport.yGSr; %distance between the axis of runway and the GS
                          %installation
zGS       = airport.zGS; %altitude AGL
incrthetaG = airport.incrthetaG; %degrees from the central line of the cone

%% Conversion
%From degrees to feet conversion using KSEA diagram of FAA:
%In the plane of the KSEA airport, two measures are represented:
% 47º 26' = 47.43º
% 47º 27' = 47.45º
% Between them, their distance is divided into 10 segments.
% 1 segment = 0.1' = 0.0017º
% The length of the runway is 8500 feet and it takes 14 segments like
% those, that means 14*0.0017º = 0.0233º
% So, the conversion could be: 8500 feet / 0.0233º = 364290 feet/degree
% Or what is the same: 2.7451e-06 degrees/feet
% conv=364290; %feet in one degree

%Other conversion: 360º of longitude in the equatorial circumference of
%40075161.2 meters (Earth radius of the semi-major axis of the Earth at the
%equator is 6378137 meters) gives 111319.9 meters/degree
% conv = 111319.9*3.28084; %feet in one degree
%The inconvenient of this one is the inaccuracy for the longitude
%conversion.

%Haversine formula (in radians) to get the conversion factors:
% -latitude to feet
% -longitude to feet
%It calculates the great-circle distance between two points, the shortest
%distance over the Earth's surface.This is needed above all due to the
%decreasing distance between two meridians, that goes to zero when going
%to the poles, multiplied by the cosine of the latitude.
%The two previous conversions aren't used.
% R_Earth = 6371e3*3.28084; %feet of Earth's radius
%
% for ii=1:2
%     if ii==1 %1 degree of latitude at the location of the airport
%         %for example, the north, it could also have been south. For that,
%         %put S where N is inside this if.
%         lat1    = latN;
%         lat2    = latN+1;
%         long1   = longN;
%         long2   = longN;
%
%     else %1 degree of longitude at the location of the airport
%         %for example, the north, it could also have been south. For that,
%         %put S where N is inside this if.
%         lat1    = latN;
%         lat2    = latN;
%         long1   = longN;
%         long2   = longN+1;
%     end
%
%     ilat    = lat2-lat1;
%     ilong   = long2-long1;
%     iphi    = ilat*pi/180; %in radians
%     ilambda = ilong*pi/180; %in radians
%
%     a = (sin(iphi/2))^2+cos(lat1*pi/180)*cos(lat2*pi/180)*(sin(ilambda/2))^2;
%     c = 2*atan2(sqrt(a),sqrt(1-a));
%     d = R_Earth*c;
%
%     if ii==1
%         convlat = d; %feet in a degree of latitude at that location
%     else
%         convlong = d; %feet in a degree of longitude at that location
%     end
% end

%The value of convlat and convlong could be implemented as the fixed value
%obtained instead of running the previous lines every time:
convlat = 364812.76;
convlong = 246631.93;

incrlat = latN-latS;
incrlong = longS-longN;

xr = [0,incrlat*convlat]; %points that define the runway
yr = [0,incrlong*convlong]; %points that define the
zr = [0,0]; %this would be elevation if it was in MSL instead of AGL

delta = abs(atand(diff(yr)/diff(xr)));

if lat(1)>latN
    runway = rWNS; %runway of approach, from north to south
    direct = ' (from North to South)';
    latr   = latS; %degrees of the second extreme of the runway, the south
    longr  = longS; %degrees of the second extreme of the runway, longitude
    yGSr   = -yGSr;
    %At least in KSEA, glideslope is at the right going to north and at the
    %left going to south
    altgs = zGSaN;
    SN    = -1;    %it indicates that it doesn't go from South to North
else
    runway = rwSN; %runway of approach, from south to north
    direct = ' (from South to North)';
    latr   = latN; %degrees of the second extreme of the runway, the north
    longr  = longN; %degrees of the second extreme of the runway, longitude
    altgs  = zGSaS;
    SN     = 1;    %it indicates that it goes from South to North
end

xa = abs(lat-latr)*convlat; %of the aircraft to draw
ya = SN*(long-longr)*convlong; %of the aircraft to draw
za = zeros(1,length(altmsl)); %of the aircraft to draw

runw = 0;                    %it starts in the air, no in the runway
for ii=1:length(altmsl)
    if altagl(ii)>15 && runw==0
        za(ii) = altmsl(ii)-altgs; %to avoid the interference of buildings
    else
```

```
        za(ii) = altagl(ii);     %because not having the runway profile
        runw   = 1;              %this is needed, because the runway
        %profile varies and without this parameter, it would return the take
        %the first value of za instead of this one once it has touchdown.
    end
end

%% Representation
im1 = figure;

%% Vertical plane with initial point in the GS
subplot(2,1,1)
title(['Vertical plane - glideslope. Landing in ',runway,direct])
xlabel('horizontal distance [feet]')
ylabel('height [feet AGL]') %AGL: above ground level

%Runway 34L-16R
runwayv = line(xr,zr,'Color','k','LineWidth',2);
hold on

%Aircraft path
plot(xa,za,'r*')

%Location of LOC
xLOC = xr(1)-xLOCr*cosd(delta);
plot(xLOC,zLOC,'MarkerEdgeColor',[0 0.5 0],'Marker','+',...
    'MarkerSize',10,'LineWidth',5)

%Location of GS
xGS = xr(2)-xGSr*cosd(delta)+yGSr*sind(delta);
plot(xGS,zGS,'+b','MarkerSize',10,'LineWidth',5)

%Center line of glideslope
xg = [xGS,xa(1)];
zg = [zGS,tand(3)*(xg(2)-xg(1))+zGS]; %3 degrees is the optimal
glideslope=line(xg,zg,'Color','b','LineStyle','--');

%Lines of the vcone: u:upper; l:lower
thetag = atand(diff(zg)/diff(xg));
Lg     = sqrt(diff(xg)^2+diff(zg)^2);

for ii=1:length(incrthetaL)
    cug    = [xg(1)+Lg*cosd(thetag+incrthetaG(ii)),zg(1)+...
        Lg*sind(thetag+incrthetaG(ii))];
    clg    = [xg(1)+Lg*cosd(thetag-incrthetaG(ii)),zg(1)+...
        Lg*sind(thetag-incrthetaG(ii))];
    xcug   = [xg(1),cug(1)];
    zcug   = [zg(1),cug(2)];
    vconeu = line(xcug,zcug,'Color','b');
    xclg   = [xg(1),clg(1)];
    zclg   = [zg(1),clg(2)];
    vconel = line(xclg,zclg,'Color','b');
end

%Height of transition
ht = 10; %feet AGL. According to the FAA, 10-20 feet AGL
xh = [0,xg(2)+2];
```

```
zh = [ht,ht];
htrans = line(xh,zh,'Color','k','LineStyle','-.');

% axis equal %to see properly the scale

%% Horizontal plane changes with initial point in the LOC
subplot(2,1,2)
title(['Horizontal plane - localizer. Landing in ',runway,direct])
xlabel('horizontal distance [feet]')
ylabel('top view [feet AGL]') %AGL: above ground level

%Runway 34L-16R
runwayh = line(xr,yr,'Color','k','LineWidth',2,'LineStyle','--');
xr1     = xr+[wr,wr]/2*sind(delta);
yr1     = yr+[wr,wr]/2*cosd(delta); %semiwidth of the runway
xr2     = xr-[wr,wr]/2*sind(delta);
yr2     = yr-[wr,wr]/2*cosd(delta); %semiwidth of the runway
runwayh1 = line(xr1,yr1,'Color','k','LineWidth',2);
runwayh2 = line(xr2,yr2,'Color','k','LineWidth',2);
hold on

%Aircraft path
plot(xa,ya,'r*')

%Location of LOC
xLOC = xr(1)-xLOCr*cosd(delta);
plot(xLOC,(yLOCr+sind(delta)*xLOCr),'MarkerEdgeColor',[0 0.5 0],...
    'Marker','+','MarkerSize',10,'LineWidth',5)

%Location of GS
xGS = xr(2)-xGSr*cosd(delta)+yGSr*sind(delta);
plot(xGS,(yGSr*cosd(delta)-xGS*tand(delta)),'+b','MarkerSize',10,'LineWidth',5)

%Center line of LOC
xloc=[xLOC,xa(1)];
yloc=[yLOCr*cosd(delta)+sind(delta)*xLOCr,-xa(1)*tand(delta)];
hold on
LOC=line(xloc,yloc,'Color',[0 0.5 0],'LineStyle','--');

%Lines of the hcone: u:upper; l:lower; 1:interior cone; 2: exterior cone
thetaL=atand(diff(yloc)/diff(xloc));
Ll=sqrt(diff(xloc)^2+diff(yloc)^2);

for ii=1:length(incrthetaL)
    cuL=[xloc(1)+Ll*cosd(thetaL+incrthetaL(ii)),yloc(1)+...
       Ll*sind(thetaL+incrthetaL(ii))];
    clL=[xloc(1)+Ll*cosd(thetaL-incrthetaL(ii)),yloc(1)+...
       Ll*sind(thetaL-incrthetaL(ii))];
    xcuL=[xloc(1),cuL(1)];
    ycuL=[yloc(1),cuL(2)];
    hcone1u=line(xcuL,ycuL,'Color',[0 0.5 0]);
    xclL=[xloc(1),clL(1)];
    yclL=[yloc(1),clL(2)];
    hconel=line(xclL,yclL,'Color',[0 0.5 0]);
end
% axis equal
```

9.2.1 airportdata.m

Code 9.3 airportdata.m.

```
function airport=airportdata
% Function that provides information about runway, glideslope and localizer.
%If in the airport there aren't glideslope or localizer, set the data
%of where they would be optimally: the point to touchdown and the point
%where to locate the origin point of horizontal guide

%Information of the north extreme
airport.rWNS = '16R';     %name of runway from North to South
airport.latN = 47.46384; %latitude
airport.longN = -122.31785; %longitude
airport.elevN = 330.33;  %altitude MSL, it isn't used
airport.zGSaN = 351.79;  %altitude MSL glideslope in north

%Information of the south extreme
airport.rwSN = '34L';     %name of runway from South to North
airport.latS = 47.44056; %latitude
airport.longS = -122.31808; %longitude
airport.elevS = 353;      %altitude MSL, it isn't used
airport.zGSaS = 359.86;  %altitude MSL glideslope in south

%Runway given by two points in its central line:
airport.lr = 8500; %the longitudinal length of runway
airport.wr = 150; %width of runway

%Localizer
airport.xLOCr = 800; %distance between the extreme of runway and the LOC
airport.yLOCr = 0; %distance between the axis of runway and the LOC
airport.zLOC = 0;  %altitude AGL
airport.incrthetaL = [1.5,3]; %degrees from the central line of the cone(s)
% try to put a difference such as the space created is equally distributed
% between cones, for example: incrthetaL=[1,3] gives an interior cone of 2
% degrees and a exterior one of 6 degrees in total
% [1.5,3] is what gives the most similar result to what X-Plane uses,
% following the normative

%Glideslope
airport.xGSr = 793; %distance between the extreme of runway and the GS
airport.yGSr = 175; %distance between the axis of runway and the GS
airport.zGS = 0;  %altitude AGL
airport.incrthetaG = [0.25,0.7]; %degrees from the central line of the cone
% try to put a difference such as the space created is equally distributed
% between cones, for example: incrthetaG=[1,3] gives an interior cone of 2
% degrees and a exterior one of 6 degrees in total
% [0.25, 0.7] is what gives the most similar result to what X-Plane uses,
% following the normative
```

9.3 XPC_IRL_converter.m

Code 9.4 XPC_IRL_converter.m.

```
function XPC_IRL_converter
% Program to convert data taken from X-Plane with X-Plane Connect (XPC)
% Toolbox into the kind of data managed by the IRL Toolkit
% It produces a .mat file which contains a structure: autopilot_result,
% with two fields:
% - mdp_params, a struct with the common data of every iteration
% - example_samples, with a cell matrix that contains in each cell a state
%   and an action, defined by:
%      - columns with the number of steps considered (it doesn't have to be
%        a complete iteration, it could be divided into several segments
%      - rows with the different iterations (they have to have the same
%        characteristics)
% IMPORTANT:
% Copy the files generated by XPC to the folders:
% - landing_demos, which is going to be modified
% - XPC original data, for doing tests and don't loose the data.
%   In that case, browse the data in landing_demos and copy again from this
%   or from the folder of XPC
% Then run this script to generate the landing_i.mat files with the
% segments of a set of iterations.

clear all, clc, close all
%% DATA information (CAN BE CHANGED)
%FILE CONFIGURATION
tlen = 300; %total length to uniformize the iterations
len  =  50; %length of the segment SEE the Note about len
% NOTE ABOUT len. It is a parameter to have the landing discretized in
% segments of the same length (for example 60 steps of time, in which the
% connection has been successful), because in each iteration, the number of
% taken data is different. When it is in the last segment, every landing is
% different and to solve that, those with a minor number of components are
% filled repeating their last value.
% The number of instants of tiemea landing performed with 0.2s as step time
% from 3 nmi of the runway oscillates between 261 and 280.
% If the discretized length is 50, taking a total extension of tlen as 300
% instants, it would result in 6 segments and of files.
ini       = -1; %initial point of the range
last      =  1; %last point of the range
divisions = 21; %number of possible values in the range
%201 with range [-1,1] and step of 0.01
%Another good option would be 21: [-1,1] and step of 0.1
%This first method allows to discretise variables to have less values. It
%is useful if the discretisation is approximately logical and simplifies
%the result (a correct combination of ini, last, divisions; according to
%the original values, such as the two options presented).
inis      =   0; %initial value of the range of indicated airspeed
lasts     = 200; %last value of the range of indicated airspeed
%for speed, the initial value doesn't reach 200knots and the last is 0knots
%It is considered the same number of divisions, but this divisions are now
%limits of the segments
%This second method is useful to take directly the component of the possible
```

```
%values of a parameter. The difference is that with that, the variable
%isn't discretised. While the first one gives divisions as the number of
%possible values, in this one, they are one possible value less due to be
%between two consecutive values. (SEE the paper of the project to
%understand it better).
%If the values outside the range are also considered, the number of
%components of possible values are 2 more than the provided by the
%first method. This is applied for the horizontal and vertical lanes
%related to longitude and altitude.

% AIRPORT
%If in the airport there aren't glideslope or localizer, set the data
%of where they would be optimally: the point to touchdown and the point
%where to locate the origin point of horizontal guide
airport = airportdata; %Change in this file any of the airport information

% IRL ALGORITHM
contin  = 0;    %discrete (0) or continuous (1)
deter   = 1.0;  %probability of correct transition
discount = 0.9; %temporal discount factor to use

%% Processing
addpath landing_demos %directory where the .mat files are located

files = dir('landing_demos/*.mat'); %files of .mat extension in the
%directory where the generated XPC files are

%CAUTION: it includes some files (generally two) that have directory
%purposes: "." and "..". For that, only the .mat files are considered. That
%also allows to have the .txt files which provide information about landings

for ii=1:length(files)
    ii %To show that it is working and how the progress is going.
    file = matfile(files(ii).name, 'writable', true); % Open .mat file to write

%     Note: there is a problem when the connection fails, the vertical
%     velocity increases in the right next iteration after the bad
%     connection. To solve this, unusual values are searched and
%     substituted with the media of the previous (it has been considered 0
%     if it was the first component or the last one) and the next one.

    ind = find(file.vvel>1e4); %indices of the components that are too high
    if ~isempty(ind)        %to do this operations only if it has to
        for k=1:length(ind)             %for each of the found indices
            if ind(k) == 1              %the previous case doesn't exist
                file.vvel(1,ind(k)-1) = 0; %the previous case is created
            elseif ind(k) == length(file.V) %the next case doesn't exist
                file.vvel(1,ind(k)+1) = 0; %the next case is created
            end
            file.vvel(1,ind(k))=(file.vvel(1,ind(k)+1)+file.vvel(1,ind(k)-1))/2;
            %the media of the previous and the next values of the wrong one
        end
    end

    %% Extend parameters to have the same length in all files
    file.Tr(1,end:tlen)   = file.Tr(1,end);
    file.V(1,end:tlen)    = file.V(1,end);
    file.Vg(1,end:tlen)   = file.Vg(1,end);
    file.altagl(1,end:tlen) = file.altagl(1,end);
    file.altmsl(1,end:tlen) = file.altmsl(1,end);
    file.autob(1,end:tlen) = file.autob(1,end);
    file.pitch(1,end:tlen) = file.pitch(1,end);
    file.roll(1,end:tlen) = file.roll(1,end);
    file.head(1,end:tlen) = file.head(1,end);
    file.pitchr(1,end:tlen) = file.pitchr(1,end);
    file.rollr(1,end:tlen) = file.rollr(1,end);
    file.headr(1,end:tlen) = file.headr(1,end);
    file.cont(1,end:tlen) = file.cont (1,end);
    file.flaps(1,end:tlen) = file.flaps(1,end);
    file.gear(1,end:tlen) = file.gear(1,end);
    file.lat(1,end:tlen)  = file.lat(1,end);
    file.long(1,end:tlen) = file.long(1,end);
    file.vvel(1,end:tlen) = file.vvel(1,end);

    %% Pitch, roll, heading and throttle ratios and V into coordinates
    %Discretize to the possible values and then convert them to coordinates
    %to generate the state
    %%%ATTENTION
    %The discretized coordinates are with '_r' and the obtained from
    %X-Plane don't have '_'
    [file.pitch_r,file.roll_r,file.head_r,file.T_r,file.V_] = ...
       ratiostocoords(file.pitchr,file.rollr,file.headr,...
       file.Tr,file.V,ini,last,divisions,inis,lasts);

    Vs = divisions-1;         %number of possible indicated airspeeds

    %% Latitude, longitude, altitude into coordinates
    %Discretize to the possible values and then convert them to coordinates
    %to generate the state
    %- Latitude is going to be the direction of x, going one step forwards
    %  each step of time
    %- Altitude is going to be transformed into vertical lanes. It is going
    %  to be defined as alt(msl)-alt(respective glideslope) to avoid landing
    %  interferences and after reaching 50 ft, agl (above ground level)
    %- Longitude is going to be transformed into horizontal lanes
    [file.xv,file.Vlanev,file.Hlanev]=degreestocoords(file.lat,...
       file.long,file.altmsl,file.altagl,airport);

    Hlanes = 2*length(airport.incrthetaL)+1; %number of possible horizontal lanes
    Vlanes = 2*length(airport.incrthetaG)+1; %number of possible vertical lanes
end

%% First structure, mdp_params
mdp_params=struct(...
    'seed',0,...          %initialization for random seed
    'ini',ini,...         %initial point of the range of all ratios
    'last',last,...       %last point of the range of all ratios
    'inis',inis,...       %initial point of the range of indicated airspeed
    'lasts',lasts,...     %last point of the range of indicated airspeed
    'length',len,...      %number of iterations considered in a segment
    'Hlanes',Hlanes,...   %number of possible horizontal lanes (longitude)
    'Vlanes',Vlanes,...   %number of possible vertical lanes (altitude)
    'pitchrs',divisions,... %number of possible pitch ratios
    'rollrs',divisions,... %number of possible roll ratios
    'headrs',divisions,... %number of possible heading ratios
    'Trs',divisions,...   %number of possible T ratios
    'Vs',Vs,...           %number of possible indicated airspeeds
    'airport',airport,... %data about runway, glideslope and localizer
    'continuous',contin,... %discrete (0) or continuous (1)
    'determinism',deter,... %probability of correct transition
    'discount',discount); %temporal discount factor to use

%% Second structure, example_sample
% example_samples is a matrix with:
%    - the length of the 'road' considered as number of columns
%    - the number of iterations as number of rows
% in each cell of a row, a vector with the state and the actions indicates
% the sequence of the followed trajectory

%% Files to save
files_gen = tlen/len; %number of files that are going to be the segments

for file_gen=1:files_gen
    file_name=['landing_',num2str(file_gen),'.mat']; %file to use with IRL

    s = zeros(length(files),len); %matrix of states
    a = zeros(length(files),len); %matrix of actions
    example_samples = cell(length(files),len); %cell matrix to save both

    for ii=1:length(files)
        fd=load(files(ii).name);

        %For actions it is more accurate to use the data obtained with
        %X-Plane without discretise or converting to coordinates
        x=(file_gen-1)*len; %previous component to use in present segment
        dp=[diff(fd.pitchr),0]; %due to diff, one component gets lost
        dr=[diff(fd.rollr),0]; %due to diff, one component gets lost
        dy=[diff(fd.headr),0]; %due to diff, one component gets lost
        dT=[diff(fd.Tr),0];   %due to diff, one component gets lost
        totald=[dp;dr;dy;dT];
        [~,maxind]=max(abs(totald));
        %maxval: maximum absolute value of difference
        %maxind: index of row with the maximum value

        %For states it is needed the discretised parameters converted to
        %coordinates
        T_r     = fd.T_r;
        pitch_r = fd.pitch_r;
        roll_r  = fd.roll_r;
        head_r  = fd.head_r;
        xv      = fd.xv; %the same as x
        Hlanev  = fd.Hlanev;
        Vlanev  = fd.Vlanev;
        V_      = fd.V_;

        for jj=1:len
            %STATES: from the definition of the coordinates in pitch, roll,
            %heading and throttle, an unique value is generated and save as
            %the state in that step of time using this function:
            s(ii,jj) = landingcoordstostate(x+jj,Hlanev(1,x+jj),...
                Vlanev(1,x+jj),pitch_r(1,x+jj),roll_r(1,x+jj),...
                head_r(1,x+jj),T_r(1,x+jj),V_(1,x+jj),mdp_params);

            %ACTIONS: each step time an action is taken among 9 of them.
            %The process to choose what to do is based in the biggest
            %difference of the variables each step of time, after observing
            %the behaviour. After several steps, throttle is inoperative,
            %because of getting null. If all of them are equal (even zero),
            %it takes the first one that is the maximum: pitchr (first row)
            if maxind(x+jj) == 1
                if totald(1,x+jj)<0
                    a(ii,jj) = 1; %decrease pitch_ratio
                elseif totald(1,x+jj)>0
                    a(ii,jj) = 2; %increase pitch_ratio
                elseif totald(1,x+jj) == 0
                    a(ii,jj) = 9; %no action
                end
            elseif maxind(x+jj) == 2
                if totald(2,x+jj)<0
                    a(ii,jj) = 3; %decrease roll_ratio
                else
                    a(ii,jj) = 4; %increase roll_ratio
                end
            elseif maxind(x+jj) == 3
                if totald(3,x+jj)<0
                    a(ii,jj) = 5; %decrease heading_ratio
                else
                    a(ii,jj) = 6; %increase heading_ratio
                end
            elseif maxind(x+jj) == 4
                if totald(4,x+jj)<0
                    a(ii,jj) = 7; %decrease throttle_ratio
                else
                    a(ii,jj) = 8; %increase throttle_ratio
                end
            else
                a(ii,jj) = 9; %no action
            end

            %IMPROVEMENT: take into account several actions at the same
            %time, either with an index or changing the algorithm code
            %(much more complicated)

            example_samples{ii,jj}=[s(ii,jj); a(ii,jj)];
        end
    end
    example_samples = {example_samples}; %to create a cell matrix

    autopilot_result = struct(...
        'example_samples',example_samples,...
        'mdp_params',mdp_params);

    save(file_name,'autopilot_result');
end
```

The functions mentioned in this code are in the following Subsection 9.2.1, Subsection 9.3.1, Subsection 9.3.2 and Subsection 9.3.3.

9.3.1 ratiostocoords.m

Code 9.5 ratiostocoords.m.

```
function [pitch_r,roll_r,head_r,T_r,V_] = ...
    ratiostocoords(pitchr,rollr,headr,Tr,V,ini,last,divisions,inis,lasts)
%Function for converting from ratios to components of the possible values.

%% Discretize to the possible values
%Before converting ratios to coordinates, they are discretized to their
%possible values. It has taken into account the same range and possible
%values for all of them.

vect=(ini:(last-ini)/(divisions-1):last); %vector of possible values

for ii=1:4 %variables of control pitch, roll, heading and T ratios
    if ii==1
        var = pitchr; %var is used to reduce the lines of the code
    elseif ii==2
        var = rollr;
    elseif ii==3
        var = headr;
    elseif ii==4
        var = Tr;
    end
    for jj = 1:length(var) %for every component of var
        for kk = 1:length(vect)-1 %for every component of possible
            %values except the last one, because it is analysing if the
            %X-Plane values are inside two close possible values given
            %by range (ini and last) and divisions
            if var(jj)>=vect(kk) && var(jj)<(vect(kk+1)+vect(kk))/2
                var(jj) = vect(kk); %associate with the inferior value
            elseif var(jj)>=(vect(kk+1)+vect(kk))/2 && var(jj)<=vect(kk+1)
                var(jj) = vect(kk+1); %associate with the superior value
            end
        end
    end
    if ii==1
        pitchr = var; %get back the name parameter discretized
    elseif ii==2
        rollr = var;
    elseif ii==3
        headr = var;
    elseif ii==4
        Tr = var;
    end
end

% IMPROVEMENT: generalize this to do it with another loop using for,
% for example, instead of doing with every field, being able to
% do with every field included, not these that are fixed.
% So if a field is added, the code doesn't need to be changed,
% adding a line for the new field. Exclude field Properties (it only is
% created by default with command matlabfile, with the properties of the
% .mat Both options give problems:
% namefield = fieldnames(file); % Name of fields of file
% for jj=1:length(namefield)-1 %to not overwrite Properties
%    namefield{jj}(end:tlen)=files(ii).name.namefield{jj}(end) %1
%    set(file,namefield{jj}(end:tlen),namefield{jj}(end)) %2
% end


%% Ratios to coords
%pitchr, rollr, headr and Tr are ratios, and therefore, a conversion
%to the 'component of the possible number of each parameters is needed'.
%For example, pitchrs could be 201 possible values, from -1 to 1 with a step
%of 0.01. So the values of a linear conversion have to be found.
%Two possible equations are:
%-1*a+b=1 %the first possible value of pitchr
%1*a+b=201 %the last possible value of pitchr
%Substracting: 2a=200, a=100 and therefore, b is 101

%If pitchr is 0.1, its conversion would be: 0.1*100+101=112
%It is the same with each of the rest parameters

%Generalizing for a number of possible pitch_r such as pitchrs and [ini,last]:
%ini*a+b=1      %the first possible value of pitch_r
%last*a+b=pitchrs %the last possible value of pitch_r
%Subtracting: (last-ini)*a=pitchrs-1, a=(pitchrs-1)/(last-ini) and therefore,
%b=1-a*ini; b=1-ini*(pitchrs-1)/(last-ini)-, b=(last-ini*pitchrs)/(last-ini);
%It is going to be considered the same number of possible ratios of all
%variables: pitchrs=rollrs=headrs=Trs=divisions

%The last abstraction would be converting and having different ranges of
%each variable or/and different number of possible values of each variable
%It is a future improvement

a = (divisions-1)/(last-ini);
b = 1-ini*(divisions-1)/(last-ini);

pitch_r = pitchr*a+b;
roll_r = rollr*a+b;
head_r = headr*a+b;
T_r    = Tr*a+b;

%% Speed
vects = (inis:(lasts-inis)/(divisions-1):lasts);
V_ = zeros(1:length(length(V)));
for ii = 1:length(V) %for every component of V
    for jj = 1:length(vects)-1 %for every component of possible values
        if  V(ii)>=vects(jj) && V(ii)<=vects(jj+1)
            V_(ii) = jj;
%         else
%             V_(ii) = jj+1; %it is the case of not having calculate
            %accurately the end of the range, and that some values
            %of speed exceed it.
        %However, this can't be implemented due to change in each iteration
        %the number of possible speeds and therefore changing the state.
        end
    end
end
```

9.3.2 degreestocoords.m

Code 9.6 degreestocoords.m.

```
function [xv,Vlanev,Hlanev]=degreestocoords(lat,long,altmsl,altagl,airport)
%Utility for converting latitude, longitude and altitude into simplified
%coordinates such as vectors of one step discretized.
%Only for converting X-Plane data into the kind of data managed by IRL
%Toolkit

%% Conversion degrees to feet
%From degrees to feet conversion using KSEA diagram of FAA:
%In the plane of the KSEA airport, two measures are represented:
% 47º 26' = 47.43º
% 47º 27' = 47.45º
% Between them, their distance is divided into 10 segments.
% 1 segment = 0.1' = 0.0017º
% The length of the runway is 8500 feet and it takes 14 segments like
% those, that means 14*0.0017º = 0.0233º
% So, the conversion could be: 8500 feet / 0.0233º = 364290 feet/degree
% Or what is the same: 2.7451e-06 degrees/feet
% conv=364290; %feet in one degree

%Other conversion: 360º of longitude in the equatorial circumference of
%40075161.2 meters (Earth radius of the semi-major axis of the Earth at the
%equator is 6378137 meters) gives 111319.9 meters/degree
% conv = 111319.9*3.28084; %feet in one degree
%The inconvenient of this one is the inaccuracy for the longitude
%conversion.

%Haversine formula (in radians) to get the conversion factors:
% -increment of latitude to feet
% -increment of longitude to feet
%It calculates the great-circle distance between two points, the shortest
%distance over the Earth's surface.This is needed above all due to the
%decreasing distance between two meridians, that goes to zero when going
%to the poles, multiplied by the cosine of the latitude.
%The two previous conversions aren't used.
% R_Earth = 6371e3*3.28084; %feet of Earth's radius
%
% for ii=1:2
%     if ii==1 %1 degree of latitude at the location of the airport
%         %for example, the north, it could also have been south. For that,
%         %put S where N is inside this if.
%         lat1   = airport.latN;
%         lat2   = airport.latN+1;
%         long1  = airport.longN;
%         long2  = airport.longN;
%
%     else %1 degree of longitude at the location of the airport
%         %for example, the north, it could also have been south. For that,
%         %put S where N is inside this if.
%         lat1   = airport.latN;
%         lat2   = airport.latN;
%         long1  = airport.longN;
%         long2  = airport.longN+1;
%     end
%
%     ilat   = lat2-lat1;
%     ilong  = long2-long1;
%     iphi   = ilat*pi/180; %in radians
%     ilambda = ilong*pi/180; %in radians
%
%     a = (sin(iphi/2))^2+cos(lat1*pi/180)*cos(lat2*pi/180)*(sin(ilambda/2))^2;
%     c = 2*atan2(sqrt(a),sqrt(1-a));
%     d = R_Earth*c;
%
%     if ii==1
%         convlat = d; %feet in a degree of latitude at that location
%     else
%         convlong = d; %feet in a degree of longitude at that location
%     end
% end

%The value of convlat and convlong could is implemented as the fixed value
%obtained instead of running the previous lines every time
convlat = 364812.76;
convlong = 246631.93;

%Data from struct airport:
incrthetaG = airport.incrthetaG; %degrees of cones in vertical plane
xGSr       = airport.xGSr;     %distance between GS and runway in x axis
yGSr       = airport.yGSr;     %distance between GS and runway in y axis

incrthetaL = airport.incrthetaL; %degrees of cones in horizontal plane
xLOCr      = airport.xLOCr;    %distance between LOC and runway in x axis
yLOCr      = airport.yLOCr;    %distance between LOC and runway in y axis

incrlat = airport.latN-airport.latS;
incrlong = airport.longS-airport.longN;

if lat(1)>airport.latN
    latr  = airport.latS; %degrees in second extreme of the runway, the south
    longr = airport.longS; %degrees in second extreme of the runway, longitude
    yGSr  = -airport.yGSr;
    %At least in KSEA, glideslope is at the right going to north and at the
    %left going to south
    altgs = airport.zGSaN;
    SN    = -1;    %it indicates that it doesn't go from South to North
else
    latr  = airport.latN; %degrees in second extreme of the runway, the north
    longr = airport.longN; %degrees in second extreme of the runway, longitude
    altgs = airport.zGSaS;
    SN    = 1;    %it indicates that it goes from South to North
end

%% Runway in feet
xr = [0,incrlat*convlat]; %points that define the runway
yr = [0,incrlong*convlong]; %points that define the
zr = [0,0]; %this would be elevation if it was in MSL instead of AGL
delta = abs(atand(diff(yr)/diff(xr)));

%% Aircraft in feet
```

```
xa = abs(lat-latr)*convlat; %of the aircraft
ya = SN*(long-longr)*convlong; %of the aircraft
za = zeros(1,length(altmsl)); %of the aircraft

runw = 0;                   %it starts in the air, no in the runway
for ii=1:length(altmsl)
    if altagl(ii)>15 && runw==0
        za(ii) = altmsl(ii)-altgs; %to avoid the interference of buildings
    else
        za(ii) = altagl(ii);     %because not having the runway profile
        runw   = 1;              %this is needed, because the runway
        %profile varies and without this paraeter, it would return the take
        %the first value of za instead of this one once it has touchdown.
    end
end

xv = 1:length(xa);

%% Vertical plane with initial point in the GS (glideslope)
vdeg    = sort([3-incrthetaG,3+incrthetaG]); %limit of vertical lanes
%sort orders the elements of a vector in ascending order

xGS     = xr(2)-xGSr*cosd(delta)+yGSr*sind(delta); %Location of GS

vdegair = atand(za./(xa-xGS)); %degrees of aircraft in vertical plane as to GS

%VERTICAL LANES
Vlanev = zeros(1,length(vdegair));
for ii = 1:length(vdegair) %for every component of vdegair
    jj = 1;
    found = 0;
    while jj<=length(vdeg)-1 && found==0 %for every component of possible values
        if jj==1 && vdegair(ii)<vdeg(1)
            Vlanev(ii) = 1; %in the first lane, the lowest
            found      = 1; %to stop searching and save time and operations
        elseif vdegair(ii)>=vdeg(jj) && vdegair(ii)<=vdeg(jj+1)
            Vlanev(ii) = jj+1;
            found      = 1; %to stop searching and save time and operations
        elseif jj==length(vdeg)-1 && vdegair(ii)>vdeg(jj+1)
            Vlanev(ii) = jj+2; %in the last lane, the highest
            found      = 1; %to stop searching and save time and operations
        end
        jj = jj+1; %update index
    end
end

%Height of transition (from approach to flare)
ht = 15; %feet AGL. According to the FAA, 10-20 feet AGL
xh = [0,xa(end)+2];
zh = [ht,ht];

Vlanev(za<ht) = length(vdeg)/2+1; %it is is under ht, it is practically on
% the runway, the central vertical lane
% approximation

%% Horizontal plane changes with initial point in the LOC (localizer)
hdeg    = sort([-incrthetaL,incrthetaL]); %limit of horizontal lanes
```

```
%sort orders the elements of a vector in ascending order

xLOC = xr(1)-xLOCr*cosd(delta); %Location of LOC (negative, behind
%the origin point in the near extreme of the runway
yLOC = yr(1)-yLOCr;           %In general, it is zero, because of being aligned

hdegair = atand((ya-yLOC)./(xa-xLOC)); %degrees of aircraft in horizontal
%plane according to GS

%HORIZONTAL LANES
Hlanev = zeros(1,length(hdegair));
for ii = 1:length(hdegair) %for every component of hdegair
    jj = 1;
    found = 0;
    while jj<=length(vdeg)-1 && found==0 %for every component of possible values
        if jj==1 && hdegair(ii)<hdeg(1)
            Hlanev(ii) = 1; %in the first lane, the most to the left
            found     = 1; %to stop searching and save time and operations
        elseif hdegair(ii)>=hdeg(jj) && hdegair(ii)<=hdeg(jj+1)
            Hlanev(ii) = jj+1;
            found     = 1; %to stop searching and save time and operations
        elseif hdegair(ii)>=hdeg(jj+1)
            Hlanev(ii) = jj+2; %in the last lane, the most to the right
            found     = 1; %to stop searching and save time and operations
        end
        jj = jj+1; %update index
    end
end
```

9.3.3 landingcoordstostate.m

Inside the definition of the world where the project takes form, it is needed to transform parameters to a state that can be controlled by the main software of this toolbox.

Code 9.7 landingcoordstostate.m.

```
function s = landingcoordstostate(x,Hlane,Vlane,pitch_r,roll_r,head_r,...
    T_r,V_,mdp_params)
% Utility function for converting coordinates to state indices.
%x: position in that step of time, x=1 in the first iteration (latitude)
%pitch_r: coordinate of ratio of deflection of elevator discretised
%roll_r: coordinate of ratio of deflection of ailerons discretised
%head_r: coordinate of ratio of deflection of rudder discretised
%T_r: coordinate of ratio of throttle discretised
%V_: indicated airspeed converted to coordinate
%mdp_params.Hlanes: number of possible horizontal lanes (longitude)
%mdp_params.Vlanes: number of possible vertical lanes (altitude)
%mdp_params.pitchrs: number of possible pitch ratios
%mdp_params.rollrs: number of possible roll ratios
%mdp_params.headrs: number of possible heading ratios
%mdp_params.Trs: number of possible throttle ratios
%mdp_params.Vs: number of possible indicated airspeed

%To a better understanding of why these expressions have been obtained like
%that, see the memory of the project.
```

```
s = (x-1)*mdp_params.Hlanes*mdp_params.Vlanes*mdp_params.pitchrs*...
        mdp_params.rollrs*mdp_params.headrs*mdp_params.Trs*mdp_params.Vs...
    + (Hlane-1)*mdp_params.Vlanes*mdp_params.pitchrs*mdp_params.rollrs*...
        mdp_params.headrs*mdp_params.Trs*mdp_params.Vs...
    + (Vlane-1)*mdp_params.pitchrs*mdp_params.rollrs*mdp_params.headrs*...
        mdp_params.Trs*mdp_params.Vs...
    + (pitch_r-1)*mdp_params.rollrs*mdp_params.headrs*mdp_params.Trs*...
        mdp_params.Vs...
    + (roll_r-1)*mdp_params.headrs*mdp_params.Trs*mdp_params.Vs...
    + (head_r-1)*mdp_params.Trs*mdp_params.Vs...
    + (T_r-1)*mdp_params.Vs...
    + (V_);
end
```

9.4 IRL Toolkit

These are the codes for the created environment that are executed together with the IRL Toolkit.

There are two unfinished functions: *landingfeatures.m* and *landingbuild.m* which are not included here but they can be found in the attachments.

9.4.1 landingcoordstostate.m

It is the same as Subsection 9.3.3

9.4.2 landingstatetocoords

Inside the definition of the world where the project takes form, it is needed to transform state into parameters to get a vision of the main characteristics of the object in each step of time.

Code 9.8 landingstatetocoords.m.

```
function [x,Hlane,Vlane,pitch_r,roll_r,head_r,T_r,V_] = ...
    landingstatetocoords(s,mdp_params)
% Utility function for coverting state indices to coordinates.
% See landingcoordstostate to understand how s has been created from
% coordinates. It is easier to understand in that way and then try with this
%x: position in that step of time, x=1 in the first iteration (latitude)
%pitch_r: coordinate of ratio of deflection of elevator discretised
%roll_r: coordinate of ratio of deflection of ailerons discretised
%head_r: coordinate of ratio of deflection of rudder discretised
%T_r: coordinate of ratio of throttle discretised
%V_: indicated airspeed converted to coordinate
%mdp_params.Hlanes: number of possible horizontal lanes (longitude)
%mdp_params.Vlanes: number of possible vertical lanes (altitude)
%mdp_params.pitchrs: number of possible pitch ratios
%mdp_params.rollrs: number of possible roll ratios
%mdp_params.headrs: number of possible heading ratios
%mdp_params.Trs: number of possible throttle ratios
%mdp_params.Vs: number of possible indicated airspeed

x      = ceil(s/(mdp_params.Hlanes*mdp_params.Vlanes*mdp_params.pitchrs*...
    mdp_params.rollrs*mdp_params.headrs*mdp_params.Trs*mdp_params.Vs));
s      = s - (x-1)*(mdp_params.Hlanes*mdp_params.Vlanes*mdp_params.pitchrs*...
    mdp_params.rollrs*mdp_params.headrs*mdp_params.Trs*mdp_params.Vs);
Hlane  = ceil(s/(mdp_params.Vlanes*mdp_params.pitchrs*mdp_params.rollrs*...
    mdp_params.headrs*mdp_params.Trs*mdp_params.Vs));
```

```
s      = s -(Hlane-1)*(mdp_params.Vlanes*mdp_params.pitchrs*...
    mdp_params.rollrs*mdp_params.headrs*mdp_params.Trs*mdp_params.Vs);
Vlane  = ceil(s/(mdp_params.pitchrs*mdp_params.rollrs*mdp_params.headrs*...
    mdp_params.Trs*mdp_params.Vs));
s      = s - (Vlane-1)*(mdp_params.pitchrs*mdp_params.rollrs*...
    mdp_params.headrs*mdp_params.Trs*mdp_params.Vs);
pitch_r = ceil(s/(mdp_params.rollrs*mdp_params.headrs*mdp_params.Trs*...
    mdp_params.Vs));
s      = s - (pitch_r*-1)*(mdp_params.rollrs*mdp_params.headrs*...
    mdp_params.Trs*mdp_params.Vs);
roll_r = ceil(s/(mdp_params.headrs*mdp_params.Trs*mdp_params.Vs));
s      = s - (roll_r*-1)*(mdp_params.headrs*mdp_params.Trs*mdp_params.Vs);
head_r = ceil(s/(mdp_params.Trs*mdp_params.Vs));
s      = s - (head_r*-1)*(mdp_params.Trs*mdp_params.Vs);
T_r    = ceil(s/(mdp_params.Vs));
s      = s - (T_r*-1)*(mdp_params.Vs);
V_     = s;
end
```

9.4.3 coordstoratios.m

This function is needed to convert the coordinates (defined by the state and action indicated by the IRL algorithm) to ratios that will be introduced in X-Plane.

Code 9.9 coordstoratio.m.

```
function [pitchr,rollr,headr,Tr,V] = ...
    coordstoratios(pitch_r,roll_r,head_r,T_r,V_,ini,last,divisions,inis,lasts)
%Function for converting from components of the possible values to ratios
%and speed

%% Coords to ratios discretised to the possible ones
%Generalizing for a number of possible pitch_r such as pitchrs and [ini,last]:
%ini*a+b=1      %the first possible value of pitch_r
%last*a+b=pitchrs %the last possible value of pitch_r
%Subtracting: (last-ini)*a=pitchrs-1, a=(pitchrs-1)/(last-ini) and therefore,
%b=1-a*ini; b=1-ini*(pitchrs-1)/(last-ini)-, b=(last-ini*pitchrs)/(last-ini);
%It is going to be considered the same number of possible ratios of all
%variables: pitchrs=rollrs=headrs=Trs=divisions
%Now the conversion is the inverse, extracting the ratios discretised

%The last abstraction would be converting and having different ranges of
%each variable or/and different number of possible values of each variable
%It is a future improvement

a = (divisions-1)/(last-ini);
b = 1-ini*(divisions-1)/(last-ini);

pitchr = (pitch_r-b)/a;
rollr  = (roll_r-b)/a;
headr  = (head_r-b)/a;
Tr     = (T_r-b)/a;

%% Speed: an approximation, the important parameters are the previous actions
%X-Plane will present more accurate data after implementing the actions
vects = (inis:(lasts-inis)/(divisions-1):lasts);
```

```
V = zeros(1:length(length(V_)));
%It is going to be associated the inferior value always in this case
for ii = 1:length(V_) %for every component of V_
    V(ii)=vects(V_(ii));
end
end
```

9.4.4 landingdefaultparams.m

Code 9.10 landingdefaultparams.m.

```
function mdp_params = landingdefaultparams(mdp_params)
% Fill in default parameters for the landingworld example.

ini       = -1; %initial point of the range
last      =  1; %last point of the range
divisions = 21; %number of possible values in the range
%201 with range [-1,1] and step of 0.01
%Another good option would be 21: [-1,1] and step of 0.1
%This first method allows to discretise variables to have less values. It
%is useful if the discretisation is approximately logical and simplifies
%the result (a correct combination of ini, last, divisions; according to
%the original values, such as the two options presented).

inis      =   0; %initial value of the range of airspeed indicated
lasts     = 200; %last value of the range of airspeed indicated
%for speed, the initial value doesn't reach 200knots and the last is 0knots
%It is considered the same number of divisions, but this divisions are now
%limits of the segments
%This second method is useful to take directly the component of the possible
%values of a parameter. The difference is that with that, the variable
%isn't discretised. While the first one gives divisions as the number of
%possible values, in this one, they are one possible value less due to be
%between two consecutive values. (SEE the paper of the project to
%understand it better).
%If the values outside the range are also considered, the number of
%components of possible values are 2 more than the provided by the
%first method. This is applied for the horizontal and vertical lanes
%related to longitude and altitude.

% AIRPORT
%If in the airport there aren't glideslope or localizer, set the data
%of where they would be optimally: the point to touchdown and the point
%where to locate the origin point of horizontal guide

%Information of the north extreme
airport.rWNS = '16R';     %name of runway from North to South
airport.latN = 47.46384; %latitude
airport.longN = -122.31785; %longitude
airport.elevN = 330.33;  %altitude MSL, it isn't used
airport.zGSaN = 351.79;  %altitude MSL glideslope in north

%Information of the south extreme
airport.rwSN = '34L';     %name of runway from South to North
airport.latS = 47.44056; %latitude
airport.longS = -122.31808; %longitude
airport.elevS = 353;      %altitude MSL, it isn't used
airport.zGSaS = 359.86;   %altitude MSL glideslope in south

%Runway given by two points in its central line:
airport.lr = 8500; %the longitudinal length of runway
airport.wr = 150; %width of runway

%Localizer
airport.xLOCr = 800; %distance between the extreme of runway and the LOC
airport.yLOCr = 0; %distance between the axis of runway and the LOC
airport.zLOC = 0;  %altitude AGL
airport.incrthetaL = [1.5,3]; %degrees from the central line of the cone(s)
% try to put a difference such as the space created is equally distributed
% between cones, for example: incrthetaL=[1,3] gives an interior cone of 2
% degrees and a exterior one of 6 degrees in total
%[1.5,3] is what gives the most similar result to what X-Plane uses, and
%determines 5 lanes: under the lowest line, between the lower lines of
%the cones, the central zone, between the upper lines of the cones and
%above the uppest line

%Glideslope
airport.xGSr = 793; %distance between the extreme of runway and the GS
airport.yGSr = 175; %distance between the axis of runway and the GS
airport.zGS = 0;  %altitude AGL
airport.incrthetaG = [0.25,0.7]; %degrees from the central line of the cone(s)
% try to put a difference such as the space created is equally distributed
% between cones, for example: incrthetaG=[1,3] gives an interior cone of 2
% degrees and a exterior one of 6 degrees in total
%[0.25, 0.7] is what gives the most similar result to what X-Plane uses, and
%determines 5 lanes: under the lowest line, between the lower lines of
%the cones, the central zone, between the upper lines of the cones and
%above the uppest line

Hlanes = incrthetaL*2+1;
Vlanes = incrthetaG*2+1;

% Create default parameters.
default_params = struct(...
    'seed',0,...          %initialization for random seed
    'ini',ini,...         %initial point of the range of all ratios
    'last',last,...       %last point of the range of all ratios
    'inis',inis,...       %initial point of the range of indicated airspeed
    'lasts',lasts,...     %last point of the range of indicated airspeed
    'length',50,...       %number of iterations considered in a segment
    'Hlanes',Hlanes,...   %number of possible horizontal lanes (longitude)
    'Vlanes',Vlanes,...   %number of possible vertical lanes (altitude)
    'pitchrs',divisions,... %number of possible pitch ratios
    'rollrs',divisions,... %number of possible roll ratios
    'headrs',divisions,... %number of possible heading ratios
    'Trs',divisions,...   %number of possible T ratios
    'Vs',divisions-1,...  %number of possible indicated airspeeds
    'airport',airport,... %data about runway, glideslope and localizer
    'continuous',0,...    %discrete (0) or continuous (1)
    'determinism',1.0,... %probability of correct transition
    'discount',0.9);      %temporal discount factor to use
```

```
% Set parameters.
mdp_params = filldefaultparams(mdp_params,default_params);
```

Attachments

This chapter contains the management of attachments. While this memory can be found in its digital version in a file of extension .pdf in the general, the rest of files are organized in folders:

- *X-Plane software*, with the installation files of demo versions X-Plane 10 and X-Plane 11, as well as the aircraft employed in this project: A380.
- *XPlaneConnect-master*, which comprehends:
  - Folders of the toolkit in different languages and files for testing the different contents in \ TestScripts. For the present document only MATLAB is used.
    - Inside MATLAB, some simple examples are found, the inner functions described in Section 5.2 and the folder landing with all what has been developed for this project.
      - *Landing* specifically contains the SERP3nmiapp.m, Original data taken from X-Plane, which has got several modifications explained in a README inside the folder, to become the data that is in Correct data. This is the one copied into *landing_demos* folder of IRL Toolkit and which the machine is supposed to process in order to learn the reward function *R*. The last file is *SERP3nmiappML.m* which receives in real time the data extracted from X-Plane and implements in the simulator the actions generated by the IRL algorithm after processing the data with the learn reward and policy functions.
- *IRL-Toolkit-master* which has been deeply explained in Chapter 7.
- *Papers*, with all the important previous investigations, and interesting slides and lectures of Stanford and Berkeley.

The general perspective is in Figure 9.1, and the respective detailed views of the folders *XPlaneConnect-master* and *IRL-Toolkit-master* are in Chapter 5 and Chapter 7.

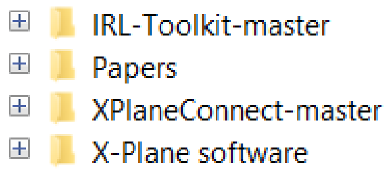

IRL-Toolkit-master
Papers
XPlaneConnect-master
X-Plane software

Figure 9.1 General directory of attachments.

List of Figures

1.1 Effect of flaps on the landing point [1] 2
1.2 Changing angle of attack during flare [1] 3
1.3 GBAS [7] 4
1.4 Horizontal and vertical view of ILS [4] 5
1.5 Localizer aerial, glide slope installations and ILS indicator [4] 5
1.6 AI vs ML vs DL [13] 6
1.7 Procedure in blocks 7

3.1 X-Plane 11 logo [31] 11
3.2 X-Plane 10 logo [32] 11
3.3 MATLAB logo [33] 12
3.4 Tools in blocks 12

4.1 First selection 13
4.2 Loading X-Plane 14
4.3 QuickFlight Setup 14
4.4 Aerial image of Seattle Tacoma airport in simulator 15
4.5 Aerial image of Seattle Tacoma airport [41] 15
4.6 Diagram of Seattle Tacoma airport [43] 17
4.7 Adding A380 18
4.8 Airbus A380 [48] 18
4.9 Airbus A380 from X-Plane 18
4.10 Second selection 19
4.11 Cockpit in full screen 19
4.12 Cockpit in half screen 20
4.13 X-Plane information 21
4.14 Aids about instructions in cockpit 21
4.15 Local map 22
4.16 X-Plane dataset 23
4.17 Operations and Warnings 24
4.18 3D cockpit 25
4.19 Real A380 cockpit from a 360° view in [49] 25
4.20 Enable/Disable Plugins 26
4.21 Parts of the cockpit [50] 27
4.22 Autopilots controls and modes 27
4.23 Navigation screens 28
4.24 General characteristics and landing gear indicators 29
4.25 FMC, engine, throttle and communications 30
4.26 ILS of 16R runway 31
4.27 Glideslope and localizer installations 16R 31
4.28 Key association 32

5.1 Folder to be copied and where to place it 34
5.2 Inner functions, those relevant to the project circled in red 35
5.3 Developed code and folder of iterations 40

7.1 cvx_setup localization 46
7.2 Directory of IRL Toolkit 47
7.3 Extra files of IRL Toolkit directory 51
7.4 Vertical and horizontal views 56
7.5 ILS with localizer and glideslope according to [8] 57
7.6 XPlane representation in Local Map of an aircraft landing 58
7.7 Landing directory 60
7.8 States in a highway 61
7.9 Horizontal and vertical lanes 62

9.1 General directory of attachments 97

List of Tables

1.1 ILS Categories [8] 6

5.1 First iterations 40

6.1 Differences between RL and IRL 41

7.1 Information about 16R. Landing north $\rightarrow$ south 52
7.2 Information about 34L. Landing south $\rightarrow$ north 53
7.3 Percentage of development of the different items in the project 65

List of Codes

5.1	SERP3nmiapp	36
7.1	Setup of cvx	45
9.1	SERP3nmiapp.m	69
9.2	landing.m	75
9.3	airportdata.m	80
9.4	XPC_IRL_converter.m	81
9.5	ratiostocoords.m	86
9.6	degreestocoords.m	88
9.7	landingcoordstostate.m	91
9.8	landingstatetocoords.m	92
9.9	coordstoratio.m	93
9.10	landingdefaultparams.m	94

Bibliography

[1] FAA, "Chapter 8: Approaches and Landings," in *Airplane Flying Handbook*, March 2016. [Online]. Available: https://www.faa.gov/regulations_policies/handbooks_manuals/aviation/airplane_handbook/media/10_afh_ch8.pdf

[2] "Seattle-Tacoma International Airport," September 2018. [Online]. Available: https://www.faa.gov/nextgen/snapshots/airport/?locationId=45

[3] A. Helfrick, *Principles of Avionics*, 9th ed. Avionics Communications, 2015.

[4] F. Colodro, "Unit 3 Air navigation Aids - Avionics and Navigation Systems," November 2018. [Online]. Available: https://www.dinel.us.es/docencia/docencia.php?c=3&d=1&titulacion=12&asignatura=105&opcion=12

[5] "Ground-Based Navigation - Instrument Landing System," December 2016, accessed on 24-04-2019. [Online]. Available: https://www.faa.gov/about/office_org/headquarters_offices/ato/service_units/techops/navservices/gbng/ils/

[6] "Satellite Navigation - Wide Area Augmentation System (WAAS)," May 2019. [Online]. Available: https://www.faa.gov/about/office_org/headquarters_offices/ato/service_units/techops/navservices/gnss/waas/

[7] "Satellite Navigation — Ground Based Augmentation System (GBAS)," April 2018. [Online]. Available: https://www.faa.gov/about/office_org/headquarters_offices/ato/service_units/techops/navservices/gnss/laas/

[8] U. S. D. of Transportation Federal Aviation Administration, "Aeronautical Information Manual," October 2017. [Online]. Available: https://www.faa.gov/air_traffic/publications/media/AIM_Basic_dtd_10-12-17.pdf

[9] "Transponder Landing System | ILS for Short Runway Airports | ANPC," January 2012. [Online]. Available: http://www.anpc.com/Transponder-Landing-System/

[10] "Lighting Systems — Runway Visual Range (RVR)," July 2017. [Online]. Available: https://www.faa.gov/about/office_org/headquarters_offices/ato/service_units/techops/navservices/lsg/rvr/

[11] "Commission implementing regulation (EU) No 923/2012 SERA.5005 Visual flight rules," October 2012. [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32012R0923&from=EN

[12] "Code of Federal Regulations Part 91 General Operating and Flight Rules Minimum safe altitudes," January 2010. [Online]. Available: http://rgl.faa.gov/regulatory_and_guidance_library/rgfar.nsf/b4a0cab3e513bb58852566c70067018f/91693c93525de33e862576c100763e31!OpenDocument

[13] Atul, "AI vs Machine Learning vs Deep Learning," April 2019. [Online]. Available: https://www.edureka.co/blog/ai-vs-machine-learning-vs-deep-learning/

[14] M. Ambur, D. N. Mavris, and K. G. Schwartz, “Machine Learning Technologies and Their Applications for Science and Engineering Domains Workshop,” Langley Research Center National Aeronautics and Space Administration (NASA), Tech. Rep., December 2016. [Online]. Available: https://ntrs.nasa.gov/search.jsp?R=20170000679

[15] “Machine Learning What it is and why it matters,” April 2019. [Online]. Available: https://www.sas.com/en_us/insights/analytics/machine-learning.html

[16] H. Li, “Which machine learning algorithm should I use?” April 2017. [Online]. Available: https://blogs.sas.com/content/subconsciousmusings/2017/04/12/machine-learning-algorithm-use/

[17] P. Larrañaga, “Machine Learning in aviation,” July 2013. [Online]. Available: http://cig.fi.upm.es/sites/default/files/com_phocadownload/container/Ml-in-aviation.pdf

[18] O. Hazi, “Final Approach – An Automated Landing System for the X-Plane Flight Simulator,” 2011. [Online]. Available: http://cs229.stanford.edu/proj2011/Hazi-FinalApproach.pdf

[19] E. Gizzi and D. Zabner, “Flying Controlled Trajectories using Linear Approximation Reinforcement Learning,” Tufts University, Medford, USA, Tech. Rep., December 2018. [Online]. Available: https://www.eecs.tufts.edu/~jsinapov/teaching/comp150_RL/reports/Gizzi_Zabner_report.pdf

[20] H. Baomar and P. J. Bentley, “An Intelligent Autopilot System that Learns Piloting Skills from Human Pilots by Imitation,” Dept. of Computer Science, University College London, Gower Street, London, UK, Tech. Rep., June 2016. [Online]. Available: http://www0.cs.ucl.ac.uk/staff/h.baomar/files/RP1.pdf

[21] ——, “An Intelligent Autopilot System that Learns Flight Emergency Procedures by Imitating Human Pilots,” Dept. of Computer Science, University College London, Gower Street, London, UK, Tech. Rep., October 2016. [Online]. Available: http://www0.cs.ucl.ac.uk/staff/h.baomar/files/RP2.pdf

[22] ——, “Autonomous Navigation and Landing of Airliners Using Artificial Neural Networks and Learning by Imitation,” Dept. of Computer Science, University College London, Gower Street, London, UK, Tech. Rep., July 2017. [Online]. Available: http://www0.cs.ucl.ac.uk/staff/h.baomar/files/RP3.pdf

[23] ——, “Autonomous Landing and Go-around of Airliners Under Severe Weather Conditions Using Artificial Neural Networks,” Dept. of Computer Science, University College London, Gower Street, London, UK, Tech. Rep., September 2017. [Online]. Available: http://www0.cs.ucl.ac.uk/staff/h.baomar/files/RP4.pdf

[24] *Learning to Fly by Combining Reinforcement Learning with Behavioural Cloning*. 21 st International Conference on Machine Learning, 2004. [Online]. Available: https://www.icml.cc/imls/conferences/2004/proceedings/papers/187.ps

[25] “Machine Learning CS229 Stanford Syllabus and Course Schedule,” June 2019. [Online]. Available: http://cs229.stanford.edu/syllabus.html

[26] A. Y. Ng and S. Russell, “Algorithms for Inverse Reinforcement Learning,” Computer Science Division, U.C. Berkeley, Berkeley, California, USA, Tech. Rep., 2000. [Online]. Available: https://ai.stanford.edu/~ang/papers/icml00-irl.pdf

[27] P. Abbeel and A. Y. Ng, “Apprenticeship Learning via Inverse Reinforcement Learning,” Computer Science Division, Stanford University, Stanford, California, USA, Tech. Rep., 2004. [Online]. Available: https://ai.stanford.edu/~ang/papers/icml04-apprentice.pdf

[28] S. Levine, Z. Popovic, and V. Koltun, “Nonlinear Inverse Reinforcement Learning with Gaussian Processes,” Computer Science Division, Stanford University, Stanford, California, USA, Tech. Rep., 2011. [Online]. Available: https://graphics.stanford.edu/projects/gpirl/

[29] “Download the free X-Plane 11 demo,” 2019. [Online]. Available: https://www.x-plane.com/desktop/try-it/

[30] “Install X-Plane 10,” 2019. [Online]. Available: https://www.x-plane.com/desktop/try-it/older

[31] "X-Plane-Logo," September 2016. [Online]. Available: https://www.x-plane.com/2016/10/x-plane-11-coming-holiday-season/x-plane-logo/

[32] "X-Plane-10-Logo."

[33] "The MathWorks Logo is an Eigenfunction of the Wave Equation," 2003. [Online]. Available: https://es.mathworks.com/company/newsletters/articles/the-mathworks-logo-is-an-eigenfunction-of-the-wave-equation.html

[34] A. Bittar, "X-Plane Library.zip," September 2014. [Online]. Available: https://es.mathworks.com/matlabcentral/fileexchange/47516-x-plane-library-zip

[35] M. A. Zahana, "Simulink-Xplane10 Communication Via UDP," November 2015. [Online]. Available: https://es.mathworks.com/matlabcentral/fileexchange/47144-simulink-xplane10-communication-via-udp

[36] C. Teubert, J. Watkins, and B. Bole, "X-Plane Connect," June 2018. [Online]. Available: https://github.com/nasa/XPlaneConnect

[37] A. Gosavi, "MATLAB Repository for Reinforcement Learning," 2012. [Online]. Available: http://web.mst.edu/~gosavia/mrrl_website.html

[38] M.-J. Cros, I. Chadès, F. Garcia, and R. Sabbadin, "Markov Decision Processes (MDP) Toolbox," January 2015. [Online]. Available: https://es.mathworks.com/matlabcentral/fileexchange/25786-markov-decision-processes-mdp-toolboxl

[39] P. Fackler, "MDPSolve," May 2017. [Online]. Available: https://github.com/PaulFackler/MDPSolve/

[40] S. Levine, "IRL-Toolkit," 2016. [Online]. Available: https://github.com/vvanirudh/IRL-Toolkit

[41] Jelson, "Aerial KSEA," May 2012. [Online]. Available: https://es.m.wikipedia.org/wiki/Archivo:Aerial_KSEA_May_2012.JPG

[42] "Airport Capacity Profile: Seattle-Tacoma International Airport," 2018. [Online]. Available: https://www.faa.gov/airports/planning_capacity/profiles/media/SEA-Airport-Capacity-Profile-2018.pdf

[43] "Seattle-Tacoma International Airport Diagram," January 2015. [Online]. Available: https://aeronav.faa.gov/d-tpp/1907/00582ad.pdf#nameddest=(SEA)

[44] T. D. Support and S. AIRBUS, "Aircraft characteristics airport and maintenance planning," January 2019. [Online]. Available: https://www.airbus.com/content/dam/corporate-topics/publications/backgrounders/techdata/aircraft_characteristics/Airbus-Aircraft-AC-A380.pdf

[45] "A380 destinations," 2019. [Online]. Available: https://www.iflya380.com/a380-destinations.html

[46] "A380 - Airport Compatibility," 2019. [Online]. Available: https://www.airbus.com/aircraft/passenger-aircraft/a380/airlines-destinations/airport-compatibility.html

[47] "Calculate distance, bearing and more between Latitude/Longitude points." [Online]. Available: https://www.airbus.com/content/dam/corporate-topics/publications/backgrounders/techdata/general-information/Airbus-Commercial-Aircraft-ICAO-ARC-FAA-ADG-App-Cat.pdf

[48] "A380," 2019. [Online]. Available: https://www.airbus.com/aircraft/passenger-aircraft/a380.html

[49] "A380 cockpit virtual visit," 2014. [Online]. Available: https://ccntservice.airbus.com/apps/cockpits/a380/

[50] "A380-800 Flight Deck and Systems Briefing for Pilots ," March 2006. [Online]. Available: http://www.smartcockpit.com/docs/A380_Briefing_For_Pilots_Part%202.pdf

[51] J. Watkins, "XPC Client Reference," June 2015. [Online]. Available: https://github.com/nasa/XPlaneConnect/wiki/XPC-Client-Reference

[52] ——, "Datarefs for X-Plane 1041," October 2015. [Online]. Available: http://www.xsquawkbox.net/xpsdk/docs/DataRefs.html

[53] ——, “Datarefs for X-Plane 1041,” October 2015. [Online]. Available: http://www.xsquawkbox.net/xpsdk/mediawiki/Sim/cockpit/autopilot/autopilot_state

[54] J. Heidecke, “Inverse reinforcement learning part i.”

[55] ——, “On the difference between R(s) and R(s,a),” February 2018. [Online]. Available: https://thinkingwires.com/posts/2018-02-11-reward-function-domain.html

[56] M. Grant and S. Boyd, “CVX: Matlab Software for Disciplined Convex Programming,” December 2018. [Online]. Available: http://cvxr.com/cvx/

[57] S. Levine, Z. Popović, and V. Koltun, “Feature construction for inverse reinforcement learning,” Tech. Rep., 2010. [Online]. Available: https://homes.cs.washington.edu/~zoran/firl.pdf

[58] N. D. Ratliff, D. Silver, and J. A. Bagnell, “Learning to search: Functional gradient techniques for imitation learning,” Tech. Rep., 2009. [Online]. Available: https://www.ri.cmu.edu/pub_files/2009/7/learch.pdf

[59] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey, “Maximum entropy inverse reinforcement learning,” in *Proc. AAAI*, 2008, pp. 1433–1438. [Online]. Available: http://www.cs.cmu.edu/~bziebart/publications/maxentirl-bziebart.pdf

[60] N. D. Ratliff, J. Bagnell, and M. A. Zinkevich, “Maximum margin planning,” 2006. [Online]. Available: http://martin.zinkevich.org/publications/maximummarginplanning.pdf

[61] N. D. Ratliff, J. Bagnell, and J. Chesnutt, “Boosting structured prediction for imitation learning,” 2007. [Online]. Available: https://papers.nips.cc/paper/3154-boosting-structured-prediction-for-imitation-learning.pdf

[62] U. Syed and R. E. Schapire, “A game-theoretic approach to apprenticship learning,” Tech. Rep., 2008. [Online]. Available: https://papers.nips.cc/paper/3293-a-game-theoretic-approach-to-apprenticeship-learning.pdf

[63] K. Dvijotham and E. Todorov, “Inverse optimal control with linearly-solvable mdps,” Tech. Rep., 2010. [Online]. Available: https://homes.cs.washington.edu/~todorov/papers/DvijothamICML10.pdf

Glossary

ADF-NDF	Automatic Directional Finding/Non Directional Beacon
ADIRS	Air Data and Inertial Reference System
AGL	Above Ground Level
AI	Artificial Intelligence
ANN	Artificial Neural Networks
DH	Decision height
DL	Deep Learning
DME	Distance Measuring Equipment
ETSI	Escuela Técnica Superior de Ingeniería
FAA	Federal Aviation Administration
FMC	Flight Management Computer
FMS	Flight Management System
G/S	Glideslope
GBAS	Ground Based Augmentation System
GNSS	Global Navigaton Satellite System
GPIRL	Gaussian Process Inverse Reinforcement Learning
GPS	Global Position System
ILS	Instrument Landing System
IMC	Instrument Meteorological Condition
IOC	Inverse Optimal Control
IRL	Inverse Reinforcement Learning
KSEA	Seattle-Tacoma International Airport (ICAO code)
LAAS	Local Area Augmentation System
LOC	Localizer
MDP	Markov Decision Process
ML	Machine Learning

MLS	Microwave Landing System
NASA	National Aeronautics and Space Administration
RL	Reinforcement Learning
RNAV	Area Navigation
RNP	Required Navigation Performance
RVR	Runway Visual Range
TLS	Transponder Landing System
U.S. NAS	United States National Airspace System
UDP	User Datagram Protocol
US	Universidad de Sevilla
VHF	Very High Frequency
VMC	Visual Meteorogical Conditions
VNAV	Vertical Navigation
VOR	Very High Frequency Omnidirectional Range
WAAS	Wide Area Augmentation System
WOW	Weight On Wheels
XPC	X-Plane Connect