Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

TREBALL FINAL DE GRAU

TITLE: Analysis with modelling and simulation of the potential benefits of Automation of ground controller functions.

DEGREE: Bachelor's degree in Aerospace Systems Engineering and Telecommunications Systems Engineering

AUTHOR: **Pol Borbonet Legares**

DIRECTORS:
César Trapote Barreira
Miguel Valero García

DATE: June 17 th 2022

Títol: Anàlisi amb modelització i simulació dels beneficis potencials de l'automatització de les funcions de controlador de terra.

Autor: Pol Borbonet Legares

Director: César Trapote Barreira, Miguel Valero García

Data: 17 de juny del 2022

Resum

La demanda aeronàutica no para de créixer des de fa molts anys, però la capacitat en les infraestructures aeroportuàries no pot seguir el mateix ritme. Recentment estem veient nous projectes d'ampliacions dels principals aeroports d'Europa per augmentar la seva capacitat i mantenir un nivell de servei excel·lent.

Per que Barcelona pugui mantenir el nivell dels principals aeroports europeus com Heathrow, Paris Charles de Gaulle, Amsterdam Schiphol entre altres, no és suficient en millorar simplement les infraestructures, sinó que la operativa és, fins i tot, més important. Aquesta millora es pot realitzar a partir de les ultimes tecnologies (automatització, intel·ligència artificial, ...).

El propòsit d'aquest treball és implementar un simulador que automatitzi les funcions que té un controlador de terra. Aquestes funcions són guiar i evitar conflictes entre la porta d'embarcament i les pistes i viceversa. A la vegada, mitjançant la ultilització d'aquest simulador els errors humans desapareixeran i el temps en rodadura s'optimitzarà.

Per dur a terme aquest treball, es crearà un simulador desde el principi utilitzant un escenari conegut com l'aeroport de Barcelona – El Prat i unes dades obtingudes a partir de pàgines web com FlightRadar24 o FlightStats. Durant les simulacions el codi s'ha anat modificant cada cop que algun avió no realitzava el moviment desitjat, i posteriorment es tornava a simular. Un cop transcorregut les 24 hores en les que s'obtenien les dades dels moviments, un seguit de variables permeten analitzar el comportament del simulador

Finalment, es pot dir que el programa no genera delay, i que millora substancialment la operativa del aeroport minimitzant les cues prèvies a la pista d'enlairament. També és veritat que no prodeuix la mateixa millora en les sortides que en les arribades. Aquest simulador és un primer pas per l'automatització de les funcions dels controladors aeris, demostrant que té beneficis tant econòmics com operacionals.

Title: Analysis with modelling and simulation of the potential benefits of Automation of ground controller functions.

Author: Pol Borbonet Legares

Director: César Trapote Barreira, Miguel Valero García

Date: June 17 th 2022

Overview

Aviation demand has been growing steadily for many years, but airport infrastructure capacity cannot keep pace. We have recently seen new expansion projects at Europe's main airports to increase their capacity and maintain an excellent level of service.

For Barcelona to maintain the level of major European airports such as Heathrow, Paris Charles de Gaulle, Amsterdam Schiphol and others, it is not enough to simply improve the infrastructure, but the operations are even more important. This improvement can be made using the latest technologies (automation, artificial intelligence,...).

The purpose of this work is to implement a simulator that automates the functions of a ground controller. These functions are to guide and avoid conflicts between the gate and runways and vice versa. At the same time, by using this simulator, human errors will disappear and taxiing time will be optimised.

To carry out this work, a simulator was created from scratch using a known scenario such as Barcelona - El Prat airport and data obtained from websites such as FlightRadar24 or FlightStats. During the simulations, the code was modified each time an aircraft did not perform the desired movement, and then simulated again. Once the 24 hours of movement data had elapsed, a series of variables allowed us to analyse the behaviour of the simulator.

Finally, it can be said that the programme does not generate delay, and that it substantially improves airport operations by minimising queues before the runway. It is also true that it does not produce the same improvement in departures as in arrivals. This simulator is a first step towards the automation of air traffic controllers' functions, demonstrating that it has both economic and operational benefits.

Acknowledgements:

To my family, for their support throughout this work

For Arnau, Eric and Inés especially but also for my friends who have helped me from the first day and have been there in the best and worst moments.

To César Trapote and Miguel Valero, without their knowledge and dedication this work would not be possible.

ÍNDEX

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

CONCEPT	DEFINITION
A-CDM	Airport Collaborative Decision Making
AENA	Aeropuertos Españoles y Navegación Aérea
AIBT	Actual In-Block Time
AIP	Aeronautical Information Publication
ALDT	Actual Landing Time
AOBT	Actual Off-Block Time
APU	Auxiliary Power Unit
ARC	Airport Research Center
ATC	Air Traffic Control
ATFM	Air Traffic Flow Management
ATM	Air Traffic Management
ATOT	Actual Take-Off Time
CO	Carbon Monoxide
CO2	Carbon Dioxide
EGTS	Electric Green Taxiing System
ETA	Estimated Time of Arrival
ETD	Estimated Time of Departure
FIS	Flight Information Service
FOD	Foreign Object Damage
LoS	Level of Service
NOX	Nitrogen Oxide
RTS	Remote Tower Services
SMR	Surface Movement Radar
Surface CDM	Surface Collaborative Decision Making
TBFM	Time Based Flow Management
TFDM	Terminal Flight Data Management
TFMS	Traffic Flow Management System
TMC	Terminal Control
TRACON	Terminal Radar Approach Control
TTP	TFDM Terminal Publication Service
URAM	Uniformly Rectilinear Accelerated Motion

INTRODUCTION

Nowadays, with the increase in aeronautical demand, it is more usual to find an airport at the limit of its capacity, and at peak hours the delays are remarkable, leading ground controllers to the limit of their capacities. In this work he presented a possible solution to reduce the workload or replace this type of controllers, trying to improve operations by modelling and simulating an airport environment that maximises the time an aircraft spends taxiing and finding the optimal route from the runway to the stand and vice versa.

This work is of great importance, in the example of Barcelona - El Prat airport, this software could be used to manage the movement of aircraft on the apron and thus possibly avoid any new infrastructure (runway extension or new terminal). In addition, a very critical feature of aviation is the pollution generated, especially when they are at an airport, and for this reason finding the optimal route without queues at the threshold of the runway, in other words, the shortest route and minimising the waiting time, could be a great step forward to have a more sustainable and environmentally friendly airport. In addition, apart from the implementation that can be achieved by replacing ground controllers, it can also be used to test possible modifications (creating a new runway, taxiway or terminal, changing the direction of taxiways,...) at an airport in order to visualise how it can affect it. The scenario used to test the programme is Barcelona-El Prat airport, using a 2D model.

The report consists of 10 chapters, the first two of which provide some theoretical background on controllers and Barcelona-El Prat airport. The third chapter explains the objectives and the work plan. In the next chapter you can see examples of simulations previously used in the airport field. The 5th chapter explains the characteristics of the programme created (language, resources used, classes, main functions, etc.). The 6th chapter explains the design of the simulator and how to make it work. The next chapter comments on the different phases the simulator has gone through until the final result is achieved. The 8th chapter analyses the results obtained from the simulator. And finally, the last two chapters are the possible implementations in a simulator.

My main motivation of the work is to do a useful and feasible project that could imply an improvement in aviation. Due to the speciality of airports and recent news about the enlargement of the Barcelona - El Prat airport, I wanted to make my contribution by trying to find a solution based on a simulation using mathematical models.

CHAPTER 1. Air Traffic Controller

This chapter will explain what an air traffic controller is, what types there are and what functions each of them has. It also explains which are the current tools used by the different air traffic controllers and finally the chapter ends with the explanation of some tools that are in the development phase that in the future will help the performance of the air traffic controllers tasks.

1.1 What is an Air Traffic Controller and what does they do?

1.1.1 What is an Air Traffic Controller

The Air Traffic Controllers (ATC) are the responsibles to ensure safety applying separation and avoiding potential conflicts between aircrafts while maintaining an orderly and expeditious flow of air traffic. Controllers apply separation rules in order to keep all aircraft within a safe distance through their assigned airspace or ground zone.

The controllers are located on the ground in different buildings depending on the function of each one. Each building has unique characteristics so that the controllers can carry out their tasks in the simplest and most effective way possible, as well as being equipped with various technological tools for the performance of their work.

The first reference to air traffic controllers is in 1920, at Cryodon airport, where the concept of the aerodrome control tower was introduced. This service provided basic information to pilots, such as traffic, weather and location.

Later on, in the United States, after a series of collisions in the air, different airlines got together to start monitoring their own flights. In 1935, in Newark, the first air control centre was created, in order to supervise together their air routes. The following year, two more centres were opened in Cleveland and Chicago. Later, when the Department of Commerce took over control of operations, eight more centres were opened covering the entire US airspace, triggered mainly by the 1956 collision in the Grand Canyon that killed 128 people. Today the FAA operates 22 air route traffic control centres. In 1960, countries such as Britain, France and Germany created Eurocontrol, with the intention of merging their airspaces.

At that time the tools that the air traffic controllers had were very rudimentary, where they wrote down the positions of the planes on a blackboard and located them on a map to avoid collisions in the air. Later in the 1950s, thanks to the invention of radar and the improvement of ground-to-air telecommunications,

this led to an improvement in safety and above all efficiency, as aircraft could be placed closer together without the danger of collisions.

1.1.2 Types of Air Traffic Controllers

There are different types of ATC's, depending on its function and area of action.

1.1.2.1 En route controllers

The en route controllers, also known as center controllers, provide service to aircraft in the airways between airports, usually above 17.000 feet. According to ICAO Annex 11 - Air traffic Services (see in [3]), an airway is a control area or portion thereof established in the form of a corridor.

They work in facilities called air traffic control centers, that are more commonly said "centers", which do not have to be close to an airport infrastructure. They use radar technology to keep track of each plane that they are responsible. In Fig. 1.1, one can observe that they work in semi-darkness due to the radar equipment.

Fig. 1.1 Air traffic control center

Each center is responsible to provide air traffic control service to controlled flights in control areas under its jurisdiction. These control areas are divided into smaller sections, of which a controller will be responsible.

The main decisions that the en route controllers make are adjusting the path of an aircraft to ensure safety by avoiding collisions, conflicts and preventing airborne traffic jams. It's important to remark that they direct the aircraft for the most of the time. Also, they are responsible to "hand off[1]" the aircraft to the next area control center when it reaches the boundary.

1.1.2.2 Approach controllers

Many airports have a radar control facility associated with the airport. This facility called Approach or terminal control (TMC) is also referred in the U.S as terminal radar approach control (TRACON). These types of controllers handle the traffic in a 30-to-50 nautical mile radius from the airport. They have the responsibility of ensuring ATC Services to arrivals, departures and overflights.

The main decisions that the Approach controllers make are to manage the air traffic flow, "hand off" the airplane to the next appropriate control facility, such as control tower or en route center, and that during the hand off the airplane is at the right altitude and has the correct rate of climb/descent.

1.1.2.3 Tower controllers

The Tower controllers of an airport work from the control tower from that airport. They are responsible for the active runway surfaces. The radar Technology make the job easier in bad weather with low visibility, however if visibility allows it they use visual observation to carry out their work due to the strategic position that is the control tower.

[1] An action taken to transfer the radar identification of an aircraft from one controller to another controller if the aircraft will enter the receiving controller's airspace and radio communications with the aircraft will be transferred

Fig. 1.2 Josep Tarradellas Barcelona–El Prat Airport Control Tower

The area of influence of these controllers is approximately a 5-mile radius, but may vary depending on the flow of each airport.

The main function of these controllers is the takeoff and landing instructions and clearances, as they try to maximize the runway capacity. They may also authorize aircraft or vehicle movements on or across the runway. If controllers see that the safety of an aircraft is being breached, they can cancel the takeoff or landing by instructing a "go-around[2]".

1.1.2.4 Ground controllers

The ground control is responsible for the airport “movement areas”. The movement areas include all taxiways, inactive runways and holding areas. Any aircraft, vehicle or person working in these areas is required to have ground control.

The ground controllers need to have good synchronization and communication with the tower controllers to maintain an acceptable traffic flow, hence these two types of controllers are situated in the same building, the control tower of the airport. Depending on the airport traffic flow, some airports can merge ground and rower control.

[2] Aborted landing of an aircraft that is on final approach or has already touched down

Most of the main airports have the latest technology to help the controller to do their work more effectively, the ground controllers use the Surface Movement Radar (SMR) that displays aircraft on the ground.

The main instructions of ground controllers are taxiing, authorization of aircraft/vehicle movement and starting push-back issues. In addition, ground controllers hand off the aircraft to the tower control when they reach the takeoff runway.

1.2 Air traffic Controller tools

1.2.1 Current

Nowadays, air traffic controllers have a variety of tools to facilitate their work. As the years have gone by, these tools have been modernised to the point of improving the capacity of airspace or an airport. These different tools that controllers currently benefit from are explained below.

1.2.1.1 Flight Progress Strips

Eurocontrol (see in [5]) defines the Flight Progress Strips, as: “Paper strip containing planned and current flight plan data for a specific flight, made available on an flight progress board for use by air traffic controllers in the provision of ATS”.

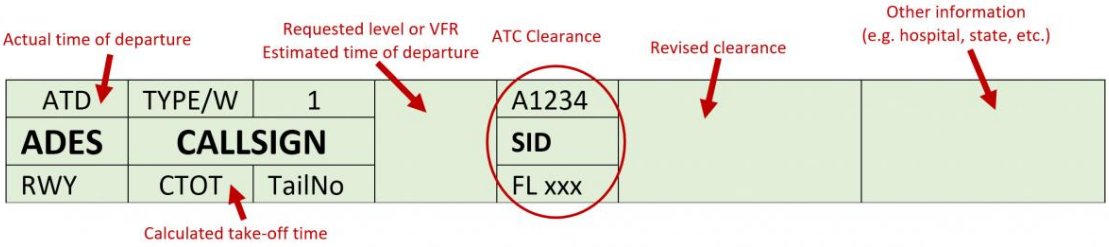


Fig. 1.3 Flight Progress Strip [5]

A Flight Progress Strip presents all the important information about an airplane divided in different fields such as aircraft identification, type and transponder code among others. Besides, Flight Progress Strips can be used in different background colours for each purpose, for example green for departing or blue for arriving traffic. Furthermore, there are different areas to put the Flight Progress Strips which each one defines different traffic situations, for example in a tower environment we could find the following areas:

- Flights that have started up, but have not started taxiing yet.

- Flights in the taxiing phase.
- Arriving flights, but they are not in the frequency yet.
- Arriving flights that are in the tower frequency and are expected landing clearances.
- Flights cleared to take off, land or cross an active runway

1.2.1.2 Electronic Flight Strips

The Electronic Flight Strips has the same purpose as the Flight Progress Strips. Arguably, they were created to solve some Flight Progress Strips problems, such as linking the paper strips with other Systems or time consuming while printing and updating. In addition, it's more visual, the information could be shared instantly and due to is linked with the ATM System, it can issue an early warning of a conflict.

In the other hand this technology can't replace the paper strips despite their advantages, since there can be power and software failures.

1.2.1.3 Radar

Several types of radar are currently used to perform the tasks of controllers. The word RADAR is an acronym for RAadio Detection And Ranging. It works very simply, it sends out energy by means of electromagnetic waves that travel at the speed of light. When it encounters an object, part of this energy is bounced back as an "echo" and is received by the radar device itself. When this signal is received, the direction and distance is known thanks to the time it has taken for the wave to travel back and forth.

At an airport there are three types of radar: primary radar, secondary radar and ground movement radar.

The primary radar is very basic, it does exactly what has been explained previously in the operation of a radar. The signal obtained from this radar is displayed on a monitor where the controllers will see a point at a certain azimuth at a certain distance, there will be no other information. In this case, whether it wants to or not, the aircraft will be detected.

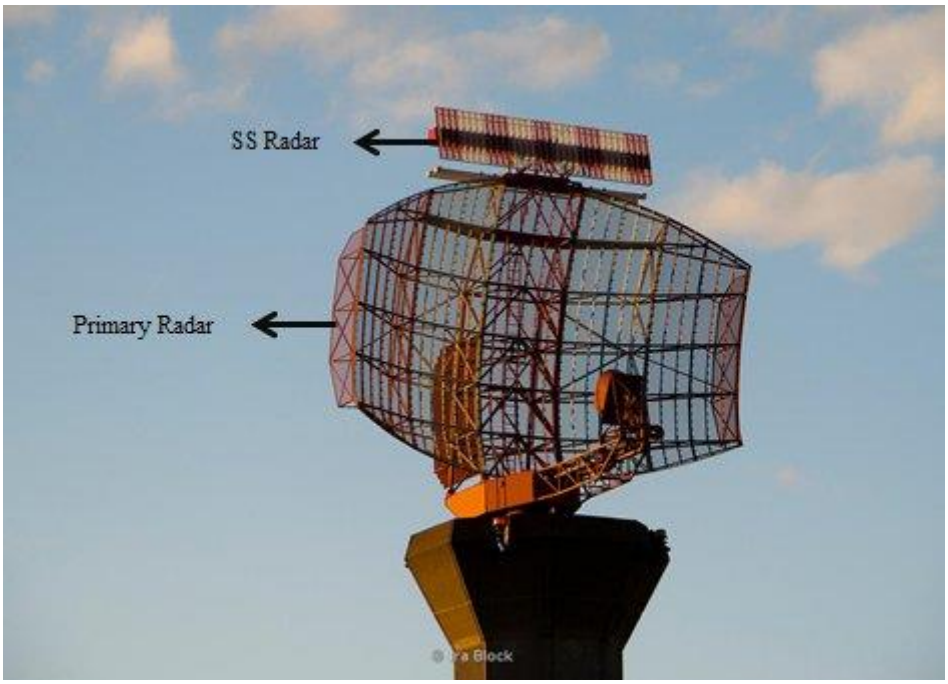


Fig. 1.4 Primary and Secondary Radar

On the other hand, the secondary radar has a different function than the primary radar. For the correct functioning of this system, the target must be cooperative, the aircraft must be equipped with a transponder (transmitter-receiver) and receive a coded signal from the radar interrogator. The active response is generated on the transponder, which is also coded and sent back to the secondary radar. The travel time is also used as a measure of distance, as in the case of the primary radar. This response contains much more information (e.g. altitude, identification or technical problems on board, e.g. radio failure) than is possible with the primary radar. Both this radar and the primary radar are not only located at airports, but are distributed over the whole territory in order to have a network covering the whole airspace.

Finally, there is the Surface Movement Radar (SMR) which is specific for the ground movement of an airport, detecting both aircraft and vehicles. The image of the SMR is shown in a console in the control tower. This tool is used by the ground and tower ATC's to supplement the visual observation or to substitute in poor visibility conditions. The SMR is used for the following purposes according to Skybrary (see in [8]):

- Enhance the controllers' situational awareness regarding the manoeuvring area
- Monitor the movement of aircraft and vehicles on the manoeuvring area
- Provide routing information to pilots and vehicle drivers as necessary
- Provide advice and assistance for the safe and efficient movement of aircraft and vehicles on the manoeuvring area.

This technology is only used in the main airports with a high aircraft flow, for this reason cannot allow the airport to close due to bad weather conditions.

1.2.1.4 Remote Tower Service

The Remote Tower Service (RTS) according to the Skybrary (see in [9]) definition is “a system which allows aerodrome Air Traffic Control (ATC) or Flight Information Service (FIS) to be provided from a location other than the aerodrome whilst maintaining a level of operational safety which is equivalent to that achievable using a manned Tower at the aerodrome to oversee both air and ground movements”.

Fig. 1.5 Remote Tower Position [9]

This service was firstly implemented in Sweden in 2015, but nowadays is being used in more airports such London City and Dublin.

The main advantages of this system with respect the conventional air traffic control tower is the low cost of the infrastructure, the reduced space it occupies and the low visual pollution it causes.

1.2.1.5 Airport collaborative decision-making

The main objective of Airport Collaborative Decision Making (A-CDM) is to optimise the use of resources and improve the predictability of air traffic, thereby improving efficiency. For this purpose, airport and aircraft operators, ground handlers and air traffic control are encouraged to work in a more transparent and collaborative way by sharing information.

This solution is already implemented in 32 airports across Europe, including Barcelona-El Prat airport. Fig. 1.6 shows the different European airports that have implemented this system or are in the process of doing so.

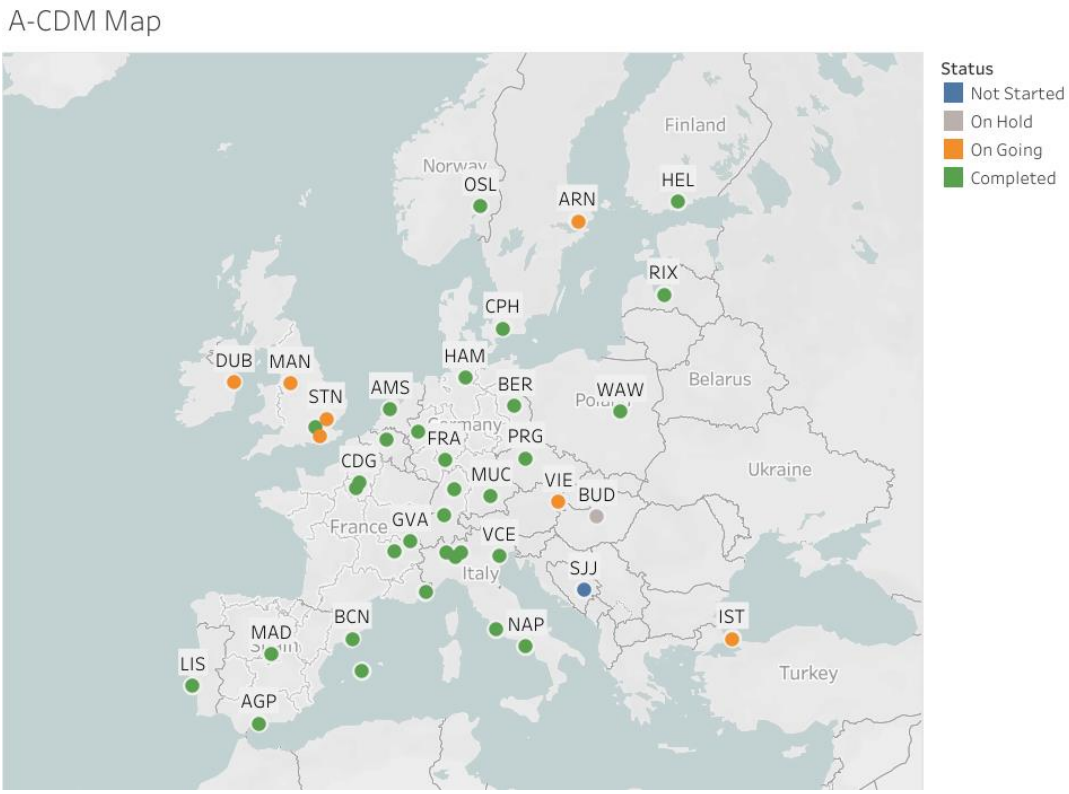


Fig. 1.6 A-CDM Map [10]

1.2.2 Future tools in testing phase

Nowadays some airports continue with the same technology as many years ago, but the workload has been multiplied since the air traffic has undergone a huge increase. Moreover, we are in a climate crisis and air traffic is one of the main causes. This is why more sustainable solutions are beginning to be developed and tested.

1.2.2.1 Surface Metering

Surface Metering is a decision support tool which brings a solution to the controller using the best available data. Using predetermined procedures, monitors the current situation and predicts future state. This tool controls the release of an aircraft from an airport, sometimes for purposes that don't affect the current airport, but the en route traffic and destination airport. For example, if the taxiways are congested and an aircraft requests a pushback, the result of the metering will be to hold in the gate until there is a smoother flow.

To bring about this solution it requires the collaboration of the FAA (the United States), flight operators and airports. Flights and Airport operators, Air Traffic Control and Flying public can benefit from this tool. The first, could benefit from a lesser taxi time and thus fewer fuel burn, so the flight operators could save more money and the airport reduce the environmental footprint. The Air Traffic Control will have fewer movement aircraft, meaning an improved safety and for the last, the public will be more satisfied because there will be fewer delays and a more reliable schedule.

1.2.2.2 Airspace Technology Demonstration 2 (ATD-2)

As I explained before, today's air transportation is suffering a big inefficiency due to the lack of information sharing amongst the operators. NASA and the FAA with the collaboration of some airlines like American Airlines and Charlotte-Douglas International Airport are developing a technology to improve the handling of arrival, departure and airport surface traffic.

Additionally, the procedures of taxi-out, takeoff and climb phases of the flight are subject to great uncertainty, which causes inaccurate predictions of the air traffic flow.

The solution that NASA has developed is the integration of the arrival, departure and surface concepts by leveraging the actual supporting systems technologies (Traffic Flow Management System (TFMS), Time Based Flow Management (TBFM), and Terminal Flight Data Management (TFDM)).

As it is explained in the NASA webpage (see in [12]), the ATD-2 Field Demonstration will preview several aspects of the under-development TFDM system such as Surface Collaborative Decision Making (Surface CDM) departure metering, the Electronic Flight Data user interface in the airport tower, and data sharing with industry via the TFDM Terminal Publication (TTP) service. This solution not only takes into account the local airport, instead, it also couples a trajectory-based surface decision supporting tool with the overhead stream insertion capabilities of the TBFM en route metering decision support system.

1.3 How will this work affect air traffic controllers?

To begin with, this work would only affect the ground controllers because the performance of the simulator only affects the movements on the apron from the runway to the assigned stand and vice versa. Depending on the results of this work, it may vary from becoming just another tool to completely replacing the ground controllers. In addition, this work may render obsolete some of the current tools used by air traffic controllers and may also use some existing tools

such as the different radars to obtain the position of the different aircraft at the airport.

CHAPTER 2. Barcelona-El Prat Airport

This chapter explains different aspects of Barcelona-El Prat airport, such as its history, its geographical environment, information about the number of passengers per year, explanation of the different terminals, among other aspects. Besides, it explains how the airport traffic is and then comments on specific characteristics of this airport, such as the different runway configurations, the stands and some peculiarities about the taxiways.

2.1 Introduction

At the beginning, in 1916, the airport was located in a place called "El remolar". Nowadays, the airport is located 12 km southwest of Barcelona. Later, between 1941 and 1946, the airport was renovated to occupy the space of the Aeroclub de Catalunya, built nearby just a few years earlier. In 1948, runway 07-25 was built, which is currently the airport's main runway. Years later, a third runway was built perpendicular to the previous one (16-34), this extension was also accompanied by new taxiways and a passenger terminal.

In the 1960s, new extensions were made to runway 07-25 with new exit taxiways and the terminal, as well as a control tower and a new airport apron for parking aircraft. At the end of the decade runways 07-25 and 02-20 took on their present form.

In the 1970s, the first transatlantic flights from Barcelona to New York and the air bridge between Barcelona and Madrid began. With these new services, in 1977 the airport reached 5 million passengers. Also, in this period a cargo terminal was built.

In the 90's, the airport underwent a great growth, mainly due to the Olympic Games held in the city of Barcelona in 1992. Terminals A and C were built and the existing terminal B was extended. The first fingers appeared and the volume of passengers exceeded 10 million a year. With the use of radio aids such as the ILS at head 25 for landings and the use of runway 20 for take-offs, operations improved to 50 operations per hour.

In the 2000s, the airport underwent its third major transformation after 1968 and 1992.This transformation consisted of improvements to the terminals, lengthening and widening of the main runway and, above all, the creation of a third runway parallel to the main runway. Apart from these improvements, in 2009 it was also inaugurated another passenger terminal, currently known as T1.

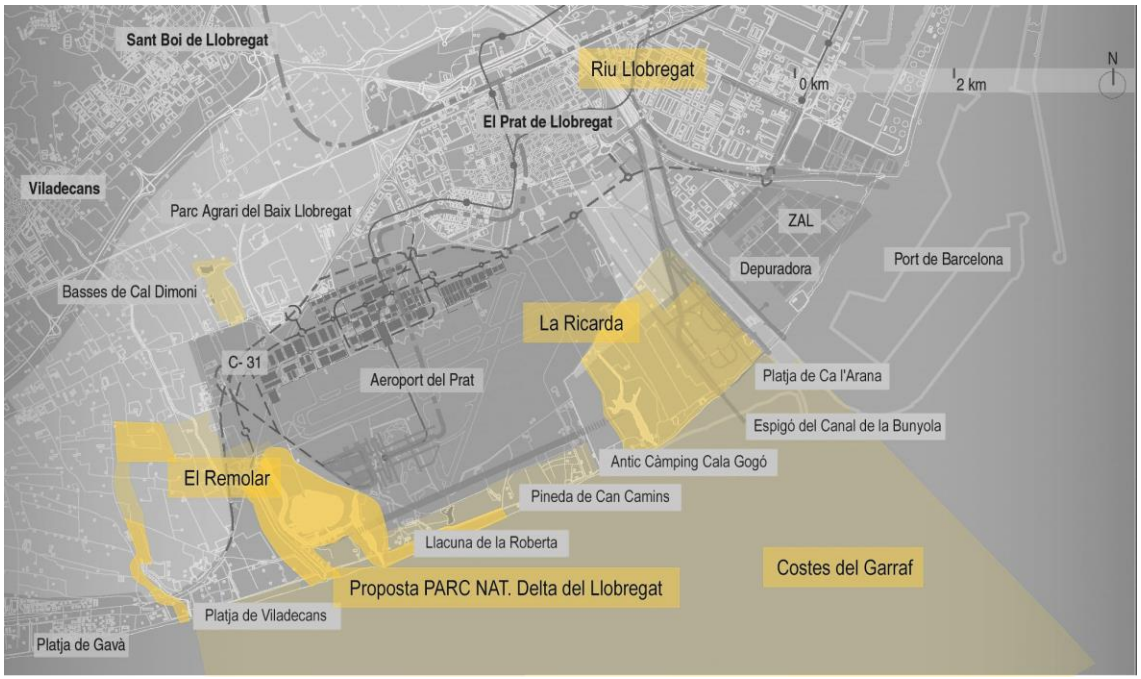


Fig. 2.1 Geographical situation of Josep Terradellas Barcelona – El Prat Airport [15]

2.1.1 General information

Currently, it is the second largest international airport in Spain, after Madrid-Barajas International Airport, and is the first in Catalonia and the Mediterranean coast. In 2018, Barcelona Airport beat the record of 50.2 million passengers and ranked sixth with the most passenger traffic in Europe.

The airport has 3 take-off and landing zones, thus obtaining 3 runways in service, two in parallel called 07L/25R and 07R/25L, and a cross, runway 02/20. At present, it can perform up to 78 operations an hour.

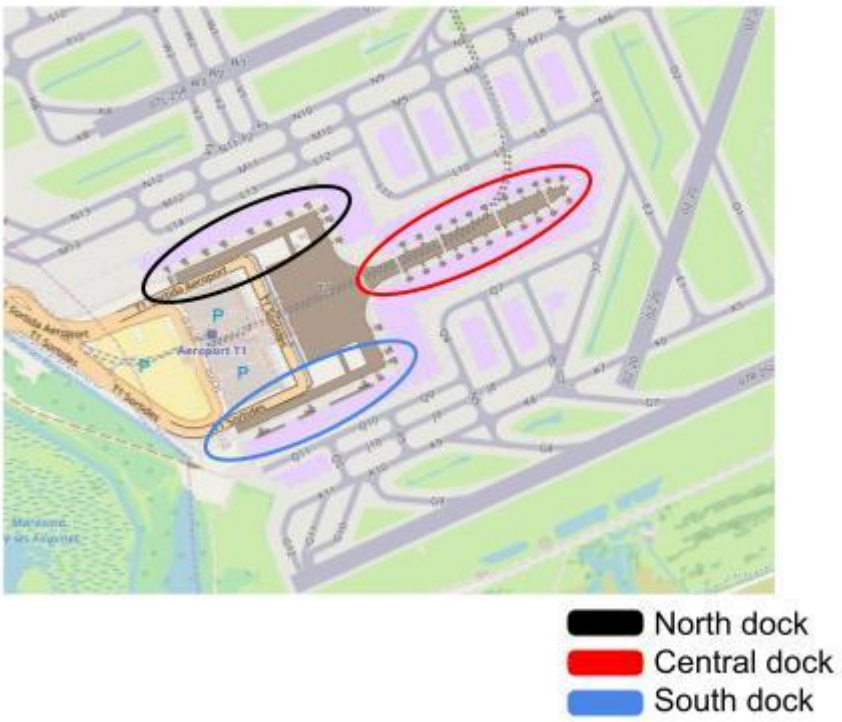


Fig. 2.2 Terminal T1 docks

It has two passenger terminals, terminal T1 and terminal T2. Terminal T1 has a total area of 544,066 m2 and a capacity of 33 million passengers per year. Currently 70% of flights use this terminal. As can be seen in Fig. 2.2, the terminal is made up of 3 large areas, the north dock, the central dock and the recently remodelled south dock. On the other hand, the T2 terminal consists of 3 subterminals, T2A, T2B and T2C. The first is used for non-Schengen flights and the second for Schengen flights. The T2C terminal is operated only by easyJet, but currently is closed. Recent changes in the structure of terminal T2 have meant that what used to be three different terminals, terminal A, terminal B and terminal C, are now united in a single terminal, terminal T2. Fig. 2.3 shows the structure of terminal T2 at Barcelona-El Prat airport.

Fig. 2.3 Passenger terminals at Josep Terradellas Barcelona - El Prat airport

The airport also has 16 apron parking ramps, which in total consist of 233 stands. There are two options of stands, those that are connected to the terminal via a finger and those that are remote, which require either boarding buses or walking on the apron to reach the terminal. Fig. 2.4 shows the different stands, their pushback path and which ramp each of them is part of.

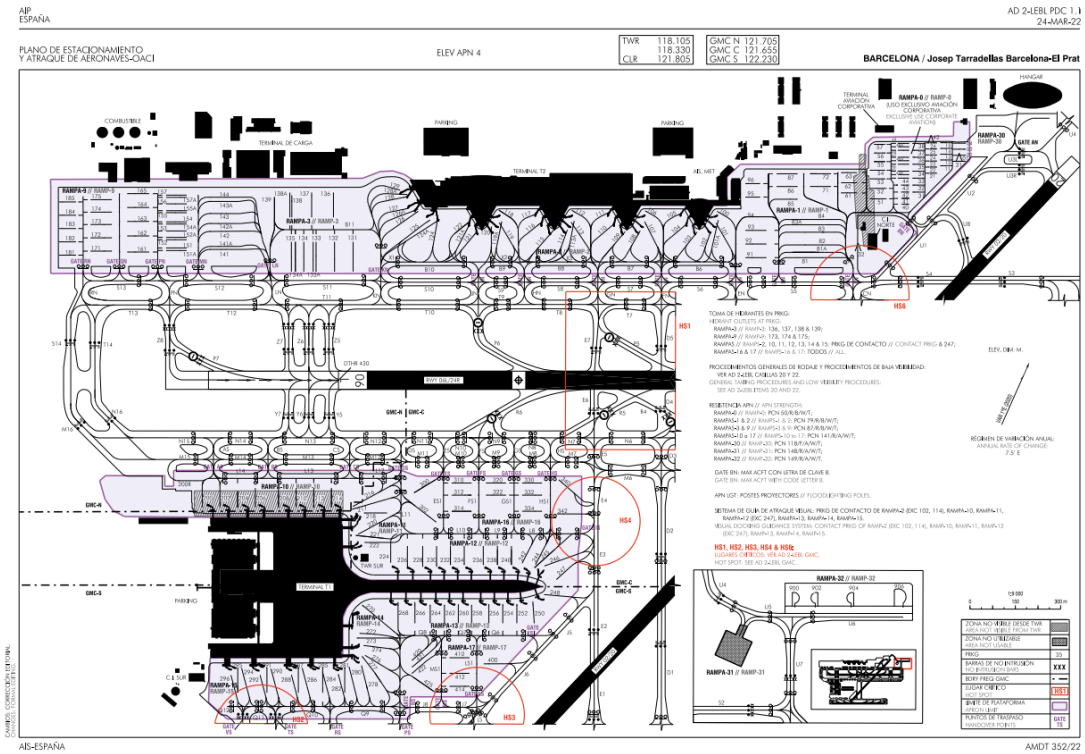


Fig. 2.4 LEBL PDC 1 [16]

The airport's main domestic destinations are Madrid and Palma de Mallorca. Internationally, the main European capitals such as London, Paris and Amsterdam stand out. The main airline in Barcelona, with the largest number of passengers and operations is Vueling, due to the fact that Barcelona is its hub.

Recently, Barcelona's El Prat airport has been in the news for its future expansion. This extension includes a new satellite terminal connected underground with the current terminal T1 and an extension of the runway 25L/07R of 500m towards the protected natural area of “La Ricarda”. The aim of this extension is to improve operations by increasing its capacity to 72 million passengers a year and the hourly capacity up to 90 movements an hour and to strengthen the long-haul to turn the airport into a major international hub.

2.1.2 Scenario

Fig. 2.5 shows that the airport's annual traffic is seasonal, in other words, in the summer months the traffic is higher than in the winter months. Apart from this seasonality, there is also a problem with certain times of the day, since at certain times of the day all airlines want to depart and/or arrive and at other times nobody wants to operate at the airport. Combining seasonality with the problem of peak and off-peak hours makes that the demand during certain hours in the summer season exceeds the capacity, creating significant delays. As can be seen in the Fig. 2.6, demand is equal to capacity between 7 and 9 a.m. and also between 5 and 7 p.m. This is the problem that is observed at the airport with the delays in August when many people come and go on holiday.

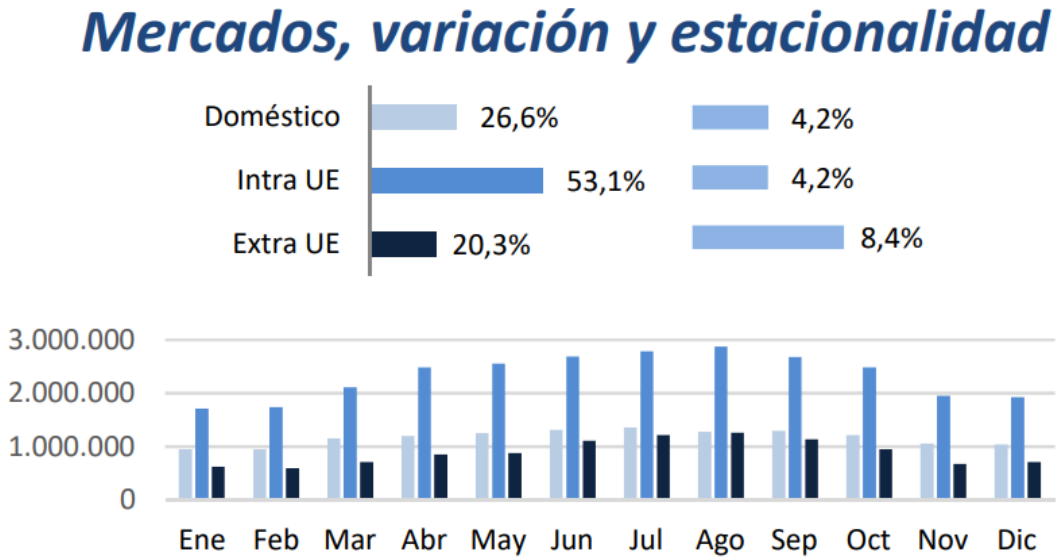


Fig. 2.5 Airport traffic by market and month [17]

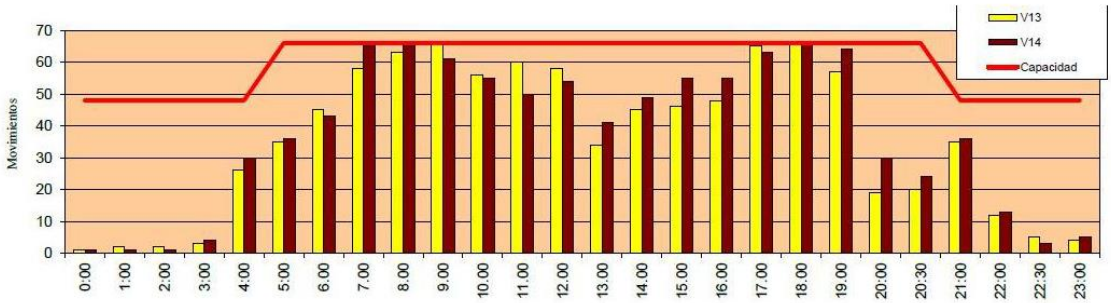


Fig. 2.6 Hourly airpoprt traffic [18]

Another point to note is the different runway configurations in which the airport can operate. Due to the fact that Barcelona has 3 runways (2 parallel runways and the other cross runway) and its geographical location (surrounded by urban centres), the different runway configurations used are as follows: west configuration parallel runways, east configuration parallel runways, north configuration intersecting runways and west configuration single runway.

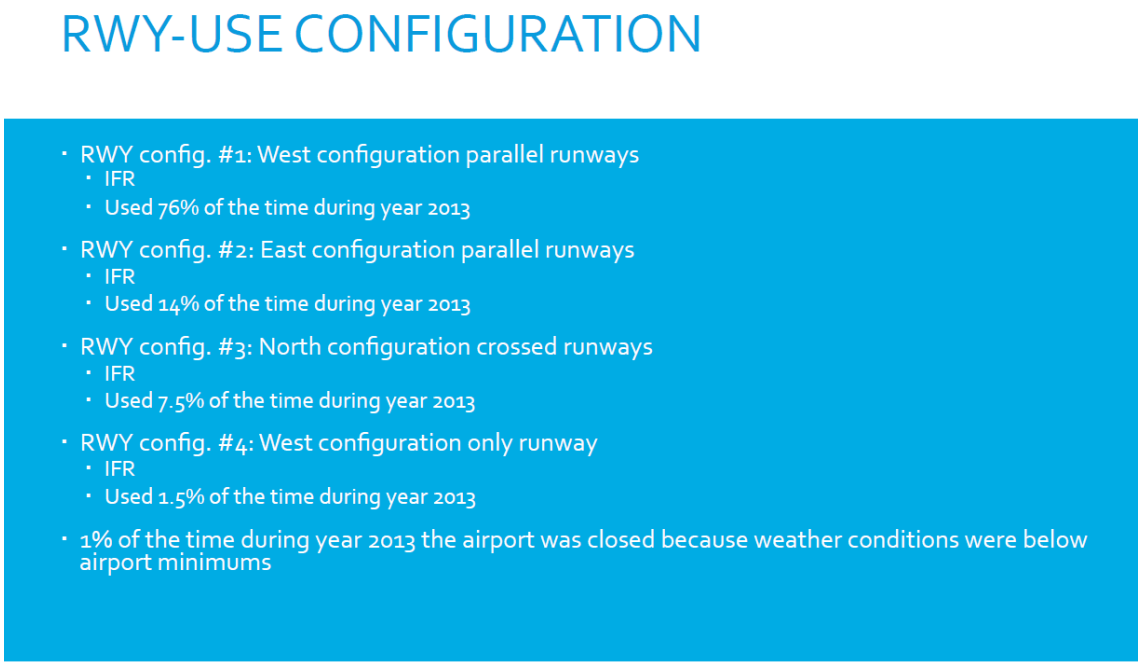


Fig. 2.7 Runway use configuration [18]

2.1.2.1 West configuration parallel runways

As seen in the Fig. 2.7, this configuration was the most used in 2013 and still, the vast majority of the time is the one that is used. The AIP of the Barcelona –

El Prat Airport, explains that this is the preferential configuration from 07:00h to 23:00h. The runway 25R is where all the airplanes are going to land. On the other hand, runway 25L is intended for take-offs only. Due to the fact that the runway 25L is shorter than the 25R, those aircraft which can justify that need more runway length than the available length for runway 25L could use the runway 25R. The optimal configuration will be the runway 25R for all the take-offs and the 25L for all the landings, but the neighbours of the urbanisation of Gavà Mar complained about the noise they received from the planes passing over them. Henceforth, this configuration was changed to the one we have now. Also, another reason why this is the preferred configuration is due to the fact that the prevailing wind is from the west. Fig. 2.8 shows the actual configuration.

Fig. 2.8 West configuration parallel runways [21]

2.1.2.2 East configuration parallel runways

As seen in the Fig. 2.7, this configuration is the second most used. The usage of this runway configuration is not preferential, it is only used when wind and weather conditions are not matched to use the West configuration parallel runways from the 07:00h to 23:00h. All the arrivals take place in the 07L runway, and most take-offs will take place on runway 07R, but for those aircraft that need more runway for take-off, they can depart on 07L. Fig. 2.9 represents schematically the East configuration parallel runways.

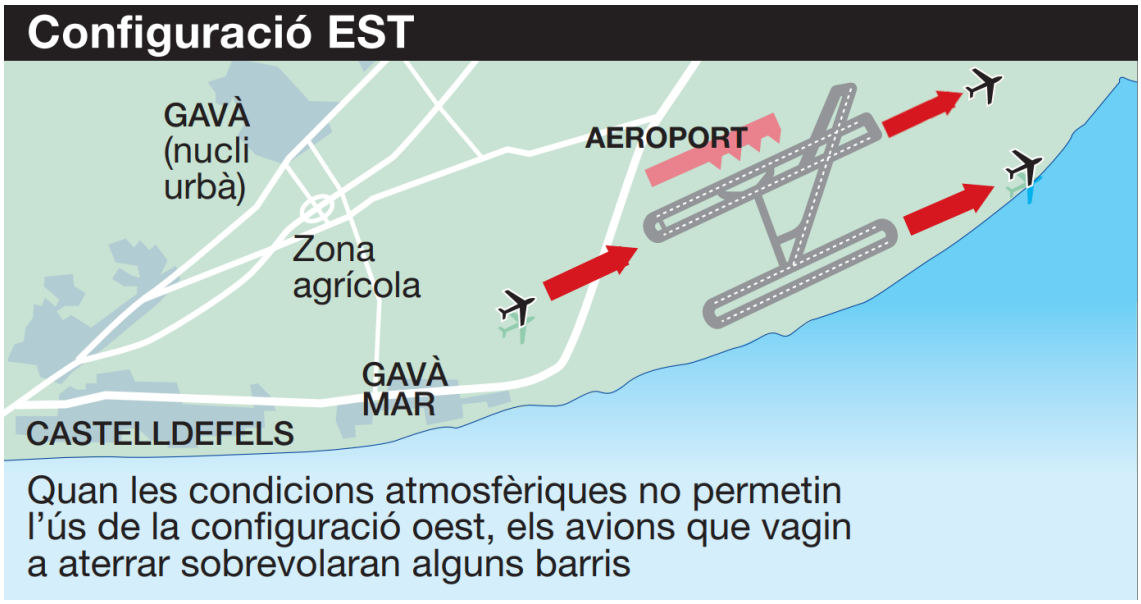


Fig. 2.9 East configuration parallel runways [21]

2.1.2.3 North configuration intersecting runways

According to Annex VIII of the AIP of Barcelona – El Prat airport (see in [20]), the North configuration intersecting runways is the preferential configuration between the 23:00h and 07:00h. The main reason for using this configuration at night is to reduce noise around nearby urbanizations by preventing aircraft from flying overhead. In Fig. 2.10, one can observe a schematic image of this configuration.

Fig. 2.10 North configuration intersecting runways [21]

2.1.2.4 West configuration single runway

This configuration is the least used nowadays and back to the 2013 too, according to Fig. 2.7. This configuration is the backup option in the night shift, but it can also be used in bad weather conditions. The arrivals and the departures take place in the same runway, the 25L.

2.1.3 Airport characteristics

As explained before, the Barcelona - El Prat airport has two terminals, T1 and T2. The oldest one (T2), is now used by low-cost companies such as Ryanair, Easyjet or Transavia. On the other hand, the T1 Terminal T1 concentrates the flights of the companies belonging to the One World, Star Alliance and SkyTeam alliances, its associates and those operating intercontinental flights.

Furthermore, there are some stands that can be merged to become a larger one to accommodate a bigger aircraft, an example of which can be seen in Fig. 2.11, that the stands 280 and 282 of the T1 terminal are merged into the stand 281 that now it can held up to a Boeing 747 and before each one of the stands were designed to park an Airbus 320 in each one.

Fig. 2.11 Stand 280, 281 and 282 from Barcelona – El Prat airport [22]

Another interesting point of Barcelona-El Prat airport is that due to the fact that the airport can have different runway configurations, in each of these configurations, the different taxiways can have one or both directions, and by changing the configuration the direction of the taxiways will also change. In the Fig. 2.12, one can observe the difference of the directions in the taxiways that are next to the runway 25R/07L.

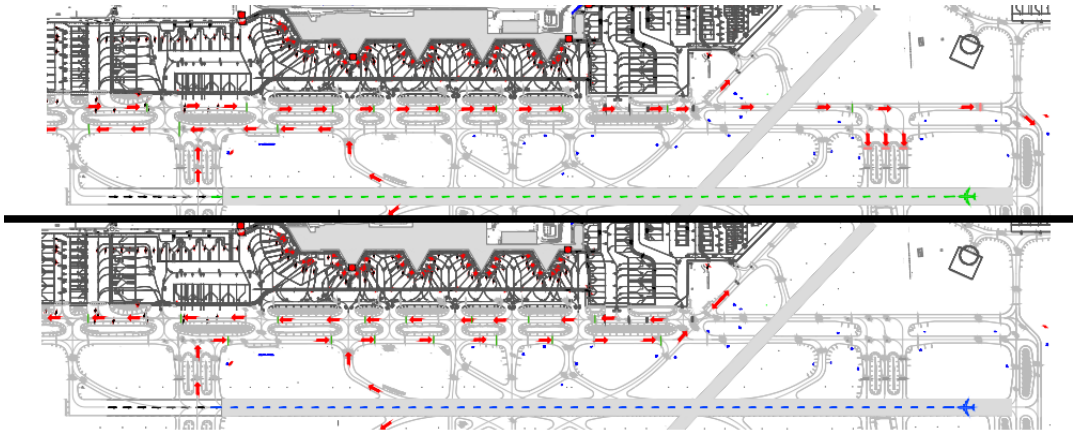


Fig. 2.12 Taxiway direction with different runway configuration [23]

One of the peculiarities of the airport is the delay that is produced during the summer months (June, July and August). According to CatalanNews website (see in [24]), in 2018 nearly half of the flights were delayed by more than 15 minutes over that period. The average time that passengers had to wait for their planes was 55 minutes during this period. This delay is mainly produced by companies such Vueling and Ryanair. It's also true that in 2018 the Ryanair cabin crew and air traffic controllers in France went on strike.

Some of the airport's peculiarities have already been mentioned, such as the use of runway 25R-07L also for take-offs when the airport is operating in west configuration parallel runways or the change of runway configuration depending on the time of day to reduce the noise footprint over the Gavà Mar urbanisation.

With the west configuration parallel runways the configuration of the taxiways is very similar to the one obtained in Fig. 5.1, with the only difference being that the two vertical taxiways in front of the central dock of T1, instead of having a single direction, both are bidirectional and are used in one direction or the other depending on what is most convenient.

If the aircraft is heading towards runway 25L for take-off, before it reaches taxiways G1, G2 and G3 (the last taxiways before entering the runway), the aircraft looks to see which aircraft are on these taxiways to see if it has priority over any aircraft. In this case, priority is determined by which aircraft has the earliest take-off time. Generally, if there are no aircraft, taxiway G1 is used, if there is an aircraft on taxiway G1 but its departure time is earlier than the aircraft, it will also go to taxiway G2. And the same happens with G3, which will be used only in the case that G1 and G2 have some aircraft and their take-off times are later than the aircraft. The Fig. 2.13 shows the location of these taxiways.

Fig. 2.13 Taxiways G1, G2 and G3

One aspect to comment on Barcelona-El Prat airport is the delay and taxiing time it has, for which we will explain some parameters that will help to understand the airport's situation in 2019.

2.1.3.1 ATFM Delay

According to Eurocontrol (see in [26]) the resulting ATFM delays are calculated as the difference between the estimated take-off time calculated from the filed flight plan including updates and the calculated take-off time allocated by the central unit of ATFM to avoid congestion in some airports or en route centers.

In Barcelona, as can be seen in Fig. 2.14, the ATFM Delay is usually above 2 minutes on average in the months of highest demand. It is important to note that this delay is mainly caused by the weather and, secondarily, by the capacity of both the airport and the airspace.

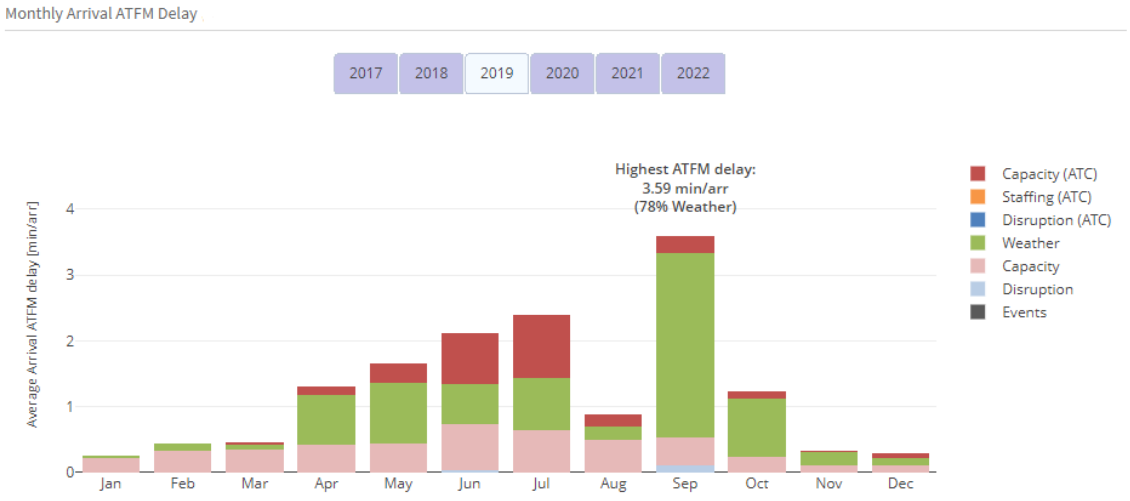


Fig. 2.14 Monthly Arrival ATFM Delay in 2019 [25]

2.1.3.2 Taxi-Out Time

El Taxi-Out Time according to Eurocontrol (see in [27]) is defined as the time spent by a flight between its actual off-block time (AOBT) and actual take-off time (ATOT). The Unimpeded taxi-out time is an average Taxi-Out Time when there is no congestion. There is one unimpeded time by departure airport, departure runway and departure stand and the Additional taxi-out time is the difference between the actual taxi-out time and the unimpeded Taxi-Out Time.

In 2019 at Barcelona-El Prat airport, the average Taxi-Out Time is 11.13 minutes per departure, but the additional time is 4.5 minutes per departure. These values can be seen in Fig. 2.15.

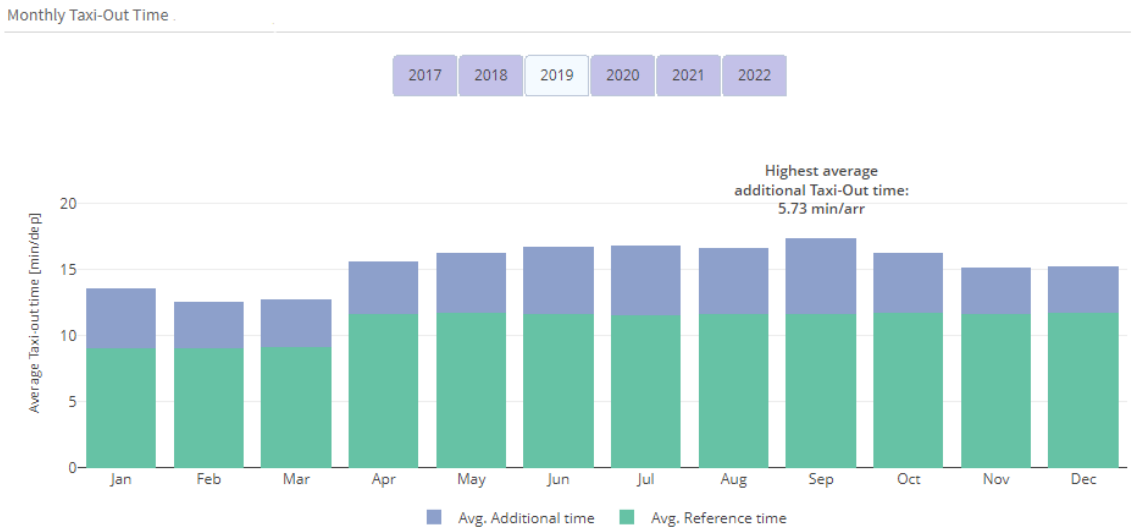


Fig. 2.15 Monthly Taxi-Out Time in 2019 [25]

2.1.3.3 Taxi-In Time

Taxi-In Time is defined by the time spend between the Actual Landing Time (ALDT) and the Actual In-Block Time (AIBT).

According to Fig. 2.16, in 2019 in Barcelona, the Taxi-In Time is approximately 2.3 minutes per arrival and has an additional time of approximately 0.64 minutes per arrival.

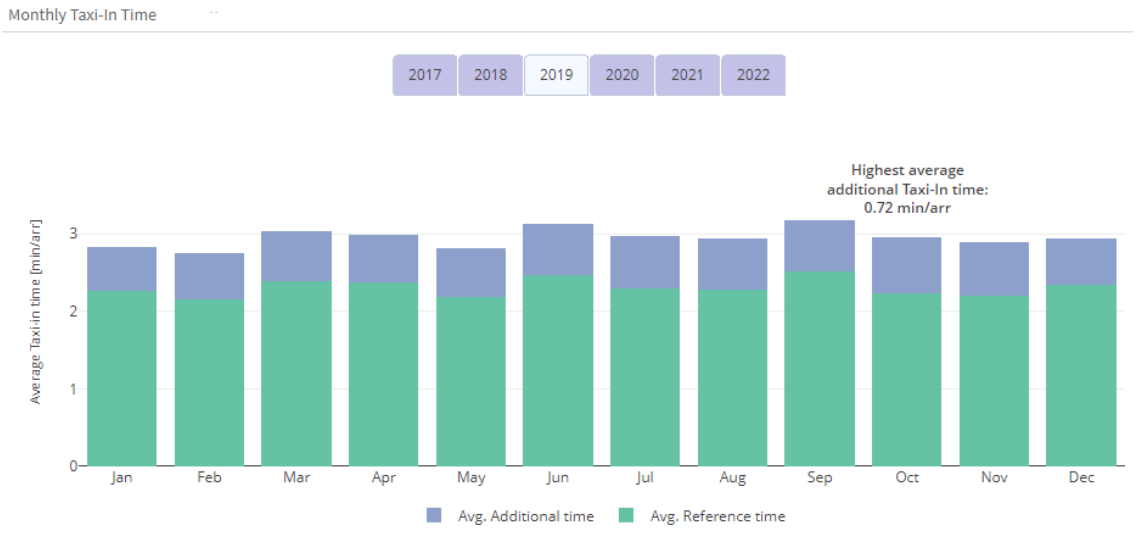


Fig. 2.16 Monthly Taxi-In Time [25]

2.1.3.4 Pre-Departure Delay

The ATC pre-departure delay according to Eurocontrol (see in [28]) is the additional time that the aircraft is held at the stand to avoid queuing at the departure runway. It is a proxy of the delay which an aircraft ready to leave its gate can be subject to, at its origin airport, due to airports constraints, demand/capacity imbalances known prior to off-blocks, take-off restrictions and/or traffic intensity at the time of operations.

In Barcelona, it can be observed that this type of delay is more common in the busiest months (summer period). According to the graph in Fig. 2.17, the main reason for this delay is not identified, but the second most important is caused by late arrival of aircraft, crew, passengers or baggage from previous journeys.

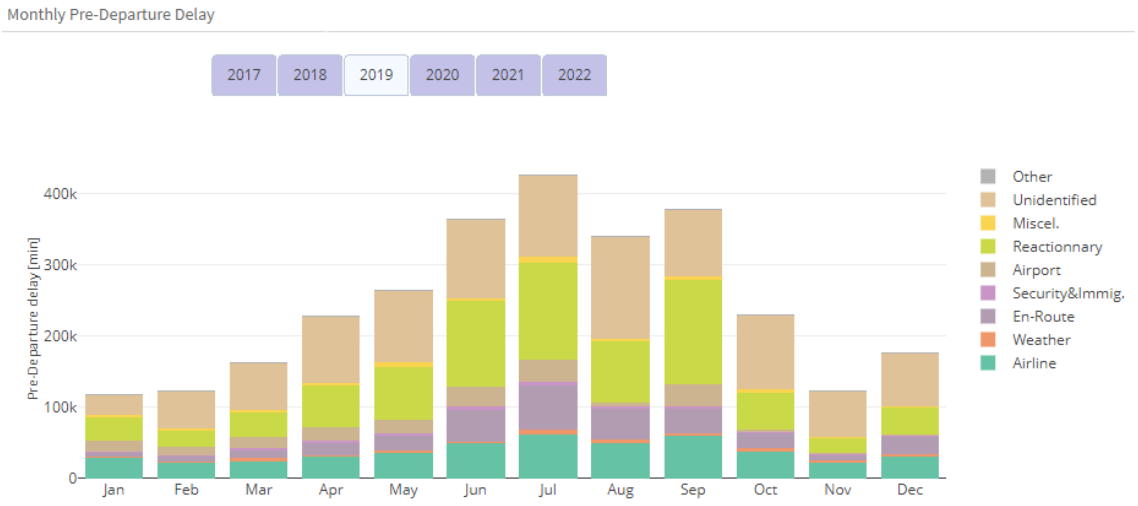


Fig. 2.17 Monthly Pre-Departure Delay in 2019 [25]

2.1.3.5 Punctuality

Punctuality means that an airplane arrives to its destination gate earlier, on-time or no more than 15 minutes later than the scheduled time.

In Fig. 2.18, we can see a graph where the departure punctuality is differentiated into different bands depending on the time difference between departure and the estimated departure time. As can be seen, the red and dark yellow colours are the aircraft that are not punctual, on average 25% of the aircraft are not punctual.

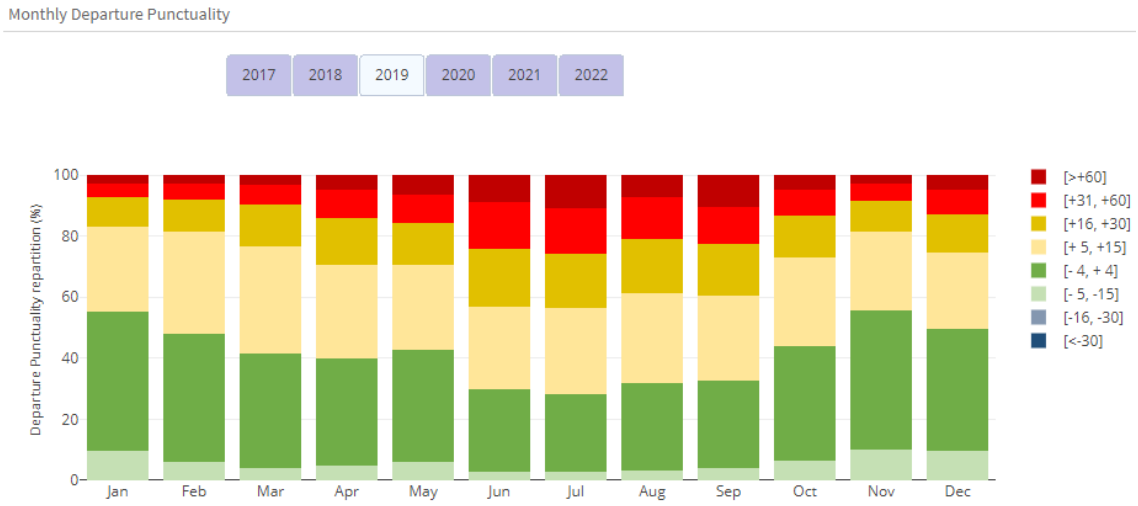


Fig. 2.18 Monthly Departure Punctuality [25]

2.2 What will be used in the simulator

In this chapter, the main information to be used for the development of the simulator is Fig. 2.4 to configure the different stands, the different runway configurations and consequently the direction of the taxiways involved in each runway configuration. Finally, it will also be used the different graphs of the delay and taxi time to compare the results obtained from the simulator with the previously shown graphs, in this way it will be possible to obtain a comparison metric and to know in what percentage the created programme is more or less good than the current system.

CHAPTER 3. Objectives and work plan

This chapter will discuss the main objectives of this Thesis, also the prediction of the possible results and conclusions of the work. Moreover, it will be explained how it is planned to develop the project, the expected result and the methodology used for the development of this Thesis

3.1 Specific objectives

As can be deduced from the title of this Thesis, the main objective is to automate the functions of ground controllers by developing a software that replaces these controllers or, at least, facilitates their work. The overall result of this thesis will be satisfactory if the answer to this main objective is positive. However, in order to achieve this objective, there are other more specific objectives that must be previously achieved for the correct development and operation of this project.

The first specific objective is to find the right simulator for the realisation of this project. To do so, we have to choose what kind of simulation we want, 3D or 2D, take an existing program like Flight Simulator and change its source code so that it performs the functions of the ground controller autonomously or take a programming environment and start from scratch. To get an answer to this specific objective it is necessary to know for sure what kind of simulation we want and is best for the controller.

Another specific objective is to learn a new programming language or to improve the level if the language used is one already used in a computer science course at the university. This objective is closely linked to the previous one because depending on the simulator used, the language will be new or not.

Another very important objective is to be able to solve all the problems encountered during the development of this work. These problems can be both on the programming side and on the side of obtaining information for the simulator. Also, it is very important that the programme is not specific to Barcelona-El Prat airport, but that it is as general as possible for future implementations in other airports, simply by changing the data inputs.

A fourth specific objective is to know how to interpret the results to see if, in addition to automating the functions of the ground controller, it is also possible to improve the operation of an airport by improving its capacity. For this reason, the airport chosen is Barcelona airport, as it is easier to obtain the necessary information to later analyse and compare the performance of the simulator with the actual performance of the airport. Furthermore, the programme can be used to model new infrastructures such as new taxiways or a new terminal, and how these changes will affect the airport.

3.2 Product description

The desired product will be a programme that can replace the ground controller by performing the same functions and also optimise the route from the runway to the corresponding stand and vice versa. In addition, the optimisation will also include the minimisation of the hold in position that many aircraft suffer during the peak hours of demand at an airport, in other words, the program will make an aircraft move most of the time so that they leave the stand at the optimal time and in the optimal order.

The programme will have a model of the airport, where you will be able to see the movement of the planes, and their information, such as the arrival and departure times, the stand they have been in, the taxiways through which they will circulate, among others. You will also be able to change the speed of the simulation and even stop it. Another feature to be added is to make this programme more interactive, with options to change the route or stop and restart a specific aircraft. In addition, at the end of the 24-hour simulation, the programme will create an Excel document with different values to analyse the behaviour of the simulator. These values are the taxiing time vs. the stopped time, the generated delay and the average speed of the aircraft.

It is also true that one of the main objectives of this thesis is to use this product in real life, so to bring it to reality the first thing to do is to contact the airlines and the different airports showing the simulation and its results to see how to implement this solution in real life. So, the simulation that will be done will show what will happen on a specific day at an airport if this system is taking over the functions of the ground controller. In other words, at the end of this project there will be a code that will work correctly but there will not be a system designed to replace those controllers because this is not the objective of the thesis.

3.3 Possible conclusions

The most possible conclusions are that the simulator works correctly, although it is possible that it does not know how to handle some very specific situations, which is why, as it is not fully robust, it is not possible that it can fully replace the ground controllers, but it can make their work much easier and even reduce the number of ground controllers needed per airport, as the programme will make decisions, mostly appropriate ones, and the controller will only have to verify them.

In addition to the issue of robustness, the simulator will have no delay, because the delay is mostly caused by the handling agents and the airlines, not by the routing of the taxiways. This does not mean that all aircraft will depart at the scheduled time.

Another conclusion that can be drawn is that the results obtained from a certain scenario cannot be compared with reality, as the airport does not provide this kind of information. For this reason, other metrics may have to be considered in order to evaluate the correct functioning of the simulator.

3.4 Technical references

The main technical references to be used in this project are a book called "Cómo mejorar la logística de su empresa mediante la simulación" (see in [30]) and some web pages about programming that can be of great help in some questions during the coding part of this Thesis. Reading the book could help by explaining how to organise the code and be more optimal by reducing some calculations.

Apart from these references according to the programming part of this project, some information about the airport to be modelled will be needed, for example the name of each taxiway and stand, and some more technical information such as the information contained in the Aeronautical Information Publication of an airport, also known as AIP (see in [19]), or the directionality of the taxiway.

3.5 Planification

The planning of this project is based on phases. Starting with being clear about which simulator to use and then how to structure the code taking into account most of the variables. Once the structure is clear the first step is to start coding with the basics and above all to do a lot of tests to make sure that the different functions work correctly, as it is seen that these functions work correctly, the next step is to add the next function to make the simulator as realistic as possible. The different tests will be purposely designed to check the functioning of the newly created function, and so on. At each stage new functions and variables will be added until the desired final product is reached where all functions of the ground controller will be automated. Finally, when the simulator is ready, a last simulation will be made to analyse the results and write them to the memory where part of the background of the work will have already been written.

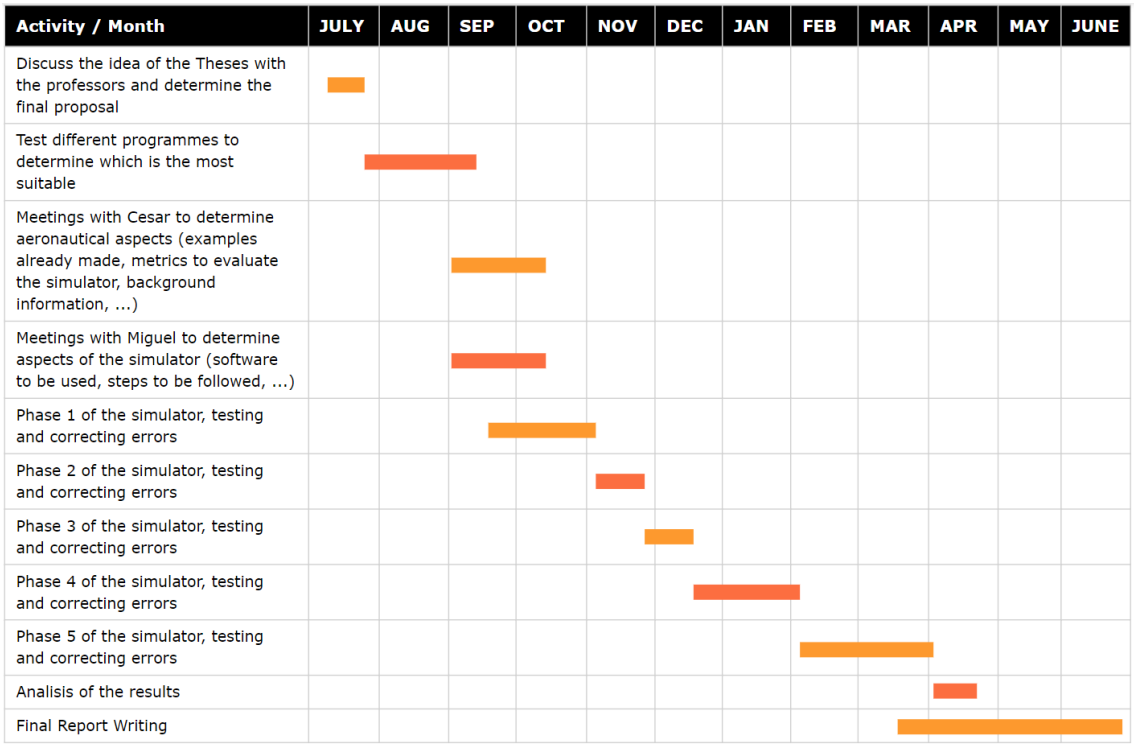


Fig. 3.1 Thesis timeline

CHAPTER 4. Modelling and simulation in the airport industry

Recently, with the significant increase in air traffic, with the help of simulation software, an improvement in the optimisation of the different processes that take place at the airport, both on the airside (the part of an airport used by aircraft for loading and unloading and takeoffs and landings) and on the landside (defines those areas and activities within the airport boundaries, exclusive of the airside), is being tried with the help of simulation software. In this way, you get more out of your resources without the need to expand them. Simulation has become a fundamental part of airport operational planning and, as a result, different software has been developed to simulate passenger behaviour. There is also a German company that has software not only to simulate the behaviour of passengers but also that of the handling and ground vehicles on the apron, the movement of taxiing aircraft, among others.

One aspect to emphasise is that the different simulation models are made to simulate the movement of the different agents (passengers, aircraft or vehicles) and predict what will happen at any given moment, in no case are they designed to support ground controllers. These predictions are used to subsequently define the capacity needed to avoid bottlenecks and thus increase or reduce the number of passport controls, security or check-in counters in the case of passenger flow within a terminal.

4.1 Passengers movement

An example of modelling previously used in the airport environment is the simulation of passenger flow in the terminal. This is a widely used resource at several large European airports to improve capacity planning and customer satisfaction, e.g. by optimising airport processes and reducing waiting times. With this programme, real airport data is used and future behaviour can be simulated. This programme was used for Frankfurt Airport, Lufthansa's hub, where there are more than 140,000 passengers per day. In this particular case, the model consists of: 26 security checkpoints, 8 boarding pass checkpoints, 15 border checkpoints, 90 stairs and lifts, 266 boarding gates, 1 tunnel and 3 SkyLine stations. This programme is run several times a day to obtain the passenger flow forecast several hours in advance and is calculated in a few minutes.

Fig 4.1 Frankfurt Airport Passenger Simulation Model [31]

4.2 CAST Software

Airport Research Center is a company that uses its CAST software to model and simulate various airport environments. This software is used by different airport owners, airport operators, ground handling companies as well as consulting companies. CAST has several variants specific to the type of simulation required, among which are the following: Terminal and Aircraft Solution, vehicle ground handling, stand & gate allocation, among others. This type of simulation results in better optimisation of resources by improving capacity and reducing delays. This programme has been tested in several airports such as Vienna and Moscow-Domodedovo. The different solutions offered by the CAST programme will be explained in more detail below.

4.2.1 Terminal Solution

This solution is primarily designed to determine the capacity and requirements of the different processes within the terminal building. It can be of great help when planning, designing and optimising existing infrastructures and processes. The software itself runs the simulation, and at the same time performs a live analysis, calculating different parameters such as Leve lof Service (LoS), waiting times, occupation levels, among others.

4.2.2 Aircraft solution

This other software provides you with a simulation of the aircraft on the tarmac of any airport. The programme evaluates the airside and assesses the

infrastructure (where there are bottlenecks, planning errors and how to optimise the existing infrastructure), the operations (reducing delay and improving slot allocation) and the capacity. By using this programme, an airport can benefit from a more optimal use of its resources, so that capacity is improved and delays are reduced.

4.2.3 Vehicle ground handling

This solution consists of a fast simulation that evaluates the traffic and its processes of handling vehicles on the airside. It offers planning support and optimises the road network and the available equipment. It also informs about possible bottlenecks, resources required for the correct operation, a decrease of the handling delay and the comparison between different strategies so that the operator can choose the most favourable one.

4.2.4 Stand & Gate Allocation

This tool is a powerful aid for cost efficient resource allocation. Unlike the others, it can be used by airports, airlines and consultancy companies for a better management of gates and allocation tasks. Its main function is to optimise airport capacity by managing higher traffic flows and improving service quality by taking into account passenger comfort, airline preferences and operational objectives.

4.2.5 Others

The Airport Research Center (ARC) also offers other solutions with its CAST software, such as the simulation of vehicles on the landside, the simulation of pedestrians in areas that are not only airports but also in railway stations, football stadiums or other crowded buildings.

CHAPTER 5. SIMULATION

This chapter explains several aspects of the simulation created such as the program used and assumptions that have been made when designing the program (track configuration, taxiways addressing, simulation day, fast outputs, ...). Also in this chapter you can find three different resources used in the program. Afterwards, the different classes of the program are explained and then some of the main functions.

5.1 Programming environment

Nowadays there are many simulation programs, with different programming languages and which work in different ways. To carry out the simulation of this Final Thesis I had two simulation options. The first option is the simulation that works with discrete events, it models the operation of a system as a (discrete) sequence of events in time. This type of simulation is suitable to simulate the unloading of a truck in a warehouse. The other option is the continuous time simulation. This type of simulation makes a sequence of events in each time differential, it is often used to design a flight simulator.

5.1.1 Decision and its reasons

Finally, after analysing and doing small modelling with both types of simulation, I have realised that the most suitable for my case is the continuous time simulation. The main reason is because discrete event-based simulations are not the best for trajectories (space-time simulations), as opposed to continuous-time ones. This fact could be demonstrated in a basic example I did with the Anylogic program (based on discrete events). Moreover, it was a type of simulation not used in my Bachelor's degree, this could lead to a much longer and more difficult project than necessary. On the other hand, continuous-time simulation was used in a course.

5.1.2 Program used

Once I knew what kind of simulation I was going to use, I had to choose a language and a program to simulate it. For this I had to choose between taking an unknown program and learning it from scratch, or the other option is to use a program previously used in some degree course in order to go further and make my simulation as realistic as possible.

The result was using the C# Language in the Visual Studio application previously used in Informatica 2 course. As I said before, I chose this combination because I could develop a better simulator, and spend my time improving and making it more realistic instead of learning how a programming language works. Also, I have opted for this option because in the Informatica 2 course the project we did was an en route airspace simulator that prevented collisions between planes, although the simulator is different, I already had a solid base where to start and how to continue. Moreover, the Visual Studio program, as its name says, it is very visual and it can be really easy to follow the simulation and the airplane movements, thanks to the forms that can be created.

5.1.3 Assumptions

In this section I am going to talk about what assumptions I made during the development of the simulator. These assumptions are based on the airport we want to simulate, which in this case is Barcelona-El Prat, and the information we have about it to make the programme as realistic as possible.

5.1.3.1 Runway configuration

As explained in section 2.1.2, the vast majority of the time, the airport uses the west configuration parallel runways, for this reason, the simulator will operate only under this runway configuration, regardless of the hour of the day. If the result of the simulation is satisfactory, the next step to make it more realistic would be to program each and every one of the previously explained configurations.

5.1.3.2 Simulation day

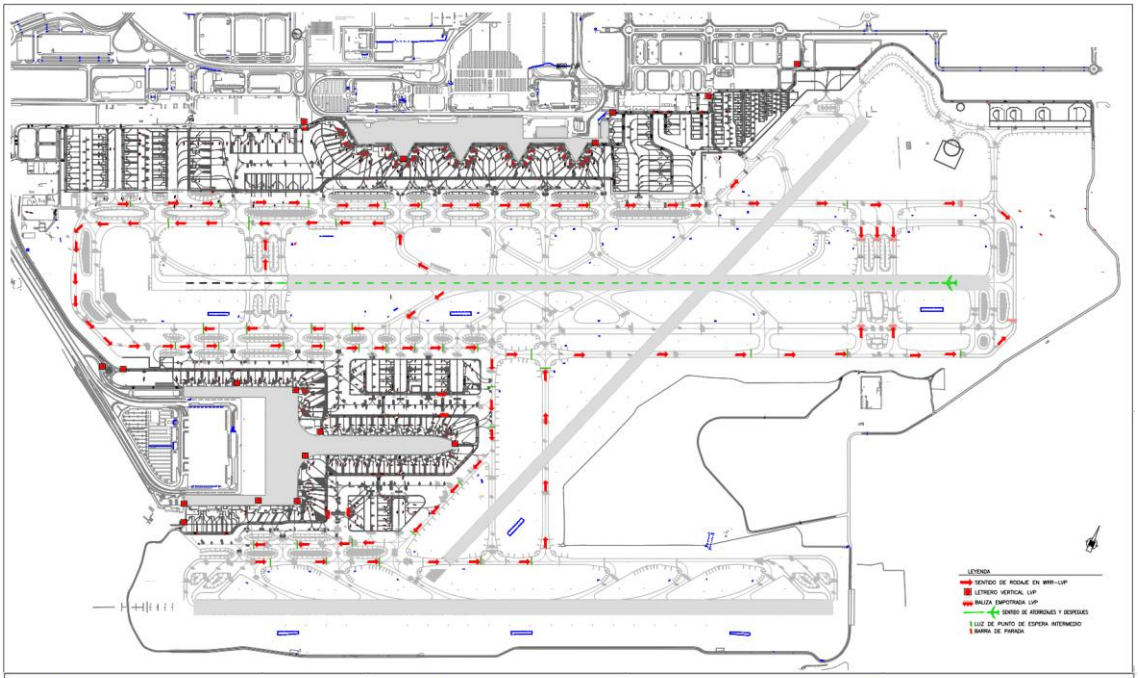
The day used for the simulation is very important to assess the performance of the code and the feasibility of the programme. For this reason, the day used has to be one of the most demanding in terms of traffic flow. As can be seen in section 2.1.2 of this report, the flow of traffic at Barcelona-El Prat airport is very seasonal, with the busiest period being the summer period. The day chosen is Wednesday 1 August 2018, an earlier date has not been chosen because the flow of movements at the airport has been reduced due to the pandemic.

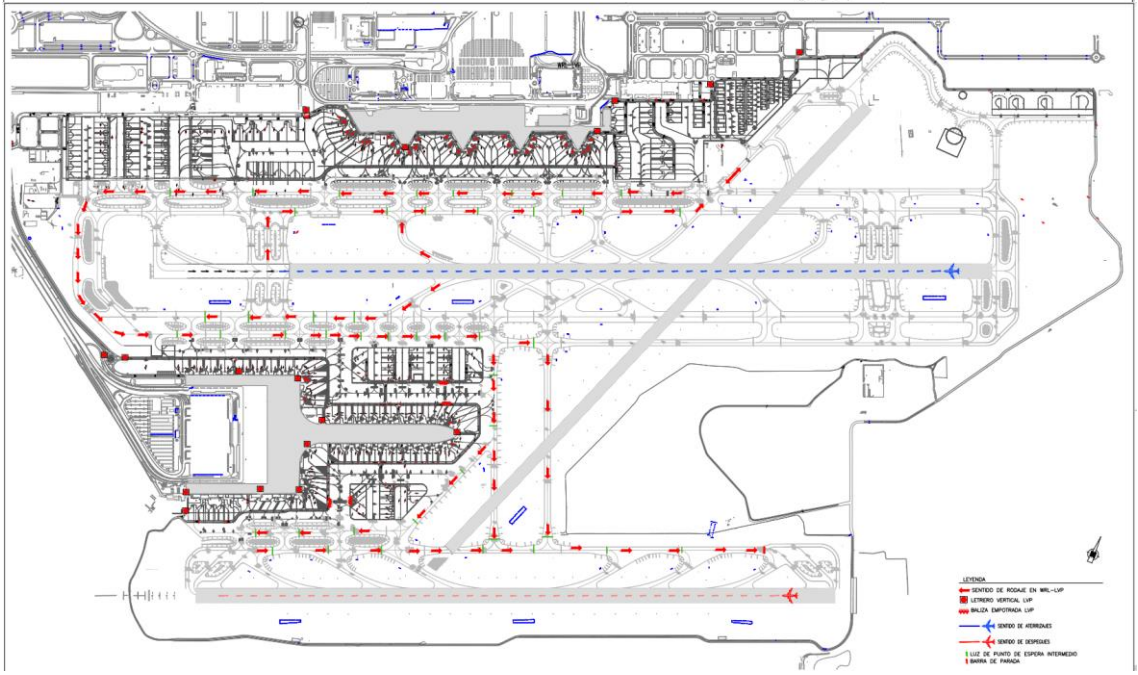
In addition, on this particular day I have been able to obtain information on the movements of the different aircraft, where they parked, the model of the aircraft, the departure and arrival times, ...

5.1.3.3 Taxiways direction

This point is crucial for the proper performance of the simulation. As explained before in the airport peculiarities, the same taxiway can have one or both directions depending on the runway configuration being used. In our case, we will operate all day on the west configuration of parallel runways, as a consequence some of the airport taxiways will be limited to one direction only. For this purpose, I have obtained information on the direction of the taxiways depending on which runways the different aircraft arrive and depart from. One problem I have had is that in this file the configuration to be used in the simulation was not implicitly shown. However, this document defines the direction of the taxiways when the planes take off on runway 25L and land on runway 25R, and also when they take off and land on runway 25R (this configuration is used when a large plane cannot take off on runway 25L). If these configurations are combined, the desired configuration is obtained.

Fig. 5.1 shows the two different configurations for which I have information, and the result obtained when we merge them together. Taxiways without an arrow mean that they are bidirectional. As can be seen, taxiways that are closer to the gates are bidirectional, while those closer to the runways are unidirectional. The merger shows that mainly the direction of the arrows of the west configuration has been kept because this will be the predominant case. However, it should be noted that the arrows go all the way to the threshold of runway 25R and the change of direction of the "E" taxiways from going towards runway 25L to go towards runway 25R, these taxiways will be used mainly by aircraft parking on stands on the south dock and part of the central dock and which need runway 25R to take off.





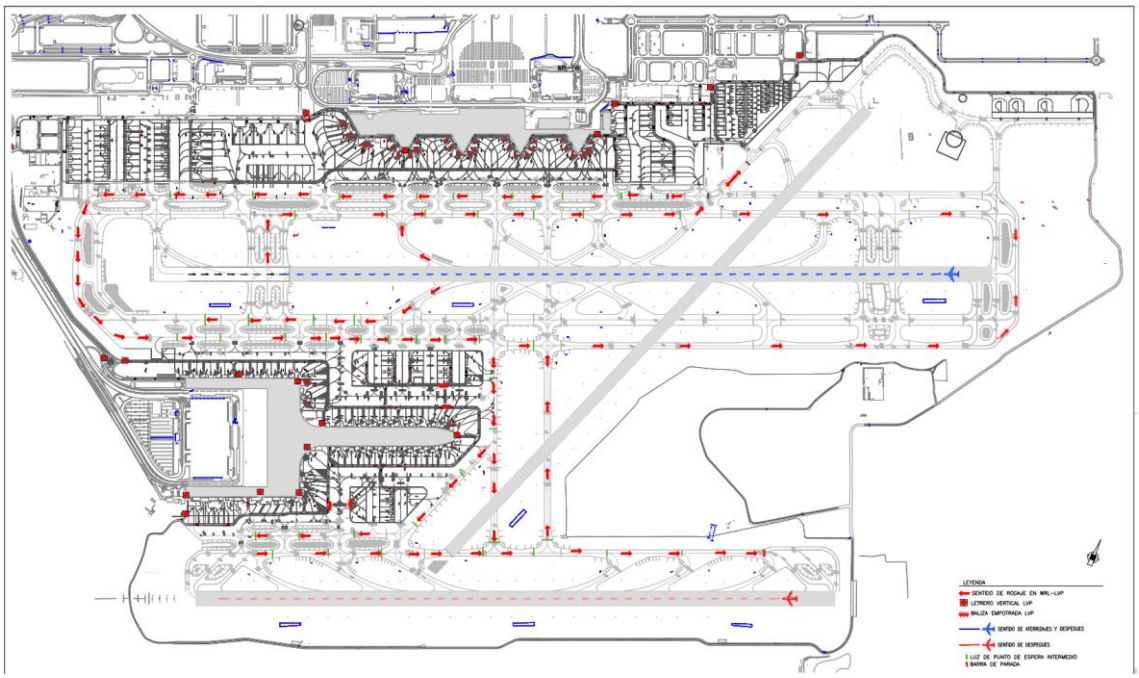


Fig. 5.1 Arrivals & departures from 25R, arrivals 25R & departures 25L, merging both options [23]

5.1.3.4 Gates

The gates used for the simulator have been obtained from the Barcelona – El Prat view in Google Maps (see in [22]), as it can be seen in the previous Fig. 2.11, and from the AIP of the airport (see in [16]). Google maps has been used because AENA does not provide enough information about the stands, their location and the pushback process. For the simulator, we have used the same name of the stands that appear in the AIP, their corresponding location, and tried to carry out the same pushback route. Thus, the movements at the gate are not the exact ones, but they are approximately the same.

5.1.3.5 Rapid Exits

Annex VIII of the AIP of Barcelona-El Prat airport (see in [20]) has been used to schedule the rapid exits taken by each aircraft. In section 20 of Annex VIII, entitled Local Regulations, there is a section dedicated to Minimum Runway Occupancy Time where there is the table shown in Table 5.1, which explains for each of the runways which rapid exits there are, the types of aircraft that can use them and the distance of each one from the runway threshold. In order to use one of the rapid exits, the aircraft has to fulfil the following two conditions:

- The type of aircraft it is, i.e. the category of the aircraft (see in [33]), if it is light (<7000kg), medium (7000kg < weight < 136000kg) or heavy (>136000kg), allows it to use the rapid exit.

- The landing distance is less than the distance between the runway threshold and the rapid exit (this distance is shown in the fourth column in Table 5.1). For this purpose, a research has been made on all the aircraft models operating on the day of the simulation and specifications of each of the models have been searched, among which there is the landing distance.

RWY	SALIDA RÁPIDA / RAPID EXIT	ACFT	DIST FM THR (m)
07L	P2	ligeras y medias / light and medium	1305
07L	P4	ligeras / light	945
07L	R2	ligeras y medias / light and medium	1051
07L	R4	ligeras / light	751
07L	P1	todas / all	1864
25R	P3	ligeras y medias / light and medium	1275
25R	P5	ligeras y medias / light and medium	1617
25R	P6	todas / all	2112
25R	R3	ligeras y medias / light and medium	1409
25R	R5	todas / all	1703
25R	R6	todas / all	2053
07R	G4	todas / all	2053
07R	G5	todas / all	1703
07R	G6	ligeras y medias / light and medium	1402
25L	G7	ligeras y medias / light and medium	1402
25L	G8	todas / all	1703
25L	G9	todas / all	2053

Table 5.1 Rapid exit depending on the category of the aircraft and its distance from the threshold

5.2 Resources used

In order to carry out the simulation, a number of resources have been used to read and write the information, as well as for the routing of the different aircraft. Each of the resources used will be explained in detail below.

5.2.1 Dijkstra

This resource is the most important of the whole simulator and the basis of the simulator, thanks to the Dijkstra algorithm the aircraft can create the most optimal route from the runway/stand to their destination. The code used in the simulator cannot be just any code from the Dijkstra algorithm for C#, and the main reason is that not only do we want to know the cost of going from point A to point B, but also in this case it is essential to know the path that has been used and through which points it has passed to go from A to B. Taking into account the necessary conditions for the Dijkstra algorithm that the simulator needs, the code used has been obtained from the Geeks for Geeks web page (see in [34]). For the use of this code, one of the necessary inputs for the code to work is a matrix containing all the points and the cost of going from one point

to another. Other inputs that are needed are the departure point, that is, the origin of the route and also the final destination will be needed to find the route. The matrix will have the following form as can be seen in Fig. 5.2. The values of the diagonal will be 0 because it is not possible to go from a taxiway to itself. The values shown in the matrix are the cost, which in the case of our simulator is the time in seconds. Using as an example the matrix in Fig. 5.2, the value of "a_{31}" would be the time in seconds to go from point 3 to point 1. If these are not connected, this value will be 0, otherwise this value will be the time in seconds it takes to travel the distance from 3 to point 1.

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ \vdots \\ m \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & \dots & n \end{array} \\ \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ a_{31} & a_{32} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \end{array}$$

Fig. 5.2 Matrix nomenclature

The airport has been divided into 3 categories: stands, taxiways and runways. For each of the categories we have the start and end coordinates, where for adjacent taxiways these coordinates have to match, name, time needed to go from the beginning to the end and a different value that corresponds to their position in the matrix, among others. A taxiway has been created for each intersection and for each abrupt change of angle, i.e. for an airport taxiway the simulator may have divided it by 2 because it intersects with another taxiway. An example of this situation can be seen in Fig. 5.3, where taxiway J7 intersects taxiway Q7. In this case, taxiway J7 is composed of taxiway J7_sup (which runs from the northernmost part to the intersection with Q7) and taxiway J7_inf (which runs from the intersection with Q7 to the beginning of taxiway J8).

Fig. 5.3 Taxiways around T1

In the case of stands, the programme looks for the nearest taxiway, taking into account the pushback movement, to put in the matrix the respective time it will take to get to that taxiway.

Finally, in the case of the runways, there are programmed a number of taxiways of entry and quick exits that can be used.

In total, the matrix used to find the optimal route is 496x496 in size. The values that make up the matrix vary over time depending on the traffic on each taxiway, which means that at each time differential the matrix is updated to try to better define what is happening at the airport at any given moment.

Right now, with this single matrix, aircraft have no movement restrictions as long as the taxiways are connected. The problem that happens is that in reality an aircraft cannot use a specific taxiway in the direction that best suits it, some of them have directionality that depends on the runway configuration that is being used at that moment. It is for this reason that a multiplicity matrix is needed where all values will be 0, except for the positions where movements are allowed, which in this case will be 1. The matrix resulting from multiplying point to point the original matrix with the multiplicity matrix will be the one used by Dijkstra's algorithm to find the optimal route for each aircraft at each moment. Unlike the original matrix, the multiplicity matrix does not need to be updated at each time difference because the runway configuration is always the same.

As an example, Fig. 5.4 shows the taxiways between terminal T2 and runway 07L-25R. If we define that the taxiways that form taxiway "S", are unidirectional and their direction is to the left, in Fig. 5.4 it can be seen which are the movements restricted by the multiplicity matrix, the restricted movements have been assigned a very high value in the multiplicity matrix so that the cost of going there is very high, causing it to be an unused route. As explained above, not only is the opposite direction of a taxiway restricted, but also the possibility

of going from a taxiway perpendicular to the opposite direction of the designated taxiway.

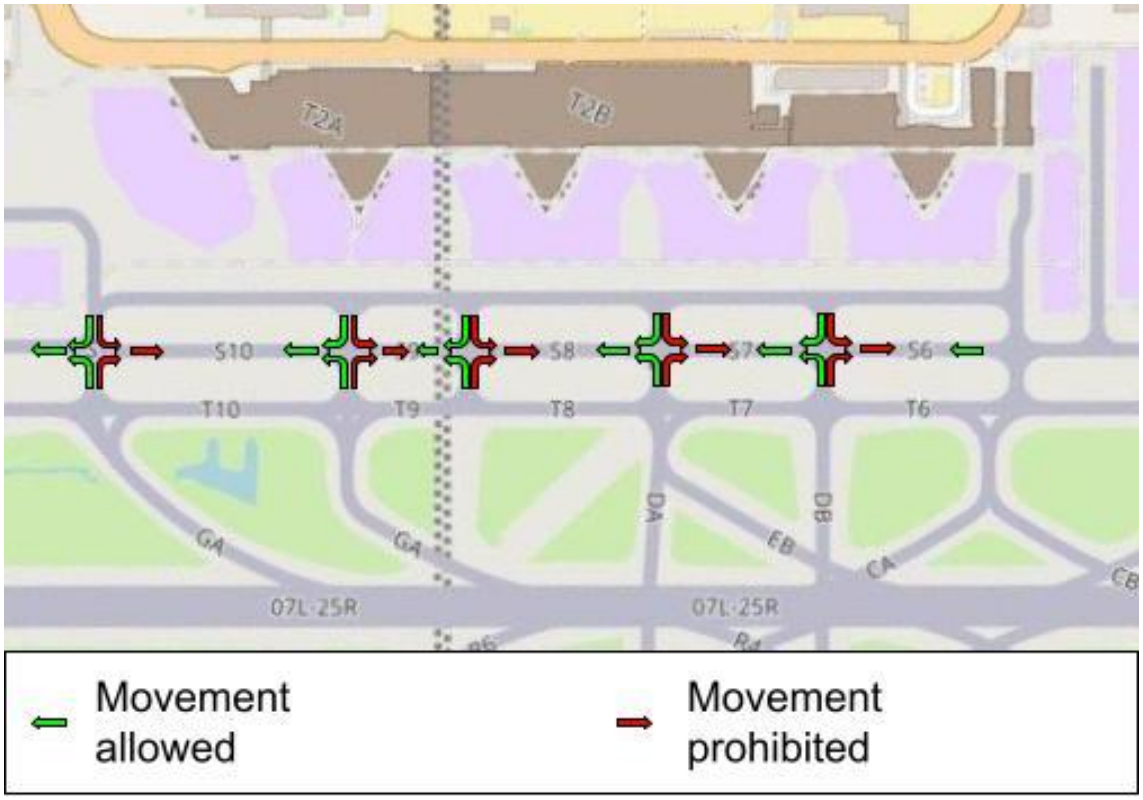


Fig 5.4 Movements allowed vs movements prohibited

At the end of using Dijkstra's algorithm, the values returned are the total cost of the route and also a list of numbers, where each number corresponds to the position within the matrix, i.e. a stand, taxiway or runway, thus detailing the optimal path that the algorithm calculated. This list is used to create the aircraft route and the total cost is used only for departures to know at what time the aircraft has to leave its stand.

5.2.2 Opening, reading and writing .txt and Excel documents

The information of each aircraft is contained in an Excel file, to obtain it and pass it to the simulator, the simulator has to be able to open, read and save in the program all the necessary information for the correct operation. Apart from the aircraft information, I have also created two more excels, one with the start and end coordinates of all the taxiways and runways with their respective names, and the second excel is of the different stands that are in the airport, where it contains information of their name, location and pushback route. For this I have used the code found in the Coderwall webpage (see in [35]), where I have modified it to read in the best way each of the different excels (see in Annex A.3).

Apart from opening, reading and saving the information in the simulator, the program also contains a function to create a new excel and save all the variables of each aircraft, to later analyse the performance of the simulator and obtain conclusions. For this second task we have moved to another webpage because the previous one was only to read an Excel file. In this case, as previously mentioned, we have to create a new Excel file and write in it, using the code from DelftStack webpage (see in [36]) and adapting it to store the aircraft variables in the same row, but each variable in a different cell (see in Annex A.2).

5.2.3 Greenshields formulation

In order to know the optimum route at any given time, the traffic conditions at each of the airport's taxiways had to be taken into account. In order to monitor the status of a taxiway and the traffic, a list of aircraft in each taxiway was created, showing which aircraft are currently on the taxiway. Each time an aircraft enters, it is added to the taxiway's aircraft list, and when an aircraft leaves the taxiway it is removed from the list. In this way we can know which aircraft is/are on which runway. In order to determine that a plane avoids a queue, a function has been used that varies the speed of the taxi and therefore the time it takes to go through the queue. This means that in the Dijkstra matrix previously mentioned, this value will be increased and possibly the optimal route will avoid the traffic. To determine the speed as a function of the number of aircraft present, the linear interpolation shown in Fig. 5.5 is used. This function has been chosen because in 1935 it was possible to describe the traffic on a road with Greenshields formula. This function determines the speed of a car from the density of that section of the road. In Fig. 5.5 you can see the Greenshields formulation in a graph together with the equation that describes the speed from the density.

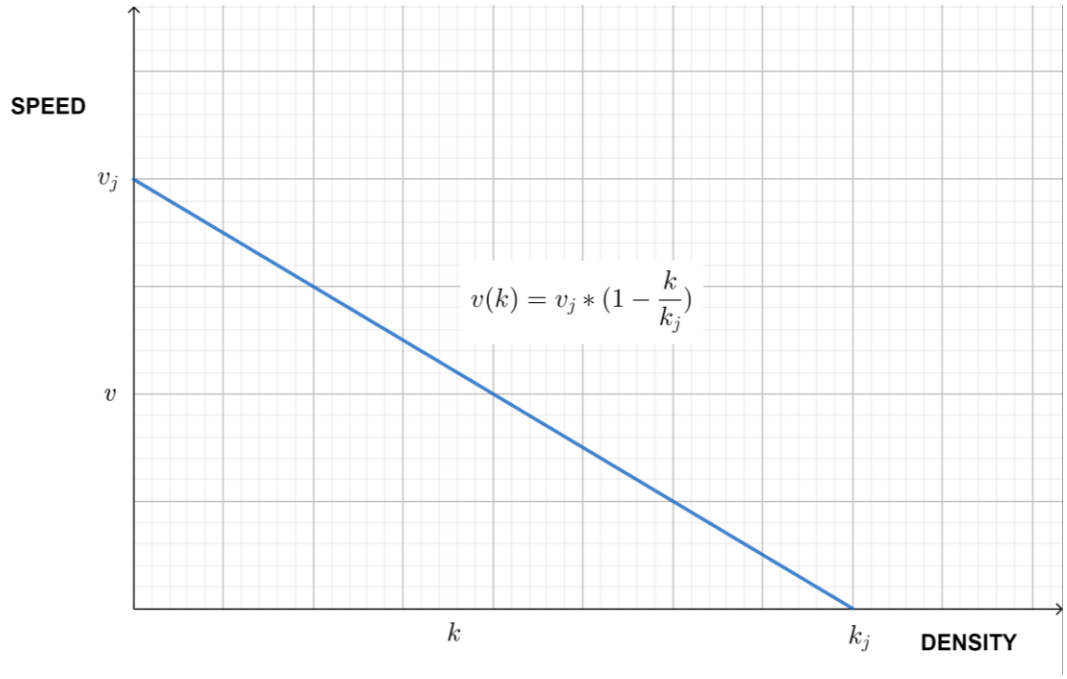


Fig 5.5 Greenshields formulation

The maximum speed corresponds to the maximum speed when there is no aircraft or only 1 aircraft, and is 15 knots. The distance that belongs to this point is 2 times the length of the aircraft, since it is the aircraft plus the theoretical separation of 1 aircraft between 2 aircraft. The critical point is when the sum of all the planes plus the separation between them is equal to or greater than the total distance of the taxiway, in this particular case the speed would be zero and therefore the time to travel the taxiway would be infinite, making the taxiway infinite.

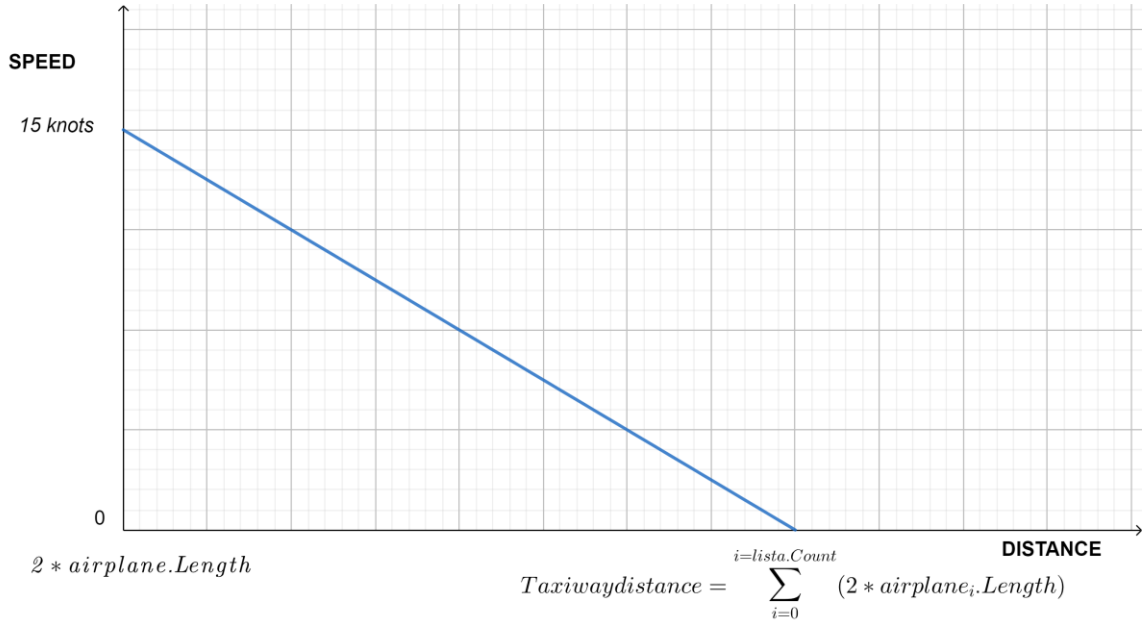


Fig. 5.6 Greenshields formulation adapted to the project

5.3 Classes in the programme

As explained above, the programming environment used for this project is Microsoft Visual Studio, and the programming language is C#, this combination is the same as the one used in the project of the "Informatica 2" course. Each of the different categories of this simulation is coded in a different class. The different classes used and the function of each of them are explained below.

5.3.1 Simulator

This class is one of the most important. This is where all the loaded information about the different taxiways, runways, stands and all aircraft is stored. It is also where all the functions of the different buttons on the simulator screen are coded. It also contains the function that makes the simulation run, which is the Timer Tick function. This function is explained in more detail in Section 5.4.1. This class is composed of about 1600 lines of code.

5.3.2 Runway

This class contains all the information related to the runways, such as the name, its start and end and whether it is occupied, among others. The only remarkable functions in this class are the one used to create the runway and the one that calculates its angle with respect to north, which is then used to calculate the new coordinates of the aircraft at each time differential.

5.3.3 Taxiway

The taxiway class is very similar to the previous one, it is created to store the information of each taxiway. This information consists of its name, start and end coordinates like the runway class, but it also has some more variables such as the time needed to cross the taxiway, the speed at which the planes must move in that taxiway and a list of the planes that are in that taxiway at that precise moment.

The functions that can be found in this class are the one that creates a taxiway, the one that calculates its angle with respect to north and the function that calculates the aircraft density from the list of aircraft and the length of the taxiway, depending on the value obtained modifies the circulation speed. This last function is explained in more detail in the previous Section 5.2.3.

5.3.4 Stand

This class contains all the information concerning each of the different stands (its name, location, pushback trajectory, whether it is occupied, and to which terminal the stand belongs). In this class there are only two functions, the one that creates the stand from the information in an excel file and the one that determines to which terminal the stand belongs from its name.

5.3.5 Dijkstra

The Dijkstra class is one of the fundamental pillars of this project. In this class there are many more functions, the main ones that do the algorithm and calculate the optimal route are obtained from the Geek for Geeks webpage (see in [34]), but we can also find other functions such as the one that creates the cost matrix and the multiplicity matrix and the function that multiplies these two point by point to obtain a final matrix that perfectly represents the possible paths and their associated cost, which will be used later in the algorithm. It must be emphasized that the code of these two matrices is the longest and most complex of this class and its code is specific for the case of the Barcelona - El Prat Airport and more specifically for the configuration of runways used. If the scenario were to be changed, this part of the simulator code would be one of the few that would have to be modified due to the fact that we wanted to make it as general as possible so that it can be used in any other airport. Then we can also find functions that convert the format of the result obtained in Dijkstra into a list of taxiways that make up the optimal path to go from point A to point B.

This entire class with its 700 lines of code is attached in the annexes, the main reason is because I wanted to show the final result of one of the most complicated parts of the program, and how I have solved some problems I have faced (not all Dijkstra codes work because in this case we needed both the cost and the path, how to declare automatically each of the taxiways, how to do the routing, ...). Besides, there is another reason to attach this part of the code, and that is that I wanted to show my programming style, and this case is perfect because there is code from the internet specially adapted for this project, for example the Dijkstra code, and there are also many new functions created from scratch, such as the functions to create the Dijkstra matrix. Moreover, this part of the code is the most independent and is the easiest to understand without attaching other classes of the program. It should be noted that the Dijkstra class is the base of the simulator, without it there would be no routing of the aircraft and therefore the main objective of the project, which is to replace the ground controllers, would not be possible.

5.3.6 Route

The route class is the simplest of all. This class is used to know the start and end of the route and which taxiways to pass through. For this, there is only one function and it is the one that creates the route itself from the values that enter it.

5.3.7 Model

The model class is where the information of the different aircraft models that operate at the airport is stored. This information consists of the name of the model, the dimensions of the aircraft and its main characteristics (takeoff and landing distance, aircraft category and aircraft weights). In this class we only find two notable functions, the first is to create a model, this function has as inputs all the information that you want to store from a specific model. The other is to determine the category of the plane from its takeoff weight. The main reason why this class is necessary is due to the fact that not all airplanes can take-off from the 25L runway, depending on the model, they need a larger runway such as the 25R. The models that need to take-off from the runway 25R on the day of the simulation are the following Airbus models: 332, 333, 346, 359, 388 and the models of Boeing 763, 767, 772, 777, 788 and 789. Moreover, the specifications of each airplane are also needed to determine which rapid exit they could use. To determine the rapid exit is needed the landing distance and the category of the airplane that depends on its weight.

5.3.8 Airplane

Finally, there is the aircraft class, which together with the simulator class is the most important class. In this class there are different types of variables. The first type of variables has all the information of the aircraft from Excel, such as the callsign, the stand where it will park, the model of the aircraft, the airport where it comes from and its next destination and the arrival and departure times. Then, we find variables that are used for the simulator such as the route it has, its speed, the current location, which procedure it is performing and a list of the different runways and taxiways. And finally, the last category of variables are for further analysis of the behaviour of the simulator.

This class is the longest of all, with more than 1700 lines of code and 57 different functions. Among the main ones are the creation of an aircraft, the main movements of an aircraft (Landing, Take-off, pushback, taxiway movement), functions to know when to enter the runway or which rapid exit to choose and above all functions for conflict detection and collision avoidance. This is the longest class of all because it contains 1895 lines of code.

5.4 Main functions

As it has been explained in the beginning of this chapter, a continuous time simulation will be used, to make this type of simulation using the C# language and the Visual Studio programming environment, a tool called timer will be used, where you can program how often a tick is made. In each tick is where the program uses its main functions to move the planes. The most important ones are explained below.

5.4.1 Timer tick

On each tick of the timer, a list is scrolled through containing all the aircraft that will operate. For each aircraft, the program looks at the situation it is in through its variables to find out which process it is in (landing, taking off, moving through the taxis, doing the pushback or staying still in its stand) and uses its corresponding function. Once it has run its function, some of the aircraft's parameters are updated, such as the coordinates of its position, which taxiway it is in or its speed, among others. In the case of the position, once the new coordinates have been updated, the program will move the plane's image to its new coordinates, and if they correspond to a new taxiway, the list of planes in each taxiway will also be updated, eliminating the plane from the list of the old one and adding it to the new taxiway.

5.4.2 Landing

This function is used when the local time of the simulator matches the arrival time of the aircraft. At this precise moment, an aircraft (magenta dot) appears on the right side of the screen which is about to land on runway 25R. A remarkable aspect of this function is that it calculates the deceleration that the aircraft needs to go from an approximate speed of 135 knots (at the runway threshold) to the 15 knots at which it is moving on the taxiways, therefore the movement of the aircraft is more realistic. This deceleration is calculated only once, the first time this function is used, and this value is used for the whole landing procedure, so we are using a Uniformly Rectilinear Accelerated Motion (URAM), and the function to calculate the acceleration is the one shown in Fig. 5.7. This formula is used because we have the total distance to travel and the initial and final velocities.

$$acceleration = \frac{(velocity^2_{final} - velocity^2_{initial})}{2 * distance_{total}}$$

Fig. 5.7 Acceleration formula

This function is used until the aircraft reaches the start of its rapid exit, which has been previously found using the values of the aircraft model (weight and landing distance). Once it reaches the start of its rapid exit, the aircraft will change its current state and from that moment on it will no longer use the landing function but the taxiway movement function.

5.4.3 Taxiway movement

This function is used once the aircraft has finished landing to go to the assigned stand, just before performing the pushback procedure, or on the contrary, to go from the stand, just after making the pushback movement to the take-off runway (which as previously explained can be either the 25R or the 25L depending on the aircraft model).

In the function that does this type of movement, before calculating the new coordinates of the aircraft, it checks if there is any conflict. In this case the possible conflicts are two, the planes that are also taxiing and the planes that are doing the pushback. For this second category, they are only affected if the aircraft is in a taxiway adjacent to a stand. The aircraft will stop if it comes within a certain distance of a pushback aircraft. For the first category, the aircraft will stop if it is in one of the following scenarios:

- If there is someone in front: In this scenario, if a plane is in front of another plane and is less than the distance of the length of the plane in front of it, the plane will stop.

- If you are arriving at an intersection and someone is already in the intersection or a priority aircraft is also arriving, it will stop to yield to you. To determine priority, taxiways closer to the runway are given priority over taxiways further away from the runway.

Once it has ascertained that there is no potential for collision and therefore the aircraft will not have to stop, it looks to see what situation it is in. The most common situation is that it follows the trajectory of the taxiway it is currently on, calculates the distance travelled in 1 second and updates the new coordinates. If this distance is greater than the remaining distance to the end of the taxiway, the program calculates the remaining distance to travel in that second and applies it to the next taxiway on the path previously calculated with the dijkstra algorithm. Once the end of its route is reached, it means that the aircraft is on the runway or is at the start of the pushback procedure.

Once they are already on one of the three taxiways previously mentioned, the aircraft that will take off will first check if the runway is occupied or not. A runway is occupied during the first 90 seconds after take-off, this is the capacity of Barcelona - El Prat airport in the west configuration parallel runways. Once

these 90 seconds have elapsed, the next aircraft can enter the runway and begin take-off.

5.4.4 Pushback procedure

This function is used to park and unpark from a stand. In the case of parking, this function starts when the aircraft is on the last taxiway of its route and the start of the entry route to the stand is on that taxiway. The same thing happens for disappearing, but the procedure is the other way round, but in this case instead of taking the last taxiway on the list of our route, it is the first one and also before leaving the stand, the aircraft checks if there is any aircraft on the taxiway at the end of the pushback route to avoid collisions with it, if there is no aircraft, it will carry out this procedure.

The entire pushback movement is carried out at a speed of 3 knots, which is the speed currently used for this type of procedure at Barcelona-El Prat airport.

5.4.5 Take-off

This function simulates the take-off of an aircraft. The first time it is used is when the runway is free and the plane is at the runway threshold, in this first iteration, the runway status is updated to occupied and the code calculates the acceleration needed for the plane to take off, and in each iteration that acceleration causes the speed to increase, making the take-off more realistic. Once the aircraft leaves the screen, the system deletes the aircraft from the list of aircraft that is run through each tick of the timer. This has been done to reduce programme memory and increase the speed of the process at each timer tick. After 90 seconds of the first iteration, that is to say the start of take-off, the runway is cleared for the next aircraft to take off.

5.5 Operation

Once the simulator is ready, the first step is to press the "Load All" button. By pressing this button, it will start a loading process from Excel files containing the information of all aircraft that operated on August 1, 2018, the characteristics of the models of these aircraft, and also the coordinates of the different taxiways, runways and stands that have the airport of Barcelona - El Prat. The whole procedure takes about 3 minutes. When the loading process is completed, the next step is to check the departure and arrival times of the different aircraft. This step is necessary because different planes are assigned the same take-off or landing time. To do this, a function has been created to separate the landings and another one that does the same but with the take-offs. In the case of landings, the function goes through the whole list of planes and for each plane looks if there is any scheduled landing in less than 100 seconds, this value is

due to the capacity of landings at the airport of Barcelona - El Prat, if it were the case, the arrival time of the plane would be modified to 100 seconds later than the landing time of the plane that precedes it. The function for separating planes on take-off is exactly the same, with the only difference that now the take-off time is taken into account and that a plane can leave the airport every 90 seconds. With these functions we allocate slots for departures and avoid queues at taxiways before a take-off runway because several aircraft have the same ETD. To finish the process of loading all the information, the last step is to load the different planes in their respective positions. To do this we look at their arrival and departure times, and depending on these values we deduce whether the plane has already left, is arriving or is parked in its stand.

Once all the information necessary for the correct functioning of the simulator has been loaded, the only thing left to do is to press the "play" button. Pressing this button activates the timer with a start time of 0:00h on the 1st of August 2018. From here on, and as explained above, in each tick of the timer a function is used which is the basis of the program, since depending on the situation of each aircraft, one function or another will be called to update its status. This function starts by going through the list of the different planes and for each plane the first thing it looks at is whether it is in its assigned stand and if it has a pre-established route. This step is done to approximate the time it takes to get from its current location to the runway, because the departure time assigned to each aircraft is the time it has to take off rather than the time it leaves the stand. For this purpose, Dijkstra's algorithm is used and the value we are interested in this time is only the cost of the route, which is the time in seconds. This process to know the time to go from the stand to the track is only calculated once. This time will be stored in a variable and each iteration will check if the current time plus the calculated time gives a time equal to or later than the scheduled take-off time. If this is the case, the route is recalculated just before leaving the stand in order to have the most optimal route and hence avoid possible congestion areas. This time, when applying Dijkstra's algorithm, what we are interested in is the path, that is, which taxiways the plane has to pass through to reach the take-off runway. As mentioned when describing the aircraft movement function, for take-offs on runway 25L, there are three taxiways that reach the threshold of the runway, the different paths are used according to the priority of each aircraft. Therefore, for aircraft using runway 25L for departures, the route calculated by Dijkstra will reach taxiway K2 as shown in Fig. 2.13, and once it reaches this point compare which aircraft are ahead of it in order to know their priority according to their departure time and consequently which taxiway they will enter the runway.

Once these previous procedures are done in the first iteration of the timer tick, the function looks at the situation of the aircraft, which can be several: landing, taxiways movement, pushback movement and take-off. These functions have already been explained above. Once the entire timer tick function has been run through, one second is added to the current time, updating the displayed clock.

When 24 hours of simulation have elapsed, that is at 0:00h on 2 August 2018, the program saves in an Excel file all the variables of the different aircraft that

have operated that day at the airport, in order to subsequently analyse the behaviour of the simulator and obtain conclusions.

CHAPTER 6. Functionality and mock-up

In this chapter you will find an explanation of the design of the simulator, with an explanation of all the functionalities of the program and what each button does. Finally, it explains step by step how to make it work.

6.1 Design

The application design is very simple, it only consists of an image where one can observe the movement of the airplanes and some buttons: Loading, play and pause buttons. If you click on an aircraft in the image, information about the aircraft is displayed as well as other buttons to give power to the controller.

6.1.1 Mock-up

In the Fig. 6.1, one can observe the mock-up of the simulator. Most of the screen is occupied by an image of Barcelona airport. The image is a map of the entire Barcelona airport, which includes all the taxiways with their corresponding names written above each one. In this part of the simulator is where the different planes will be shown, simulating their movements and behaviour with respect to the other planes.

Fig. 6.1 Simulator mock-up

Centred at the top of the simulator, a clock shows the time at which the simulation is taking place. Each tick represents 1 second in the simulator, and this does not correspond to 1 second in real life, as it is programmed to go faster. However, the simulation speed can be changed with the buttons shown in Fig. 6.2. The button on the left will be used to pause the simulation, the one in the middle to go to real speed (1 tick every second) and the one on the right to go to accelerated speed.

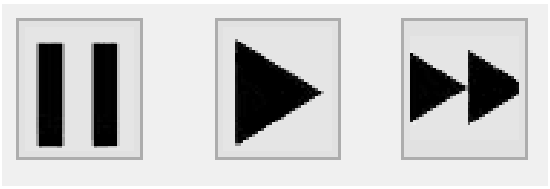


Fig. 6.2 Play and pause buttons

Apart from the picture, the clock and the play and pause buttons, one more button can be seen. The load button can be distinguished because "Load All" is written over the button. Once this button is pressed, all the information necessary for the correct functioning of the simulator is obtained. In addition, there is an area where all the information of an aircraft is displayed once you have clicked on the desired aircraft. At the same time as the information is displayed, 3 more buttons and a combo box also appear. These buttons are

made to give power to the controller, as they can stop, resume an aircraft and change its route.

6.1.2 Loading button

When one presses this button, the application loads all the information from an Excel file which contains data from each Flight on the 1st of August of 2018. This information includes the airport of origin and destination, the arrival and take-off times, the aircraft model and in which stand the aircraft will be parked.

Due to the fact that the Excel file contains a lot of information about each aircraft and there are also more than 600 aircraft to load, this phase takes some time, approximately between 2 or 3 minutes. One can know that this phase has been completed when small dots of different colours appear on the simulator as shown in Fig. 6.3. Each colour represents a different stage of the aircraft. If the colour is black it means that the aircraft is stopped at its gate, if the colour of the dot is magenta it means that the aircraft is in the process of landing (between the landing runway and its corresponding gate), or if the colour is blue it means that the aircraft is in the process of take-off (between its gate and the takeoff runway).

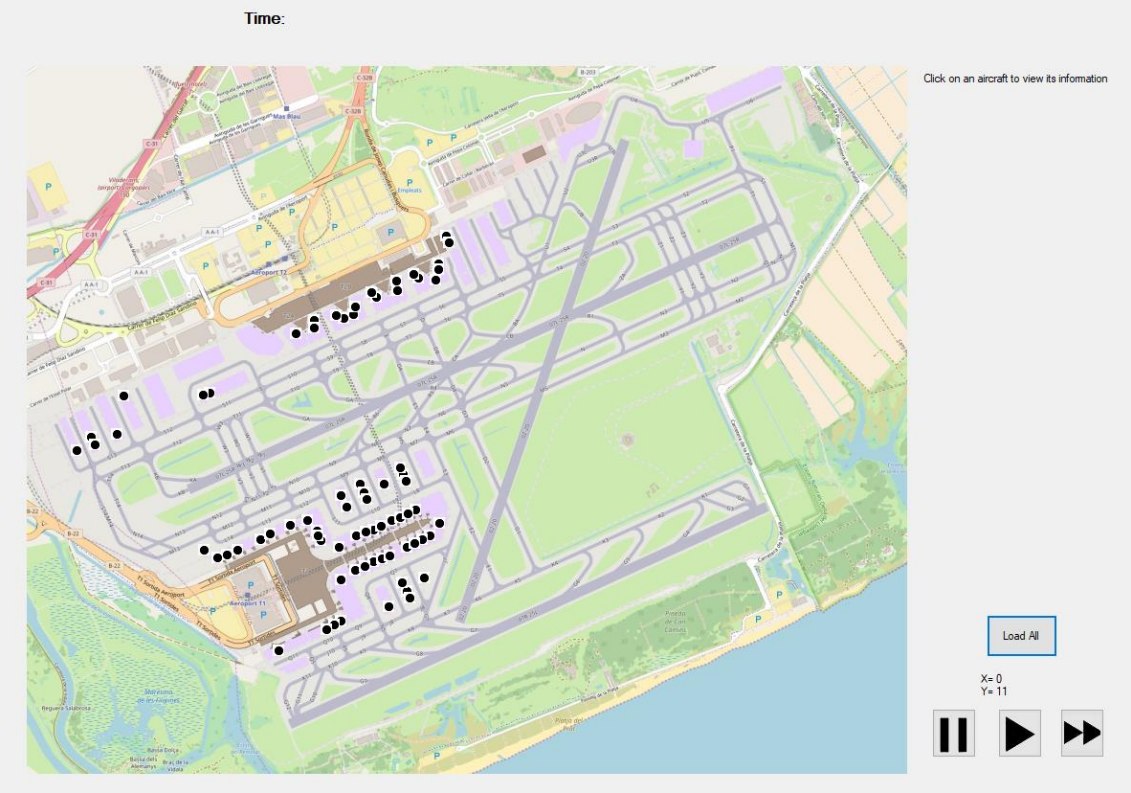


Fig. 6.3 Data loaded

In the Excel that gives the input data for the different aircraft, it has been observed that it is common that more than one aircraft has been assigned the same departure or arrival time. That is why at the end of loading all the information we run a couple of functions that separate the aircraft in arrivals and departures taking into account the capacity of the airport, which is currently 1 take-off every 90 seconds and 1 landing every 100 seconds. With these functions, we allocate slots for departures and avoid queues in the taxiways before a runway take-off because several aircraft have the same ETD.

6.1.3 Play and pause buttons

The play button is used to start the simulation. Once the button is pressed, on the upper side of the screen, one can observe the time of the simulation, as shown in Fig. 6.2, and how the time is running. In the program there are 2 speeds available, the first one is a normal speed (x1), this speed makes that 1 second in the program corresponds to one second in real life. The second option is the accelerated speed, which corresponds to the button on the right side of the screen. When this button is pressed, the simulation speed is much higher than the real time.

6.1.4 Airplane Information

On the right side of the screen, next to the map of the Barcelona airport and above the buttons, a sentence can be seen that says: “Click on an airplane to view its information”. Below this sentence there is an empty space that when one clicks on a plane it fills with all the information of that plane, the Fig. 6.4 is an image that shows how the information will be displayed. Depending on the phase of the airplane (landing, on a gate or taking-off), the represented information will be different. In the case that, when the airplane is landing or taking-off, the image is magenta and blue respectively, apart from the information described in the Excel (Arrival Callsign, Origin airport, ETA, Departure Callsign, Destination Airport, ETD, the airplane model and the stand that will use), it will also show through which taxiways the plane is going to pass, and to be more representative, the path of where the plane will pass will be shown in the airport map using black lines, as shown in the Fig. 6.4. These lines will be erased as the aircraft passes over them. On the other hand, the airplane is in a gate (the image is a black circle), the information about what Taxiways will use, is not shown.

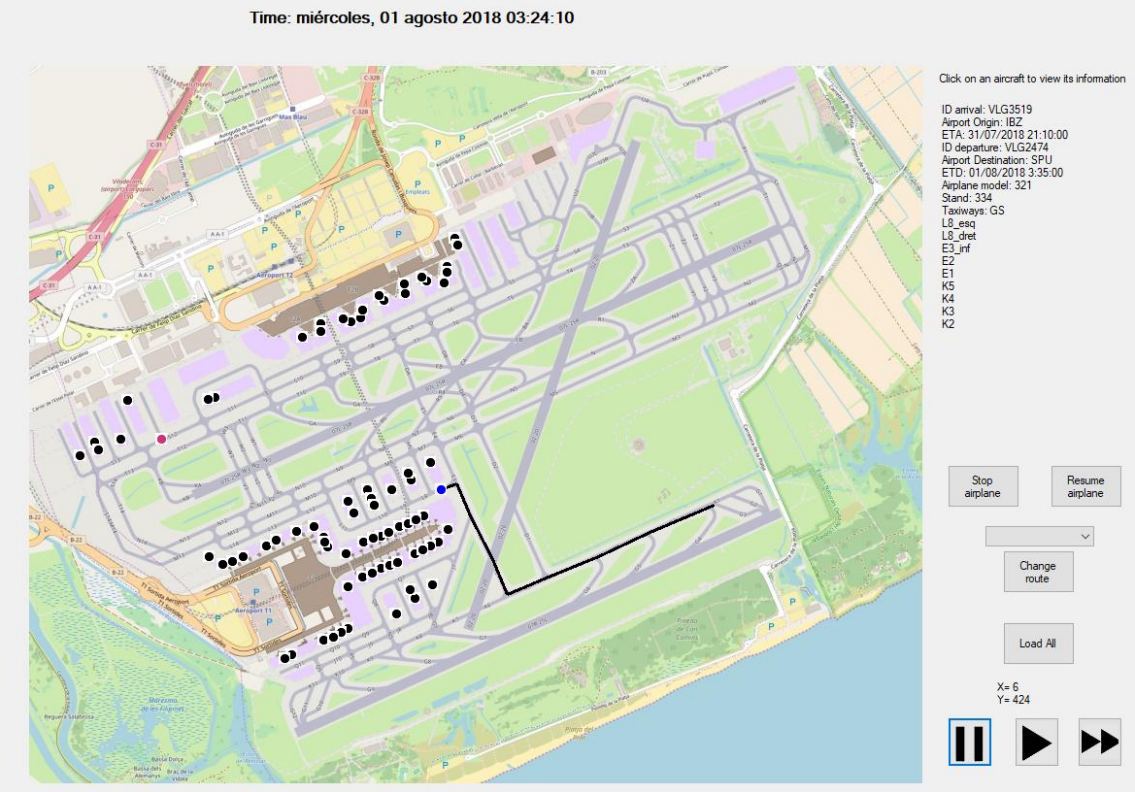


Fig 6.4 Airplane information

6.1.5 Controller buttons

As explained in the mock up sections, there are 3 buttons and a combo box that appears when one clicks on an airplane, until there they are hidden. These buttons are made specifically to give some sort of power to the controller, and make some decisions if he or she doesn't agree with the programme. As one can observe in Fig. 6.4, the upper left button, stop airplane, stops the airplane previously clicked. The button on the right-hand side makes that the airplane resumes its route. And the last button creates a new route to its destination but passing through the taxiway previously selected in the combo box above it.

6.2 How to use it

As I have explained above in the mock-up section of the application, this simulator contains only one window. Its functionality is very easy and interactive. In order to view the simulation, one needs to first load the Excel file that contains all the information about the movement of Barcelona – El Prat Airport on 1 August 2018. To do this one has to press the button that says: "Load All".

Then, once the application has loaded the file and the aircraft are shown as points on the airport map, the next and last step to run the simulator is to press the button to run at normal or accelerated speed. Once this step is done, one can observe how the clock time starts to advance, starting at 00:00h on the 1st of August.

Once these steps are done, one can only stop and continue with the simulation and as I have explained in the “Airplane Information” section, it can be seen the information of a specific aircraft by clicking on it.

CHAPTER 7. Pretest

This chapter explains how the final version of the simulator has been reached, for which the simulator has gone through 5 different phases to test the proper functioning of the different functions of the code. These phases are explained below, where you can find the main difficulties of each one and what changes were made in each of the phases.

7.1 Phase 1: Verification of configuration of taxiway and aircraft movements

This first phase consists of verifying that the taxiway and pushback movement functions are correctly coded, carrying out the planned movements and testing whether the taxiways are correctly configured. To check this, two simulations were made of the movement of an aircraft from a stand to each of the runway thresholds, 25R and 25L, and a third simulation from a rapid exit of runway 25R to the stand. This would check the correct functioning of the aircraft movement and the parking and deplaning procedure. Apart from verifying if the code performed well the calculations of the new coordinates in each time differential, in this first phase it was also verified if the programme when creating the aircraft appeared in the assigned place on the screen, with the appropriate image and its position was moving according to the new calculated coordinates.

The first step was to create the taxiways necessary to perform these routes and the stand where the aircraft would start or end its route. Fig. 7.1 shows the declared taxiways and the stand (in a circle). At this stage the route was not created with Dijkstra's algorithm, but was entered manually in the code. This phase was very important to correct errors of the two movement functions and to ensure their full effectiveness.

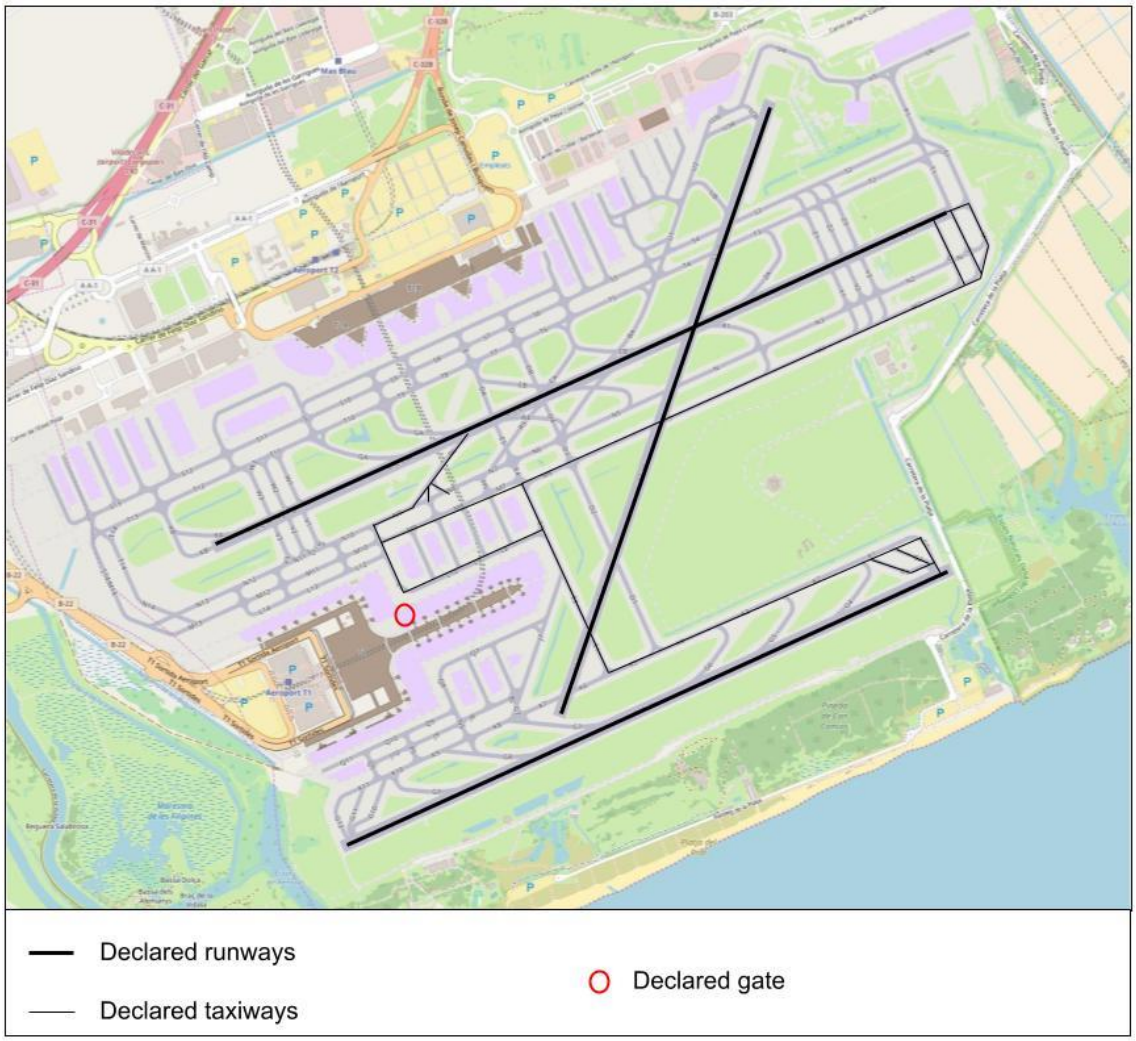


Fig 7.1 Declared Taxiways, Runways and gates

7.2 Phase 2: Apply Dijkstra's algorithm

Once the correct functioning of the movement and pushback functions has been seen, it is time to apply Dijkstra's algorithm to check how well it works. For this phase, no new routes or new taxiways were declared, because there was already a possible way to reach the destination and with the minimum taxiways it would be easier to check its correct functioning.

For this phase, new variables were added to the taxiways, tracks and stands so that they would know where each one of them was in the Dijkstra matrix. This matrix is obtained automatically from a function, which will later be used to find the optimal route.

When all the changes in the code had been applied, the functionality of the code was checked to see if the chosen route made sense and if it was the same as in the case that I had introduced it manually. The result of one of the routes, the one that goes from the stand to the threshold of the runway 25L, surprised

me because in the final part it chose to go through the taxiway G3 instead of going through the G1 which is the normal procedure, but this route certified me that the code and therefore the Dijkstra algorithm was correctly coded because to get to the threshold of the runway it is faster to go through G3 because it is the most direct route. In Fig. 2.13 where the different declared taxiways are shown, you can see the taxiways G1 and G3.

7.3 Phase 3: Implementing collision detection and prevention

In this phase new code functions were created such as collision detection and avoidance, the choice of which rapid exit to use, which taxiway to enter the runway for, take-off on runway 25R (delaying a take-off from runway 25L) and the speed function that depends on the current traffic. These functions were made at this time due to the environment with few variables (limited taxiways and few aeroplanes), this way it is easier to check its correct functioning and then implement it in the real case with the whole airport. For the realisation of some functions new variables and even classes had to be added. In the case of the function that determines which quick exit to take, we created the class model of the aircraft that contains information of each model such as the maximum take-off weight, landing weight, take-off distance and landing distance among others.

7.4 Phase 4: Declare all taxiways

This is the most important phase and the one in which most changes have been made, because we went from using a few taxiways to using the whole airport, to load all the aircraft information from Excel instead of entering it manually in the code and we also added direction to some taxiways so that they were not bidirectional.

To carry out all these modifications, we first had to create all the taxiways of the airport with their corresponding names and start and end coordinates. In addition to the taxiways, we also had to create the 3 runways with their respective addresses and each of the possible stands where a plane could park. For each of the stands the pushback trajectory had to be described together with its name and location. In total 6 runways, 259 taxiways and 231 stands have been created, which means that the Dijkstra matrix that is created each time to determine the optimal path has a size of 496x496.

As explained in the Section 5.2 Resources used, an external code obtained from the Coderwall webpage (see in [35]) has been used to read Excel files. This code has been used to read and save in variables of the simulator the Excels of the taxiways, stands and the one that contains all the information of the planes that operated at the airport on 1 August 2018.

For a simulation that is more in line with the reality of Barcelona-El Prat airport, directionality was added to some taxiways. This directionality has been obtained by creating a second matrix of the same size as the one used by Dijkstra's algorithm. This matrix is initially filled with zeros, and is replaced by a 1 at the position where you want to have this directionality. An example would be if you can go from taxiway 1 to taxiway 3 but not the other way around, so you have only one direction, in row 1 column 3 the value of 0 will be changed to a 1 (a1,3=1), and in row 3 column 1 the value will be kept at 0 (a3,1=0). In this way, when the original matrix is multiplied point by point with this one, a new matrix will be obtained that will describe the possible movements and thus the directionality of some of the taxiways.

Some of the code also had to be remodelled to accommodate the new changes, such as the functions that create the taxiways, stands and planes.

Once these changes were made, the simulator now had all the inputs that were loaded by pressing the "Load All" button. It has to be said that the implementation of these changes was not so easy because errors were found in parts of the code already made and even errors in the coordinates of some taxiways that made the matrix of Dijkstra's algorithm not represent the truth.

7.5 Phase 5: 24h simulation

This last phase consists of the simulation of the programme itself. Before starting to simulate the 24 hours, we wanted to see how the programme works in the airport's off-peak hours, such as 5 in the morning, and once this period was over, put it in the maximum stress situation, i.e. at the airport's peak time, which is approximately around 17h. These are the hours chosen because, as can be seen in Fig. 2.6, in the period around 5 there is a very low number of departures and arrivals, while at 17 the number of departures and arrivals is the highest of the day. Also, if there is any error, the simulation is stopped, the error is detected and then corrected. Once corrected, the simulation is restarted a few moments before the error is found, to see if it has been corrected properly. The results of the pre-simulation tests have been satisfactory. In the off-peak hours, the simulator performed adequately. In this test it was not possible to check the code relating to collision detection and avoidance as well as the time spacing between aircraft departures and arrivals, but all the other functions such as routing, the directionality of some taxiways and the different aircraft movements, among others, have been checked and verified. The next scenario is at peak hours. The result obtained from simulating this scenario is very remarkable because there have been few conflicts between aircraft, and all of them have been solved satisfactorily. One of the reasons for the low number of conflicts is due to the correct functioning of the spacing functions for both arrivals and departures. As a result, there are fewer aircraft on the taxiways and consequently fewer conflicts. In addition, the spacing works quite well as we can observe that the queue generated in the taxiways prior to the take-off runway are very short or almost null and therefore the time that an aircraft is in a holding position is minimal.

Once these tests prior to the full-day simulation have been completed, we can say that the code is correct, and the entire programme can be simulated. When the 24 hours of the simulator have elapsed without any outstanding problem, the programming part is concluded and all that remains is to analyse the results that have been saved in an Excel file.

CHAPTER 8. Analysis of the results

As explained above, one of the main objectives of this Thesis is to simulate and model the functions of the ground controller to see if it can be replaced by a software program. The program simulates all the ground movements of a whole day, in this particular case the 1st of August 2018 over the airport of Barcelona - El Prat. In this chapter several aspects of the simulator will be analysed, such as the number of movements with respect to the average speed of the aircraft, the time that an aircraft is moving and not moving and its reasons, the delay it has caused and its robustness in the face of possible conflicts.

8.1 Average speed

Once the programme was working perfectly, a number of variables were added to analyse its behaviour. A number of variables are dedicated to counting both arrivals and departures, the amount of time an aircraft is in motion and stopped. If an aircraft is stationary, a distinction is made between two possible cases: if it is stationary at an intersection because an aircraft is already there or is coming from a taxiway with higher priority, or if it is stationary because it has an aircraft in front of it and would violate the safety distance between aircraft.

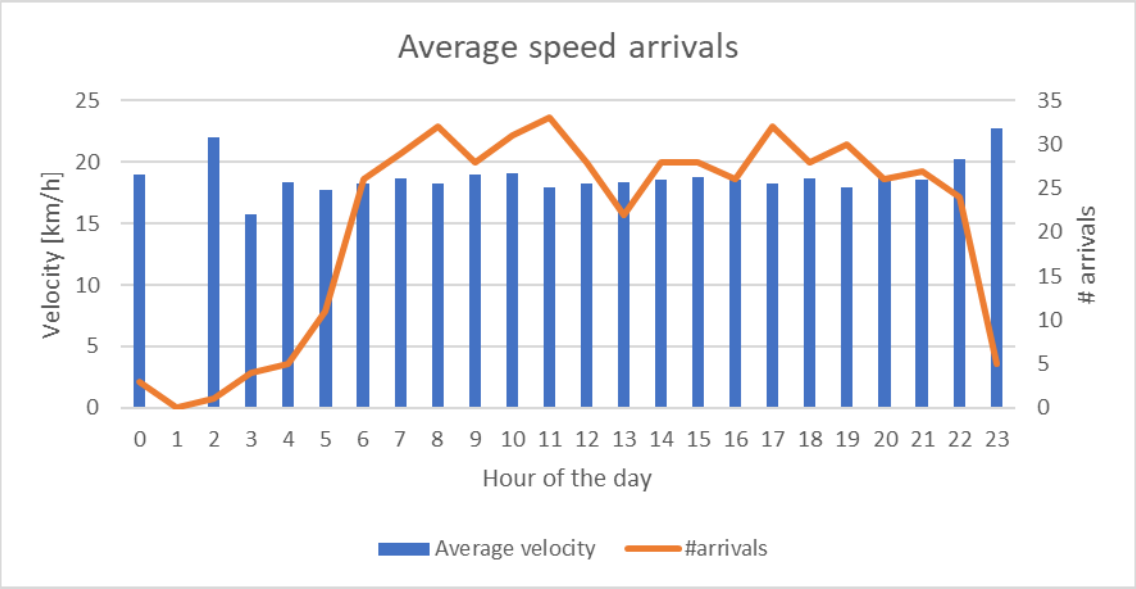


Fig. 8.1 Average speed arrivals

In the above graph, one can observe the average speed in kilometres per hour of the aircraft in the different time slots of the day. This information is represented by the blue vertical bars, and there is also an orange line that gives information on the number of departures in that time slot.

From the graph obtained we can analyse various aspects of the programme together with the operation of the airport we have used as a scenario (Barcelona-El Prat airport).

To begin with, you can see what has been said in section 2.1.2 "scenario" where the airport suffers a large drop in demand during the night shift (23:00h-06:00h), and during the rest of the day there is an average of approximately 30 arrivals per hour.

On the other hand, we can see that during the night shift, in other words, when there are fewer aircraft movements, the average speed of the arriving aircraft is higher than during the rest of the day. These two aspects are closely related because if there are more aircraft moving on the apron, it is easier to come across one and have to stop, and consequently the average speed of the aircraft will be affected by decreasing this value.

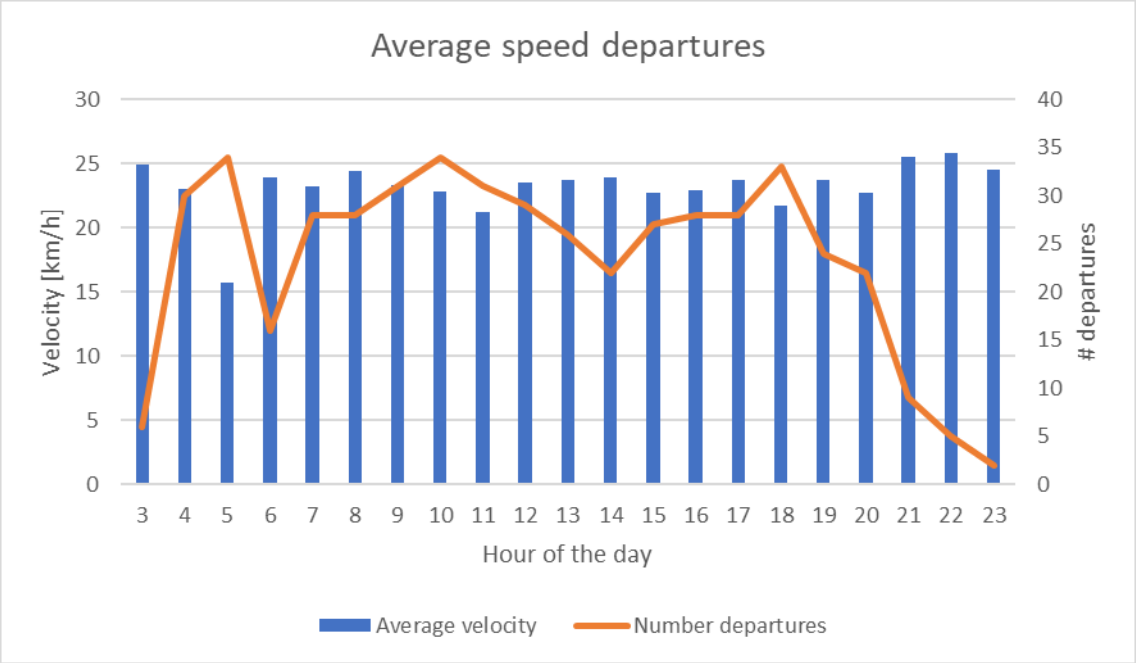


Fig. 8.2 Average speed departures

Fig. 8.2 shows the same graph as Fig. 8.1 but in this case it refers only to the departures.

In this case one can observe that as with arrivals, the demand in the night shift is much lower than the rest of the day. However, unlike arrivals, it can be observed that this night shift is shifted a couple of hours earlier (21:00h - 04:00h). During this period, the average speed is slightly higher than the rest of the day, and it should also be noted that there are no departures between 0:00h and 2:00h.

It should also be noted that at 5:00h the average speed is very low due to the fact that most of the departures take place in a very short period of time and there is a queue to take off, having to wait for a while in the taxiways.

In addition, one of the biggest problems that many of the world's major airports have, and Barcelona airport also suffers from it, is that all the airlines want to depart at the best times of the day, creating peak hours such as 10am and 6pm. Demand falls between these two hours, with a minimum of 22 departures at 14:00 noon.

The difference in speed between arrivals and departures is that in arrivals, the pushback movement is longer than in departures for safety reasons, which is why the average speed is affected by the longer time spent at reduced speed.

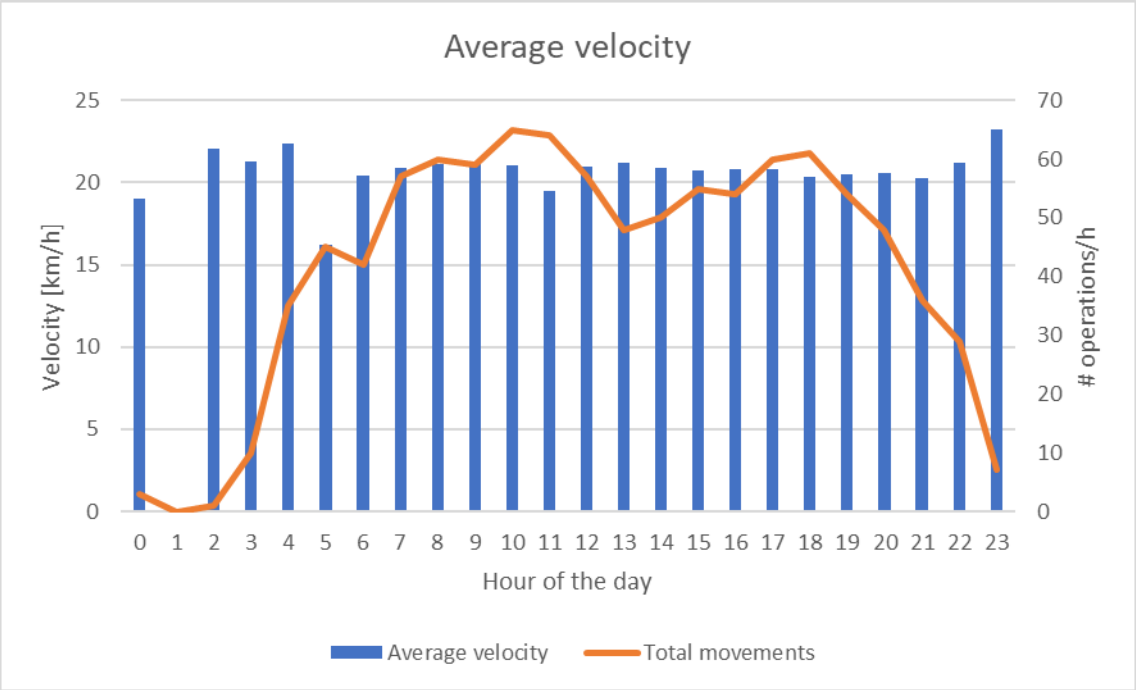


Fig. 8.3 Total average velocity

Fig. 8.3 represents the fusion of the two previous figures. This figure shows more clearly the different aspects mentioned above, such as the difference in demand between the night shift and the rest of the day, that the airport suffers peak hours between 9-10h in the morning and 17-18h in the afternoon, and that the average speed is affected. Moreover, in the night period where there are fewer movements, the average speed is slightly improved.

If one takes into account that the airport capacity is 40 departures and 38 arrivals per hour with the west configuration parallel runways according to Eurocontrol (see in [38]), it can be seen in the above graphs that the simulator is performing below these limits. This is due to the fact that the spacing has been set just right to meet the airport capacity, in other words, in the best-case

scenario, one aircraft will take off every 90 seconds, meaning that there will be 40 departures per hour. The maximum number of departures is at 10h with 34 take-offs, but the 40 operations are not reached due to the fact that in certain time slots there are not so many departures scheduled at that time. In the case of arrivals, the same thing happens, the runway capacity limit of 38 landings per hour is never reached.

8.2 Time in motion vs stationary

Another aspect that is interesting to analyse is to take into account the average time that an aircraft has been stopped and its reasons in comparison with the time it has been moving towards its destination. In this particular case, we wanted to differentiate the time that the aircraft is stopped according to its cause. The possible reasons for stopping a plane are the following: there is a plane with higher priority entering an intersection or perhaps there is already a plane at that intersection, and the second reason is because there is a plane in front of it and it is necessary to stop in order not to violate the safety distance. These two cases have been coded in order to also analyse whether the programme is well optimised.

In Fig. 8.4, the average amount of time an aircraft is in motion and when it is stationary is shown in a doughnut diagram. It should be noted that for arrivals, almost all of the time the aircraft is moving towards its destination without encountering any obstacles or conflicts. Because the percentage is so small, it is not clear that 0.22% of the time an aircraft is stopped because it has an aircraft in front of it, but 0.07% of the time it is stopped because another aircraft has precedence at an intersection.

If we put this graph together with the previous one we can see that the average speed is very stable because most of the planes have been going all the time in the taxiways at 15 knots without having to stop due to a conflict.

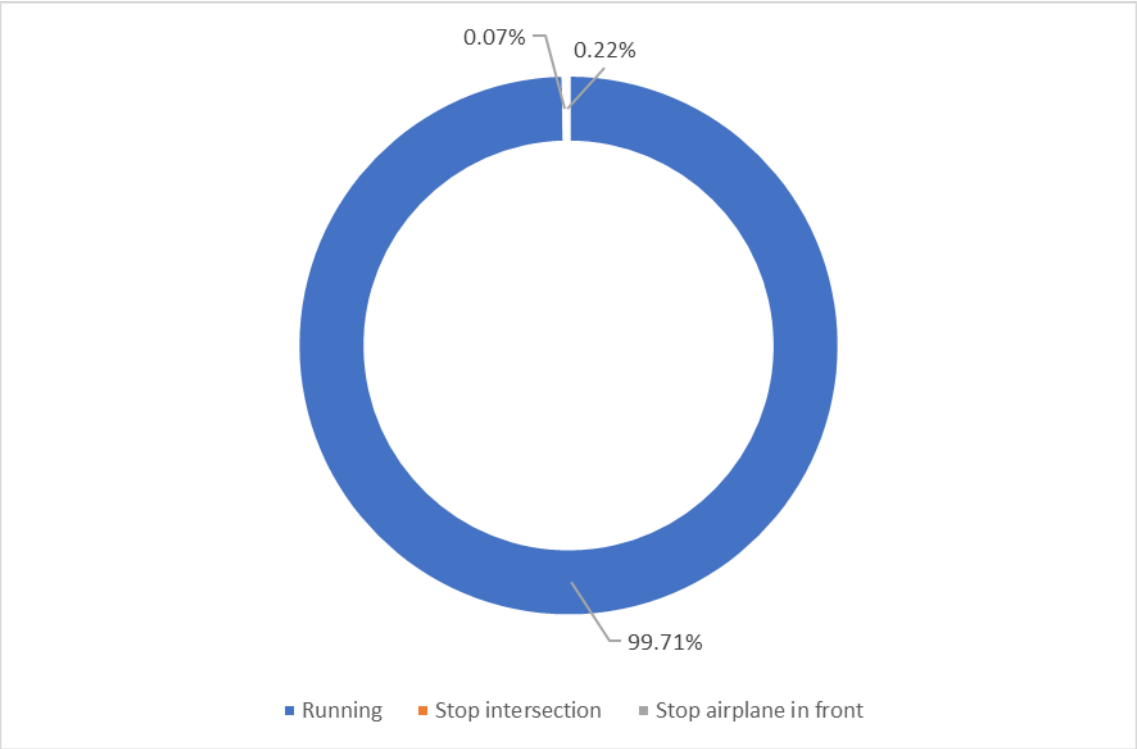


Fig. 8.4 Percentage of time in motion vs stopped on arrivals

In Fig. 8.5, one can see the same graph as the one above, but in this case instead of the information referring to arrivals, it is about departures. It can be seen that for departures it is also very predominant that the aircraft is moving rather than stopped for any reason, hence the 97.36% versus the 0.2% and 2.44% of the time that an aircraft is stopped at an intersection and because it has an aircraft in front of it, respectively.

The main reason why the percentage of time an aircraft is stationary on departures is higher than on arrivals is because, depending on the time zone, a small queue may form in the vicinity of the runway threshold. It is for this reason, that only the time a plane is standing still increases due to the fact that it has a plane in front of it.

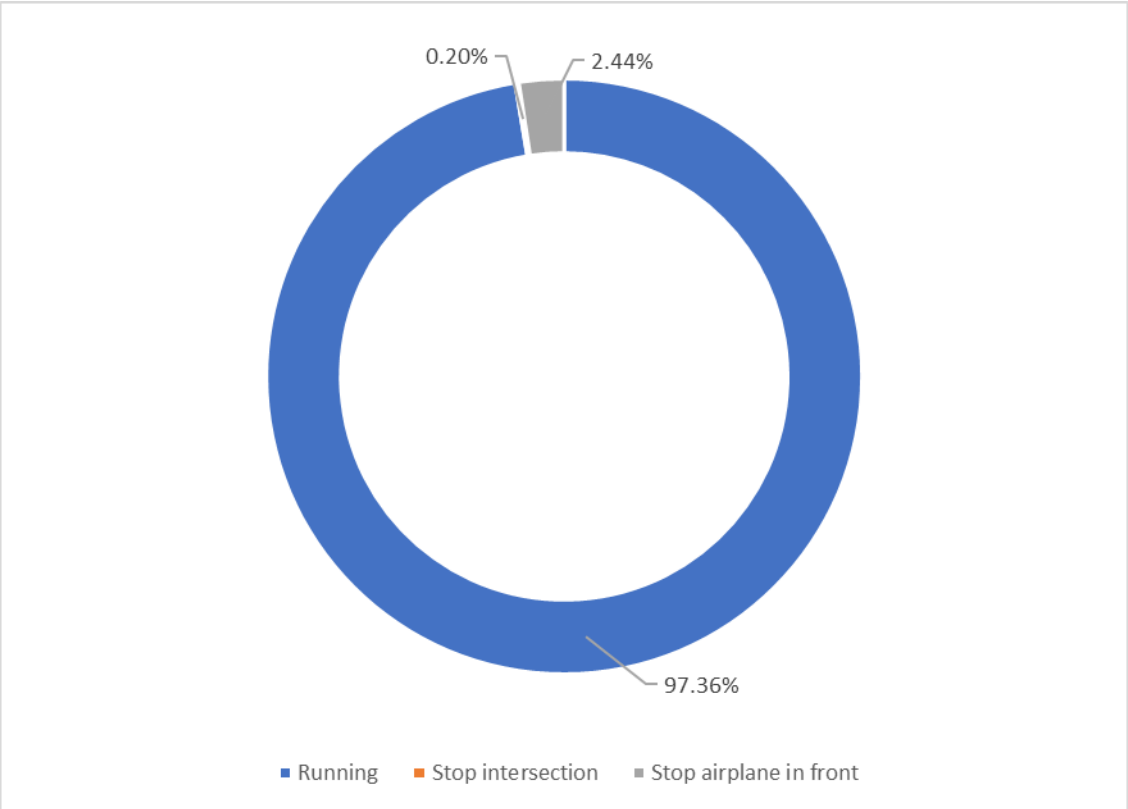


Fig. 8.5 Percentage of time in motion vs stopped on departures

In addition, in order to justify the fact that waiting at the runway head is the main reason why the idle time is greater for departures than for arrivals, another graph has been made like the two previous ones, but only flights taking off in the period between 5:00h and 5:59h will be taken into account. This is the period chosen because, as can be seen in Fig. 8.2, the average speed was lower than the rest because, as mentioned earlier, in this time slot there were many aircraft departing in a very short period of time and queues formed in the taxiways before the runway.

As can be seen in Fig. 8.6, it can be seen that there is a correlation between the pre-runway queues with the increase in the average time an aircraft is stationary due to aircraft in front of it waiting for their slot to take off. As a result, the time on the move has decreased significantly to approximately 76%. As previously mentioned, from 5 to 6 o'clock there are many departures in a short period of time, which is why there are more aircraft moving on the apron and consequently apart from increasing the time on standby due to aircraft in front, it is also easier to find an aircraft at an intersection, which is why this percentage is also a little higher than the average number of departures.

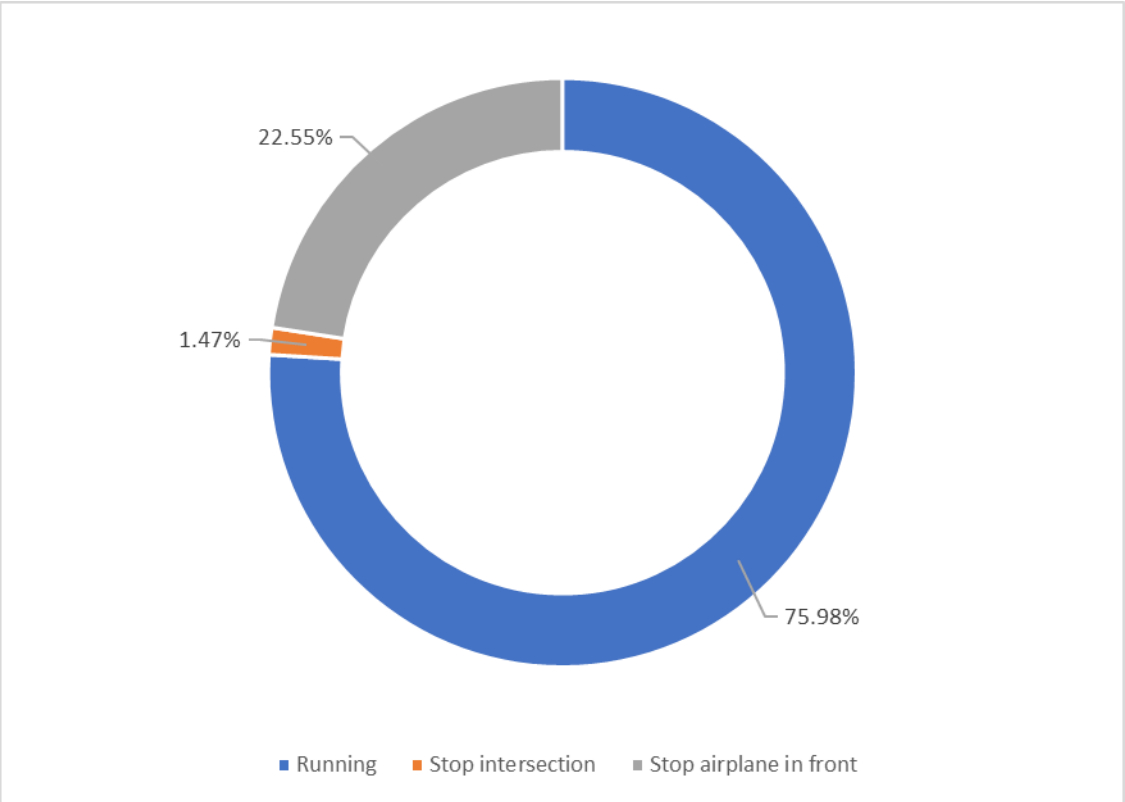


Fig. 8.6 Percentage of time in motion vs. stopped at departures at 5 a.m.

8.3 Delay

Another variable that we wanted to analyse to check the correct functioning of the simulator is the delay it generates in departures. Only departures have been taken into account because the programme simulates the functions of the ground controller, therefore, aircraft arriving at the airport have been assumed to arrive at the scheduled time.

Table 8.1 shows the number of departures scheduled on 1 August 2018, the number of these aircraft delayed with respect to the ETD and the percentage with respect to the total. In addition, information is given on the average delay time, the maximum and minimum values, the number of times a delay occurs, the median and the mode.

Total Airplanes	Number of delays	Percentage
494	75	15.18%

Delay time [minutes]	Number of times
1	22
2	8
3	11
4	10
5	5

6	7
7	3
8	4
9	3
10	1
11	0
12	1

Average delay	Median	Mode
3 minutes	3 minutes	1 minute

Table 8.1 Departure Delay

What can be deduced from the table above is that the percentage of aircraft that suffer a delay is very small, only 15.18% of them depart at least 1 minute late. But the most outstanding fact is that the average delay value is only 3 minutes, this value is very small and therefore very acceptable considering that the scenario used is the second airport in Spain and has been used as a reference day, August 1st, which is typically one of the days of the year with more aircraft movements. Another important aspect is the value that we have obtained for the mode, this value is 1 minute, therefore it means that the most repeated delay value is only 1 minute over the ETD. This means that aircraft punctuality is 100% due to the fact that all aircraft depart with a delay of less than 15 minutes.

Furthermore, it has to be taken into account that a large part of this delay is due to overbooking at 5 am, if this period is excluded from the statistics, even better values can be seen in Table 8.2. The percentage of delayed aircraft has decreased by approximately 5%, the maximum delay value has also decreased to 9 minutes. The average and median have both decreased to 2 minutes, and the mode remains at 1 minute. If we compare both tables, we can see that the difference between the number of times a delay has been repeated is mainly that the largest values, such as 12 and 10 minutes, have disappeared, that the median values have decreased the most, for example, 6 minutes has gone from being repeated 7 times to 2, and on the other hand the smallest values, such as 1 or 2 minutes, have seen the least change in their value.

This means that if we take into account the rest of the day, the values obtained are very positive, meaning that the programme can easily handle the busiest time of the day at the airport, which are the 34 departures that occur at 10 o'clock, together with the 31 arrivals that occur in the same time interval, without drastically modifying the average speed, i.e. the planes do not encounter traffic in front of them, making the airport greener and the creation of gases such as CO, CO2 and NOX less than currently.

Total Airplanes	Number of delays	Percentage
460	49	10.65%

Delay time [minutes]	Number of times

1	20
2	7
3	8
4	6
5	3
6	2
7	1
8	1
9	1
10	0
11	0
12	0

Average delay	Median	Mode
2 minutes	2 minutes	1 minute

Table 8.2 Departure Delay excluding 5 a.m

Referring to the delay, one thing to note is that the programme does not create delay. In other words, in the airport environment, the delay can be caused by several sources, by the airline itself as the plane arrives later and consequently will leave later, passengers themselves can create delays, by the ground handling service, which takes longer than expected to do the process of unloading bags and loading new bags for the next flight, other services such as refuelling or the emptying of waste water are delayed, and even one of the main reasons for delays at an airport is the weather, due to low visibility or wind on the runway.

All the factors that induce a delay at any airport that have been mentioned in the previous section have not been taken into account in the simulator designed, therefore, it is normal that the average delay that the programme has generated is so close to 0. The totality of the delay created is due to the wait that the aircraft have had to enter the runway and receive the clearance for take-off.

8.4 Robustness

A very important feature that this simulator must have is robustness. In the case of our simulator, robustness refers to several aspects, the first of which is that the programme does not hang or suddenly stop working. This aspect is very important because of the consequences that can occur at an airport if this system is implemented and suddenly stops working. Furthermore, considering that this system is designed to replace the ground controllers, if this system goes down, there would be no controllers to resume activity without consequences. Therefore, this part of the robustness of the program is very important, and the result with respect to this part is very satisfactory since in the

various tests carried out including the simulation of a full day (24 hours) at no time has had any of the above mentioned problems.

Another aspect that defines the robustness of this simulator is the ability of the code to adapt to any situation and solve any conflict. Currently the simulator is programmed to use a single runway configuration, the west configuration parallel runways. The code created recreates exactly and without any problem the directionality of the different taxis and all aircraft obtain the optimal path without any problem. Furthermore, as can be seen in Fig. 2.8, aircraft can also take off on the longer runway, and this fact has also been configured in the simulator, delaying the take-off from the sea runway and the following landing. Apart from the modelling of the scenario, this section also considers the ability of the code to adapt to any situation, especially conflict resolution.

Fig. 8.7 Potential conflict

Conflict resolution is one of the most difficult functions to program, because as we simulated different scenarios or more hours of simulation, a new conflict appeared that it did not know how to resolve and it had to partially change the code. This means that we only know how it behaves with the conflicts that the program has experienced, which are many, but not all of them, which is why in this section it is not possible to ensure 100% robustness. Moreover, as explained earlier, when a plane is arriving at the end of a taxiway, what it does is to see if there is anyone on the next taxiway and if so, in which direction it is going. But, here a problem arises, and it is the following, as can be seen in Fig. 8.7, two aircraft are arriving at the end of their taxiway, and the next taxiway (S7) that they have to take, according to their optimal route, is the same. The problem is that they both look to see if there is anyone on the next taxiway and see that there is no one, so they proceed to access this taxiway. The problem is that both aircraft are now on the same taxiway but in opposite directions. This situation can occur, although it is very unlikely, due to the knowledge that we had of this type of conflict, the easiest and best solution has been to define a directionality to most of the taxiways, therefore two planes cannot be in the same taxiway and have opposite directions. However, it is also true that due to the infrastructure of Barcelona-El Prat airport, some taxiways are not possible to define a directionality because the planes would not be able to access or leave their relevant gate or reach the desired destination. An example of a taxiway that has had to be left bidirectional are the ones highlighted in red that can be seen in Fig. 8.8.

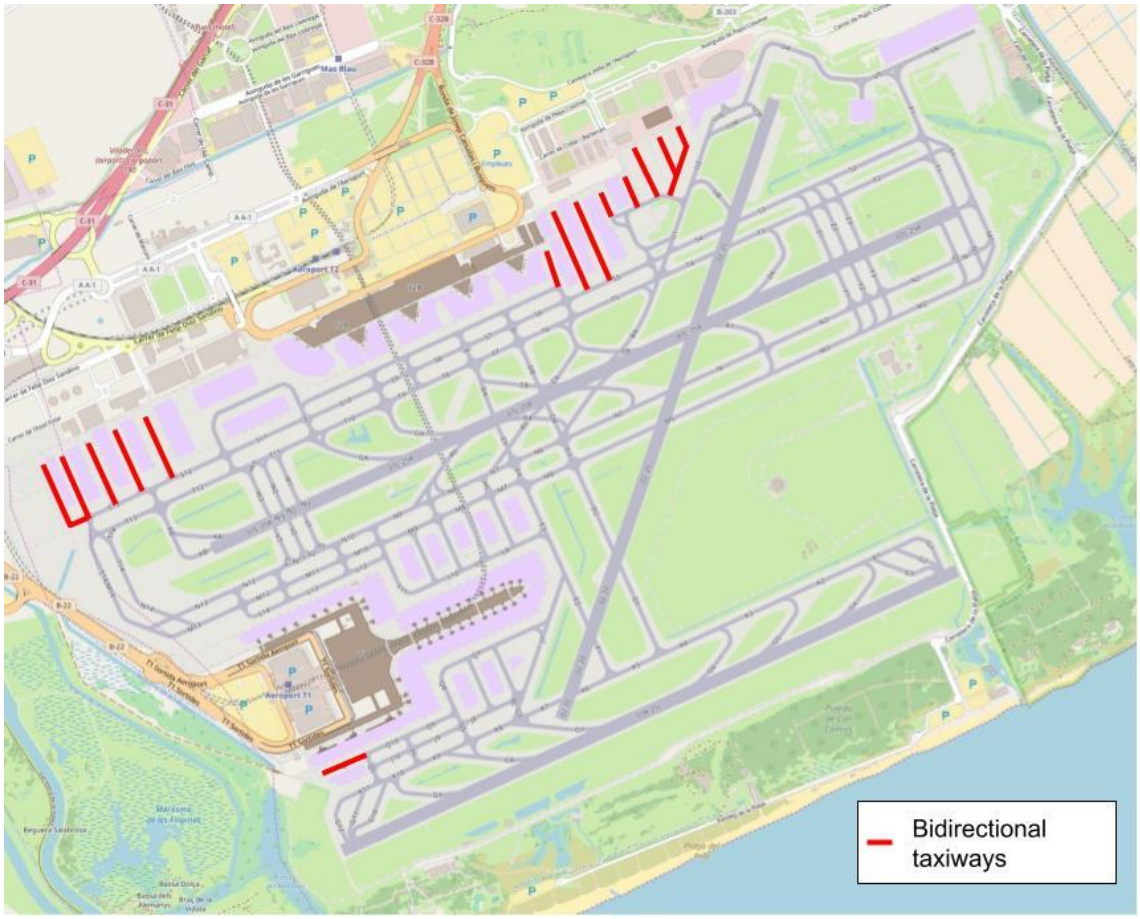


Fig. 8.8 Bidirectional taxiways

8.5 Comparison metric

Apart from the aspects discussed in the previous sections, the aim was to give a value for the improvement that this programme would bring to the scenario tested, that is, to Barcelona-El Prat airport. In order to obtain a value, a comparison metric had to be created, the best way to do this metric is by means of a trajectory analysis. The problem we have encountered is that it has not been possible to obtain sufficient data to make an acceptable comparison metric, an example of a lack of information would be the workload of controllers at Barcelona-El Prat airport or an erroneous decision by ATC. For this reason, it has not been possible to make a good comparison metric to be able to check the improvement of the tool created in this work with respect to the current system at the airport. The information that we have been able to obtain thanks to FlightRadar (see in [39]) is the trajectory of about 30 aircraft, where we obtain the coordinates of their route and at what time they are at those coordinates. With this information we have been able to find out approximately the average taxi time at the airport.

In the case of the exits, the starting coordinates have been entered in Google Earth to observe the exit stands. The result can be checked in Fig. 8.9. As can be seen, the stands of the different aircraft are located throughout the airport, both the T2 and T1 terminals. It is for this reason that the results obtained can be considered as the average time to go from the stand to the threshold of runway 25L will take into account several departure locations. The average taxiing time of the 17 departing aircraft is 12 minutes and 20 seconds, the maximum time is almost 18 minutes for aircraft U26026 and the shortest time is just over 8 minutes for flight VY2472. The main difference is the distance to be travelled.

Fig 8.9 Departure trajectories

In the case of arrivals, the times are much shorter, with an average taxiway time of 3 minutes and 8 seconds. As can be seen in Fig. 8.10, the stands are closer to runway 25R. The maximum taxi time obtained is almost 10 minutes for flight EY49, due to its location in the south dock of terminal T1.

Fig. 8.10 Arrival trajectories

If we compare these values with those obtained by the simulator, note that the taxiway time on arrivals is practically a couple of minutes higher than that analysed previously. The main reason for this difference is the speed in the fast departures, in the simulator it goes at a lower speed, but it is mainly because the few trajectories that have been analysed are located very close to the runway (upper part of T1 and central part of T2) and there have not been many aircraft with a long route.

In contrast to the previous case, in the departures where the trajectories analysed are more heterogeneous, the simulator has an average taxiways time of just over 11 minutes as opposed to over 12 minutes in the trajectories analysed. In this case, this improvement is mainly due to the optimisation of the simulator's allocation of departure slots, and consequently the reduction in the number of aircraft simultaneously in circulation and the queues produced.

CHAPTER 9. Implementation in a real environment

This chapter explains how this programme can be implemented in real life. One of the objectives of this work is to improve airport operations without having to spend any money on infrastructure, so if this programme cannot be implemented in a real environment, it makes no sense. For this purpose, there are several scenarios of how this technology can be implemented. In the following, each of these cases will be explained in detail.

In all of the following options the controller is in control and can change the trajectories of the aircraft and tell it to continue or stop or look for another route. For each of the following options the program would have to be minimally modified to send the information in a format that the receiving device or machine is able to understand the different instructions it is being told.

9.1 Local environment

An implementation is considered to be a local environment when this type of technology is applied to an airport. For this purpose, there are currently two possibilities being developed that can be combined with the programme developed in this work.

9.1.1 Option 1

For this option, the idea was taken from the Follow The Greens programme (see in [40]). This system uses the illumination of the centre lines of taxiways, using the colours green and red, to continue their flight indicating the way at any intersection or if they have to stop, respectively. This solution can be coupled with the designed programme, it can be of great help to the pilots and especially to the controllers. In the case of leaving a stand to go to a runway, the ground controllers would only have to tell the pilot "Taxi to holding point runway 25L/R, Follow the Greens", from then on the lights will guide the aircraft to the corresponding runway in the most optimal way and avoid any collision. One problem may be that this system is not as effective in daylight due to the difficulty of visualisation of the lights at certain times, on the other hand, at night there would be no problem, also in situations of low visibility or low light is especially effective improving safety and reducing the mental workload of the controllers. This solution could be a breakthrough for any airport.

The system would consist of the surface movement radar, the software created and the illumination of the taxiways. The first will give the position of each of the aircraft to the programme, which will consist of a server room and a central computer that will carry out the various calculations. In the control tower, several screens will be placed for the different ground controllers' stations where the screen of the programme designed in this work will be shown.

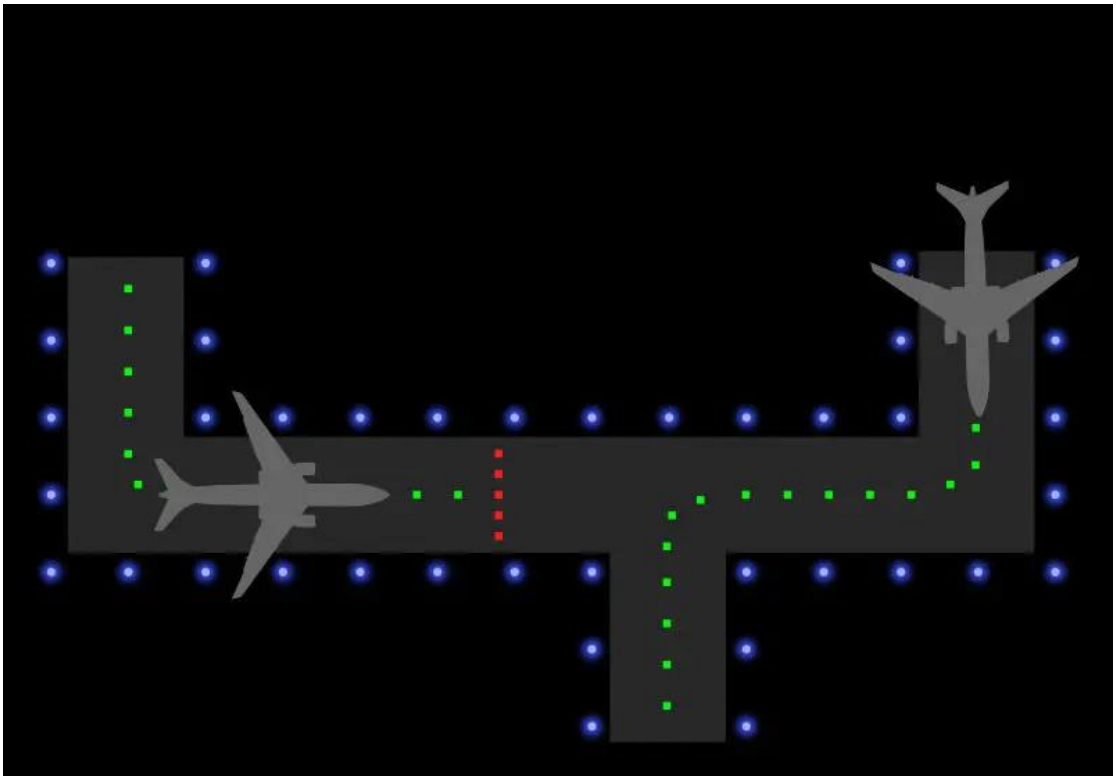


Fig. 9.1 Follow the Greens scheme

9.1.2 Option 2

This other option is the most viable in the short term, and consists of using pushback tractors to do all the taxiing on the taxis up to the threshold of the runway. This is the most viable option because by doing the whole taxiways movement with the tractor you can go with the engines off and only turn them on shortly before take-off with respect to the required warm-up time. This idea is being developed by the company Taxibot and they currently have vehicles of theirs distributed in airports such as Amsterdam Schiphol. It also has some very interesting features, as explained on the Taxibot website (see in [41]):

- No modification in airplane system
- No need for APU replacement
- No added weight to the airplane
- No impact on nose landing gear fatigue life
- Pilots fully in control during taxiing
- Improved traction on wet or icy surfaces

This technology can be very interesting due to the environmental situation we are living nowadays, the aviation sector is one of the most criticised due to pollution and many airports want to reduce both environmental pollution and noise. The benefits for airlines are mainly the reduction of fuel costs and the

Foreign Object Damage can be reduced by up to 50% due to the late start of engines. At airport level, a significant reduction of greenhouse gases and noise levels whilst it increases operational efficiency and throughput at airport gates can be observed. Pilots can be benefited by combining this idea with the programme developed in this thesis because instead of guiding aircraft, pushback tractors will be guided, turning these vehicles into autonomous and remotely guided vehicles, yet pilots will be in control at all times and will be able to use airplane tiller and brakes once the pushback procedure is completed.

Fig. 9.2 Taxibot pushback tractor

9.2 Global environment

An implementation is considered global when it is not intended for use at single airports, but for use at multiple airports. This does not mean that all airports have to upgrade and use this technology, this tool is an additional help for the controller that will improve the airport's capacity. For this purpose, there are currently two possibilities being developed that can be combined with the programme developed in this work.

9.2.1 Option 1

This first option consists of using a technology being developed by Honeywell and SAFRAN, which in 2013 Airbus agreed to develop and evaluate on the A320 family. This technology involves using electric motors on each of the main landing gears driving inboard wheels and powered by the Auxiliary Power Unit

(APU) generator. This system is called the Electric Green Taxiing System (EGTS) and is an electric system that allows the taxi and pushback procedure to be performed without the need for main engines or a tug tractor. This system is designed primarily for single-aisle aircraft because the electric motors weigh around 400kg and on a long-haul flight the taxi time is too long to justify the burden of this weight. The company claims that the system can save up to 4% of fuel per flight cycle, reduce FOD by shutting down the main engines, reduce carbon and NOx emissions, and less noise during taxiing.

If this technology is applied to more aircraft models such as the Boeing 737, it could be combined with the designed ground controller automation programme where the programme sends commands to the aircraft through a certain frequency and the aircraft receives and processes them in a completely autonomous manner, without the need for any action from the pilot, it would make the entire route as the programme specifies.

This implementation can be of great help to both the controller and the pilot who does not have to make any decisions or avoid collisions, because the program will already solve all conflicts in an optimal way. Apart from the cost of installing the software in the airport environment, aircraft manufacturers would have to invest in the development and implementation of an EGTS for the different aircraft models. This is why such a real-life implementation would be more costly and its application would be medium to long term.

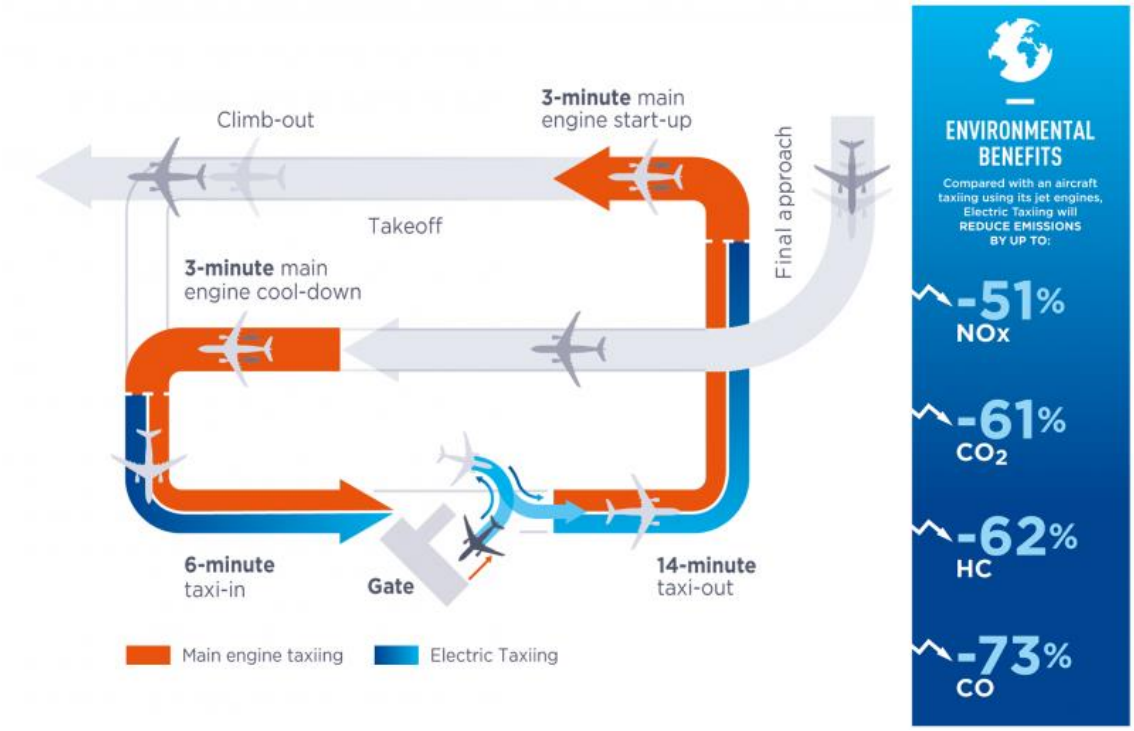


Fig. 9.3 EGTS scheme

9.2.2 Option 2

A second option would be to design a device that would be placed in the pilot's cockpit where once the pilot arrives at the airport, the tower controller would assign a different frequency to each of the aircraft so that the programme would send each of the commands to each aircraft. This system would not have to be coupled with any other movement system, which means that the pilot would have to do the entire taxi and pushback procedure. However, the only difference is that instead of receiving instructions from the ground controller, he would receive instructions from a device located in his cockpit.

This system is not the most sustainable of all the solutions because the engines are still running, but the use of the programme minimises taxi time and the delay that can occur, resulting in lower fuel consumption. It should also be noted that this system would be one of the most economical and quickest to implement as we are not dependent on third party technology. On the other hand, this option can be used as a short term solution while other more ambitious and environmentally friendly implementations are being developed.

Fig. 9.4 Airplane's cockpit

9.3 Model and simulate changes in the infrastructure

Another implementation that the programme can have is that it gives you the opportunity to change aspects of the airport such as taxiway direction, create a

new taxiway, change some stands or even try to change the runway capacity and see how these changes would affect the airport and what consequences they would have in reality without putting any aircraft at risk. Currently, the expansion of some of the most important European airports, such as Barcelona-El Prat airport, is being discussed. For this particular case, the simulator could be used to model and simulate the extension of the runway sea (25L-07R) and the construction of the satellite terminal, to see how these changes would affect the airport and to be able to demonstrate whether they are necessary or not.

This real-life implementation is one of the most likely because it does not depend on another company's system, there is no need for any kind of installation and the modification of the programme would be minimal. In addition, the consequences of acquiring this product can be a great improvement in airport operations without the need to invest in new infrastructure due to the simulation of different scenarios.

CHAPTER 10. Conclusions and Future Work

Next, we will explain the main conclusions obtained from the project and the possible future lines of work, where we will explain different aspects in which to improve the simulator in order to obtain a better performance and be more robust.

10.1 Conclusions

Once all the work has been carried out and the results obtained, it can be said that the programme produces a significant improvement in the airport's operations. By not modifying any infrastructure, the capacity of the runway, taxiways and stands is the same, but on the other hand, the movement of aircraft is more regulated, optimising taxiing time and avoiding queues. This is achieved because each aircraft leaves its stand at the exact time to arrive at its assigned departure slot.

One aspect that must be stressed is that by placing so many restrictions on the taxiways, to define directionality and avoid collisions, the optimal route found by the simulator using Dijkstra's algorithm is the same path as the one used in real life. If a more optimal route is to be found, the number of taxiways defined with directionality should be reduced, but at the same time that these conditions are removed, the conflict detection should be improved, especially in the long-range, because this type of conflict will be more concurrent and will have to avoid that two aircraft meet head-on on the same taxiway, as shown in Fig. 8.7.

It should also be noted that the programme has fixed all potential conflicts that occurred on 1 August 2018. But because the code was only tested in a 24-hour period and faced a limited number of situations, it can be concluded that it is not robust enough to completely replace the ground controllers. Besides, if one adds that one of the weaknesses of the programme is the detection of long-term conflicts, it calls into question the lack of robustness. Even so, as currently coded, there have been no collisions or problems with platform movement. Therefore, although it is not possible to replace ground controllers in their entirety, this tool can be of great help to these controllers to reduce their workload or even, due to the correct functioning of the programme, it can reduce the number of controllers needed for the same airport.

One virtue of this tool is that it has the ability for the controller to stop and resume a particular aircraft and also to modify the current route, causing it to pass through a particular taxiway. In this way, the controller has power over the programme to make decisions.

Although it has not been possible to obtain the desired comparison metric, it can be deduced that the programme does not generate much improvement in landings, but a substantial improvement can be observed in departures due to

the fact that each aircraft departs at the right time so that the queue on the taxiways prior to the runway threshold is minimal. Therefore, the additional taxi time is drastically reduced although the pre-departure delay can be increased by making the aircraft wait at their stand.

Considering the different implementations that can be carried out with this programme, for each of them the programme would have to be slightly modified to adapt it to the data input and output of each solution. Although the most realistic implementation is the simulation of the airport environment to determine the traffic at a certain time and also how the change of the infrastructure would affect the airport operations (changing taxiway routing, adding a new taxiway, runway or terminal or even extending an existing runway).

Finally, I would like to end this section with some personal conclusions. Mainly, this job has provided me with a lot of self-learning, both from looking for information about the airport in question and from searching on the internet how to program different aspects, such as how to open, read and write Excel documents or how to use the time functions that are already in the programme. During this year I have learnt that one virtue I have is problem solving, due to the large number of obstacles I have had to overcome. Another virtue but at the same time a defect is perfectionism, during the development of the simulator, at all times I wanted to go a little further, making it more realistic, giving more features, improving its performance, ... consequently it has made the final result of the work more than satisfactory, but at the same time it has lasted longer than expected and I still find many aspects that I think should be improved. One of the aspects that I would improve would be the initial approach, because I started programming step by step without being very clear about the final ideas and in the end the code is not the best or the clearest of all. Ideally, if I started again I would have to have clearer all the variables and the final objective to make a simpler code.

10.2 Future work

The continuation of this project would start by giving more realism to the simulator, that is to say, adding all the available runway configurations and with each of the configurations change the direction of the different taxiways. The program could be configured to automatically change from the west configuration parallel runways to the night configuration from 23h and then change again at 7 am. It could also be possible to set the current wind conditions (direction and speed), if there is rain (defining its intensity) and the visibility at the airport, so that the program can decide which is the best configuration for the current weather conditions. Another possible example of how to improve the realism of the program would be the following, as we have seen in the results of the work, the delay caused by the program is very small because in an airport, the delay is caused mainly by external effects to the taxiing between the stand and the runway. For this reason, each aircraft could be randomly made to be ready for pushback within ±10 minutes of the estimated departure time from the stand, making the minutes closer to 0 more likely to occur and, as is normal, positive values more common than negative

values. Once an aircraft is ready for pushback, the program should look at the next available slot and taking into account the time of the slot and the time it takes to reach the runway will give the exact time when the aircraft can start to pushback. In this way the program will be more realistic and will improve its robustness over externally caused delays.

Also, a future line of work is the improvement of conflict detection and resolution to improve the robustness of the programme, and to achieve the real goal of this work, which is to be able to replace the ground controllers.

It has to be taken into account that depending on the implementation of this program, the next improvements will be different. For example, if you want to use it in an airport, where it receives from the radar the current positions of the planes and transmits to the planes the different decisions (turn by a certain taxiway, hold on position, cleared for takeoff, ...) you will have to modify the processing of the information, modify how to read the input values and how to send the decisions to the planes. But on the other hand, if you want to use it as a simulator to see how it would affect some modifications in the infrastructure, it would be more appropriate to add on-screen buttons to add new taxiways, runways or stands or even to modify some of the properties.

A feature of the work is that it has been possible to verify the simulation model by means of the conceptual model. In other words, the program was able to make the appropriate decisions, and when it did not do so, the problem was found and the code was modified so that it would behave as desired. On the other hand, the project could not be validated because it could not be carried out in a real environment.

Finally, another future line of work, which would be complementary to the work done, would be to exploit the available trajectory resources. In this way, if this is put together with the developed thesis, a good comparison could be made by simulating different scenarios, which could conclude more accurately the real implementation that this tool can have in aviation.

To sum up, as in other aspects of life, it is very difficult for a machine to replace a human being, for the simple fact that until its safety and proper functioning is known, it will not be able to replace a human being. In our case the same thing happens, the main conclusion of the work is that the simulator makes good decisions and even improves very important aspects in an airport such as delays. But the lack of robustness and the fact that the code was tested in a limited number of scenarios (it was only tested in a period of 24 hours) is not enough to prove that this solution can be a replacement for ground controllers, although it can be a first step. However, it can actually be a helpful tool that can reduce the workload of the controllers and thus reduce the number of controllers needed.

Bibliography

[1] Air traffic control [Internet]. Wikipedia. Available from: https://en.wikipedia.org/wiki/Air_traffic_control

[2] History of Air Traffic Control [Internet]. Unión Sindical de Controladores Aéreos. Available from: https://www.usca.es/en/profession/history-of-air-traffic-control/

[3] ICAO Annex 11 - Air Traffic Services. ICAO. Available from: https://skyrise.aero/wp-content/uploads/2017/03/ICAO-Annex-11-Air-traffic-services.pdf

[4] Transfer of Radar Identification [Internet]. FAA. Available from: https://www.faa.gov/air_traffic/publications/atpubs/atc_html/chap5_section_4.html

[5] Flight Progress Strips [Internet]. Skybrary. Available from: https://skybrary.aero/articles/flight-progress-strips.

[6] Características de los distintos tipos de radares [Internet]. Gaceta Aeronautica. Available from: https://www.gacetaeronautica.com/gaceta/wp-101/?p=22789.

[7] Classification of Radar systems [Internet]. Radar tutorial. Available from: https://www.radartutorial.eu/02.basics/rp04.es.html.

[8] Surface Movement Radar [Internet]. Skybrary. Available from: https://skybrary.aero/articles/surface-movement-radar.

[9] Remote Tower Service [Internet]. Skybrary. Available from: https://skybrary.aero/articles/remote-tower-service.

[10] Airport collaborative decision-making [Internet]. Eurocontrol. Available from: https://www.eurocontrol.int/concept/airport-collaborative-decision-making

[11] Surface CDM Implementation [Internet]. ICAO. Available from: https://www.icao.int/NACC/Documents/Meetings/2019/ACDM/ACDM-P8.pdf

[12] Airspace Technology Demonstration 2 [Internet]. NASA. Available from: https://aviationsystems.arc.nasa.gov/research/atd2/index.shtml

[13] Aeropuerto Josep Tarradellas Barcelona-El Prat [Internet]. Wikipedia. Available from: https://es.wikipedia.org/wiki/Aeropuerto_Josep_Tarradellas_Barcelona-El_Prat.

[14] Historia Aeropuerto Josep Tarradellas Barcelona - El Prat [Internet]. Aena. Available from: https://www.aena.es/es/josep-tarradellas-barcelona-el-prat/conocenos/historia.html.

[15] Espais d'interès natural del delta del Llobregat [Internet]. Territori. Available from: http://territori.scot.cat/cat/notices/espais_d_interEs_natural_del_delta_del_llobregat_2005_768.php

[16] Plano de Estacionamiento y Atraque de Aeronaves – OACI – Josep Tarradellas Barcelona – El Prat. AIP LEBL. Available from: https://aip.enaire.es/AIP/contenido_AIP/AD/AD2/LEBL/LE_AD_2_LEBL_PDC_1_en.pdf

[17] Trafico en los aeropuerto españoles 2019 [Internet]. Ministerio de Transportes, Movilidad y Agenda Urbana. Available from: https://www.mitma.gob.es/recursos_mfom/listado/recursos/trafico_en_los_aeropuertos_espanoles_-_2019.pdf.

[18] Statement of Work of Work Package 3 ITA Course (2015).

[19] AIP LEBL [Internet]. Enaire. Available from: https://aip.enaire.es/AIP/#LEBL.

[20] Anexo VIII AIP LEBL [Internet]. Ministerio de Transportes, Movilidad y Agenda Urbana. Available from: https://www.mitma.gob.es/recursos_mfom/anexoviiiaip.pdf

[21] Nuevas configuraciones diurnas del aeropuerto del Prat [Internet]. Associació de veïns de Gavamar. Available from: http://www.gavamar.com/ES/index1.php?ruta=http://www.gavamar.com/ES/aeroport.htm

[22] Google Maps [Internet]. Google. Available from: https://www.google.es/maps

[23] Plano de Aeródromo para Movimientos en Tierra – OACI – Josep Tarradellas Barcelona – El Prat. AIP LEBL. Available from: https://aip.enaire.es/AIP/contenido_AIP/AD/AD2/LEBL/LE_AD_2_LEBL_GMC_1_1_en.pdf

[24] Almost half of flights in Barcelona airport delayed this summer [Internet]. CatalanNews. Available from: https://www.catalannews.com/business/item/almost-half-of-flights-in-barcelona-airport-delayed-this-summer (2018)

[25] LEBL Performance [Internet]. Ansperformance. Available from: https://ansperformance.eu/dashboard/stakeholder/airport/db/lebl#overview

[26] Airport ATFM Delay [Internet]. Eurocontrol. Available from: https://www.eurocontrol.int/prudata/dashboard/metadata/airport-atfm-delay/

[27] Additional Taxi-Out Time [Internet]. Eurocontrol. Available from: https://www.eurocontrol.int/prudata/dashboard/metadata/additional-taxi-out-time/

[28] ATC Pre-departure Delay [Internet]. Eurocontrol. Available from: https://www.eurocontrol.int/prudata/dashboard/metadata/atc-pre-departure-delay/

[29] Punctuality [Internet]. Ansperformance. Available from: https://ansperformance.eu/definition/punctuality/

[30] Piera M.A., Guasch T., Casanovas J., Figueras J*., Cómo mejorar la logística de su empresa mediante la simulación*, Ediciones Díaz de Santos, Madrid, 2006

[31] Passenger Flow Simulation at Frankfurt Airport [Internet]. Anylogic. Available from: https://www.anylogic.com/resources/case-studies/simulation-of-the-frankfurt-airport/

[32] CAST Simulation Software [Internet]. Airport Research Center. Available from: https://arc.de/cast-simulation-software/

[33] Doc 8643 – Aircraft Type Designators. ICAO. Available from: https://www.icao.int/publications/DOC8643/Pages/default.aspx.

[34] Printing Paths in Dijkstra's Shortest Path Algorithm [Internet]. Geeks for Geeks. Available from: https://www.geeksforgeeks.org/printing-paths-dijkstras-shortest-path-algorithm/.

[35] Read Excel File in C# [Internet]. Coderwall. Available from: https://coderwall.com/p/app3ya/read-excel-file-in-c.

[36] Write to Excel in C# [Internet]. DelftStack. Available from: https://www.delftstack.com/es/howto/csharp/write-to-excel-in-csharp/.

[37] Airplanes specifications [Internet]. Aerocorner. Available from: https://aerocorner.com/aircraft-categories/airplanes.

[38] Barcelona BCN / LEBL airport information [Internet]. Eurocontrol. Available from: https://ext.eurocontrol.int/airport_corner_public/LEBL.

[39] FlightRadar24 [Internet]. Available from: https://www.flightradar24.com/

[40] Follow the Greens [Internet]. Available from: http://followthegreens.com/.

[41] TaxiBot [Internet]. Available from: https://www.taxibot-international.com/.

[42] Airbus and Safran to market an electric taxiing system for A320 Family [Internet]. Safran. Available from: https://www.safran-group.com/news/airbus-and-safran-market-electric-taxiing-system-a320-family-2017-06-21 (2017).

[43] Safran suspends electric jet taxiing project after Airbus ends talks [Internet]. Reuters. Available from: https://www.reuters.com/article/us-safran-airbus-taxi-idUSKBN1Y72MN (2019).

ANNEX A.

A.1 Dijkstra class

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace TFG_versio1
{
    public class Dijkstra
    {
        public double[,] graph1; //matrix with all the costs
        public double[,] GraphMultiplier; //matrix that define the
direcctionalities of the taxiways
        public int Dijkstra1;//number of elements in the matrix
        double Slope;//line equation
        double YIntersect;//line equation
        taxiways Taxiway = new taxiways();
        Runway Runway = new Runway();
        Gate Gate = new Gate();
        int[][] path1;
        int[] path2;
        int n;
        int m;
        double[] shortestDistances;


        public double[] GetShortestDistance() //obtain the total cost
of the route
        {
            return (this.shortestDistances);
        }

        public int[][] GetPath() //
        {
            return (this.path1);
        }

        public void StartPath() //start the variables
        {
            this.path1 = new int[Dijkstra1][];
            this.path2 = new int[Dijkstra1];
        }

        public List<taxiways> CrearRuta(int [][] path, int finalpoint,
List<Gate>Gates, List<Runway>Runways, List<taxiways>Taxiways)
        {//We change the numbers of the vectors for taxiways. Each
number corresponds to a different taxiway
            List<taxiways> ruta=new List<taxiways>();
            for (int i = 0; i < Dijkstra1; i++)
            {
                if (i==finalpoint)
                {
                    for (int p = 0; p < m; p++)
                    {
```

```
                        if (path[i][p] == -1)
                        {
                            break;
                        }
                        if (Taxiways.Find(a =>
a.Dijkstra.Equals(path[i][p])) != null)
                        {
                            Taxiway = Taxiways.Find(a =>
a.Dijkstra.Equals(path[i][p]));
                            ruta.Add(Taxiway);
                        }
                    }
                }
            }
            ruta.Reverse();
            return (ruta);
        }

        public double[,] GetGraph() //obtain matrix with all costs
        {
            return (this.graph1);
        }


        //Code obtained from: https://www.geeksforgeeks.org/printing-
paths-dijkstras-shortest-path-algorithm/

        public readonly int NO_PARENT = -1;

        // Function that implements Dijkstra's
        // single source shortest path
        // algorithm for a graph represented
        // using adjacency matrix
        // representation
        public void dijkstra(double[,] adjacencyMatrix, int
startVertex)
        {
            int nVertices = adjacencyMatrix.GetLength(0);

            // shortestDistances[i] will hold the
            // shortest distance from src to i
            shortestDistances = new double[nVertices];

            // added[i] will true if vertex i is
            // included / in shortest path tree
            // or shortest distance from src to
            // i is finalized
            bool[] added = new bool[nVertices];

            // Initialize all distances as
            // INFINITE and added[] as false
            for (int vertexIndex = 0; vertexIndex < nVertices;
vertexIndex++)
            {
                shortestDistances[vertexIndex] = int.MaxValue;
                added[vertexIndex] = false;
            }

            // Distance of source vertex from
            // itself is always 0
            shortestDistances[startVertex] = 0;
```

```
            // Parent array to store shortest
            // path tree
            int[] parents = new int[nVertices];
            // The starting vertex does not
            // have a parent
            parents[startVertex] = NO_PARENT;


            // Find shortest path for all
            // vertices
            for (int i = 1; i < nVertices; i++)
            {

                // Pick the minimum distance vertex
                // from the set of vertices not yet
                // processed. nearestVertex is
                // always equal to startNode in
                // first iteration.
                int nearestVertex = -1;
                double shortestDistance = double.MaxValue;
                for (int vertexIndex = 0; vertexIndex < nVertices;
vertexIndex++)
                {
                    if (!added[vertexIndex] &&
shortestDistances[vertexIndex] < shortestDistance)
                    {
                        nearestVertex = vertexIndex;
                        shortestDistance =
shortestDistances[vertexIndex];
                    }
                }

                // Mark the picked vertex as
                // processed
                added[nearestVertex] = true;

                // Update dist value of the
                // adjacent vertices of the
                // picked vertex.
                for (int vertexIndex = 0; vertexIndex < nVertices;
vertexIndex++)
                {
                    double edgeDistance =
adjacencyMatrix[nearestVertex, vertexIndex];
                    if (edgeDistance > 0 && ((shortestDistance +
edgeDistance) < shortestDistances[vertexIndex]))
                    {
                        parents[vertexIndex] = nearestVertex;
                        shortestDistances[vertexIndex] =
shortestDistance + edgeDistance;
                    }
                }
            }

            printSolution(startVertex, shortestDistances, parents);
        }

        // A utility function to print
        // the constructed distances
        // array and shortest paths
```

```
        public void printSolution(int startVertex, double[] distances,
int[] parents)
        {
            StartPath();
            int nVertices = distances.Length;

            for (int vertexIndex = 0; vertexIndex < nVertices;
vertexIndex++)
            {
                if (vertexIndex != startVertex)
                {
                    printPath(vertexIndex, parents, startVertex);
                }
            }
        }

        // Function to print shortest path
        // from source to currentVertex
        // using parents array
        public void printPath(int currentVertex, int[] parents, int
startVertex)
        {//We save the optimal path in a array of arrays, the position
is the destiantion point and in that position there is the path to
reach its destination
            // Base case : Source node has
            // been processed
            if (currentVertex == NO_PARENT)//if the node has a value -
1
            {
                if (m == startVertex)//if the position [m] is the
start point
                {
                    int[] path3= new int [1] {-1};
                    path1[m]=path3;//we put the value -1 to this
position
                    m++;
                    path1[m] = path2;
                }
                else
                {
                    path1[m] = path2;//we put the path in its
corresponding position
                }
                n = 0;
                m++;
                path2 = new int[Dijkstra1];
                return;
            }
            path2[n] = parents[currentVertex]; //we add the next node
of the path to a vector
            n++;
            printPath(parents[currentVertex], parents, startVertex);
        }

        public void CreateMatrixToMultiply(List<Gate> Gates,
List<Runway> Runways, List<taxiways> Taxiways)
        { //creates a matrix that define the direcctionality of the
taxiways
            GraphMultiplier = new double[this.Dijkstra1,
this.Dijkstra1]; //Create the matrix
```

```
            for (int m = 0; m < this.Dijkstra1; m++) //Set all the
values at 1
            {
                for (int n = 0; n < this.Dijkstra1; n++)
                {
                    GraphMultiplier[m, n] = 1;
                }
            }
            //Create taxiways lists
            List<taxiways> Seleccionados=new List<taxiways>();
            List<taxiways> Seleccionados1 = new List<taxiways>();
            List<taxiways> Seleccionados2 = new List<taxiways>();
            Seleccionados = Taxiways.FindAll(a =>
a.Name.Contains("M"));//Add all the taxiways that its name contains a
M
            for (int p = 0; p < Seleccionados.Count(); p++)
            {//For each one of the taxiways in the list, we look for
the adjoining taxiway
                Taxiway = Seleccionados.Find(a =>
a.IX.Equals(Seleccionados[p].FX));
                if (Taxiway != null &&
Taxiway.Name.Contains("MS")==false)//if the taxiway is not named MS
                {
                    GraphMultiplier[Taxiway.Dijkstra,
Seleccionados[p].Dijkstra] = Double.MaxValue;//set a high value to go
to one taxiway to another (defining directionality)
                    Seleccionados1 = Taxiways.FindAll(a =>
a.FX.Equals(Taxiway.FX));
                    Seleccionados1 = Seleccionados1.FindAll(a =>
a.FY.Equals(Taxiway.FY));
                    Seleccionados2 = Taxiways.FindAll(a =>
a.IX.Equals(Taxiway.FX));
                    Seleccionados2 = Seleccionados2.FindAll(a =>
a.IY.Equals(Taxiway.FY));
                    Seleccionados1.AddRange(Seleccionados2);//We look
for all the perpendicular taxiway that match at the end of a desired
taxiway
                    for(int t=0;t<Seleccionados1.Count; t++)
                    {
                        if
(Seleccionados1[t].Name.Contains("M")==false &&
Taxiway.Name.Contains("M1") == false)//We make sure the taxiway is
perpendicular
                        {

GraphMultiplier[Seleccionados1[t].Dijkstra, Taxiway.Dijkstra] =
Double.MaxValue; //Set a high value to go from those taxiways to the
desired one
                        }
                    }
                }
            }
            Seleccionados = Taxiways.FindAll(a =>
a.Name.Contains("K"));//Add all the taxiways that its name contains a
K
            for (int p = 0; p < Seleccionados.Count(); p++)
            {//For each one of the taxiways in the list, we look for
the adjoining taxiway
                Taxiway = Seleccionados.Find(a =>
a.IX.Equals(Seleccionados[p].FX));
```

```
                if (Taxiway != null && Taxiway.Name.Contains("KA") ==
false && Taxiway.Name.Contains("KB") == false &&
Taxiway.Name.Contains("KW") == false && Taxiway.Name.Contains("KE") ==
false)//if the taxiway is not named one of the previously mentioned
                {
                    GraphMultiplier[Taxiway.Dijkstra,
Seleccionados[p].Dijkstra] = Double.MaxValue;//set a high value to go
to one taxiway to another (defining directionality)
                    Seleccionados1 = Taxiways.FindAll(a =>
a.FX.Equals(Taxiway.FX));
                    Seleccionados1 = Seleccionados1.FindAll(a =>
a.FY.Equals(Taxiway.FY));
                    Seleccionados2 = Taxiways.FindAll(a =>
a.IX.Equals(Taxiway.FX));
                    Seleccionados2 = Seleccionados2.FindAll(a =>
a.IY.Equals(Taxiway.FY));
                    Seleccionados1.AddRange(Seleccionados2);//We look
for all the perpendicular taxiway that match at the end of a desired
taxiway
                    for (int t = 0; t < Seleccionados1.Count; t++)
                    {
                        if (Seleccionados1[t].Name.Contains("K") ==
false)//We make sure the taxiway is perpendicular
                        {

GraphMultiplier[Seleccionados1[t].Dijkstra, Taxiway.Dijkstra] =
Double.MaxValue; //Set a high value to go from those taxiways to the
desired one
                        }
                    }
                }
            }
            Seleccionados = Taxiways.FindAll(a =>
a.Name.Contains("E"));//Add all the taxiways that its name contains a
E
            for (int p = 0; p < Seleccionados.Count(); p++)
            {//For each one of the taxiways in the list, we look for
the adjoining taxiway
                Taxiway = Seleccionados.Find(a =>
a.IX.Equals(Seleccionados[p].FX));
                if (Taxiway != null && Taxiway.Name.Contains("ES") ==
false && Taxiway.Name.Contains("EB") == false &&
Taxiway.Name.Contains("LE") == false && Taxiway.Name.Contains("KE") ==
false && Taxiway.Name.Contains("BE") == false)//if the taxiway is not
named one of the previously mentioned
                {
                    GraphMultiplier[Taxiway.Dijkstra,
Seleccionados[p].Dijkstra] = Double.MaxValue; //Set a high value to go
from those taxiways to the desired one
                }
            }
            Seleccionados = Taxiways.FindAll(a =>
a.Name.Contains("D"));//Add all the taxiways that its name contains a
D
            for (int p = 0; p < Seleccionados.Count(); p++)
            {//For each one of the taxiways in the list, we look for
the adjoining taxiway
                Taxiway = Seleccionados.Find(a =>
a.FX.Equals(Seleccionados[p].IX));
```

```
                if (Taxiway != null && Taxiway.Name.Contains("D4") ==
false && Taxiway.Name.Contains("D3") == false)//if the taxiway is not
named one of the previously mentioned
                {
                    GraphMultiplier[Taxiway.Dijkstra,
Seleccionados[p].Dijkstra] = Double.MaxValue;//Set a high value to go
from those taxiways to the desired one
                }
            }
            Seleccionados = Taxiways.FindAll(a =>
a.Name.Contains("S"));//Add all the taxiways that its name contains a
S
            for (int p = 0; p < Seleccionados.Count(); p++)
            {//For each one of the taxiways in the list, we look for
the adjoining taxiway
                Taxiway = Seleccionados.Find(a =>
a.FX.Equals(Seleccionados[p].IX));
                if (Taxiway != null && Taxiway.Name.Contains("QS") ==
false && Taxiway.Name.Contains("PS") == false &&
Taxiway.Name.Contains("NS") == false && Taxiway.Name.Contains("BS") ==
false && Taxiway.Name.Contains("CS") == false &&
Taxiway.Name.Contains("ES") == false && Taxiway.Name.Contains("FS") ==
false && Taxiway.Name.Contains("GS") == false &&
Taxiway.Name.Contains("HS") == false && Taxiway.Name.Contains("MS") ==
false && Taxiway.Name.Contains("LS") == false)//if the taxiway is not
named one of the previously mentioned
                {
                    GraphMultiplier[Taxiway.Dijkstra,
Seleccionados[p].Dijkstra] = Double.MaxValue; //Set a high value to go
from those taxiways to the desired one
                    Seleccionados1 = Taxiways.FindAll(a =>
a.FX.Equals(Taxiway.IX));
                    Seleccionados1 = Seleccionados1.FindAll(a =>
a.FY.Equals(Taxiway.IY));
                    Seleccionados2 = Taxiways.FindAll(a =>
a.IX.Equals(Taxiway.IX));
                    Seleccionados2 = Seleccionados2.FindAll(a =>
a.IY.Equals(Taxiway.IY));
                    Seleccionados1.AddRange(Seleccionados2);//We look
for all the perpendicular taxiway that match at the beginning of a
desired taxiway
                    for (int t = 0; t < Seleccionados1.Count; t++)
                    {
                        if (Seleccionados1[t].Name.Contains("S") ==
false && Taxiway.Name.Contains("S14")==false)//We make sure the
taxiway is perpendicular
                        {

GraphMultiplier[Seleccionados1[t].Dijkstra, Taxiway.Dijkstra] =
Double.MaxValue;//Set a high value to go from those taxiways to the
desired one
                        }
                    }
                }
            }
            Seleccionados = Taxiways.FindAll(a =>
a.Name.Contains("T"));//Add all the taxiways that its name contains a
T
            for (int p = 0; p < Seleccionados.Count(); p++)
            {//For each one of the taxiways in the list, we look for
the adjoining taxiway
```

```
                Taxiway = Seleccionados.Find(a =>
a.IX.Equals(Seleccionados[p].FX));
                if (Taxiway != null)
                {
                    GraphMultiplier[Taxiway.Dijkstra,
Seleccionados[p].Dijkstra] = Double.MaxValue;//Set a high value to go
from those taxiways to the desired one
                    Seleccionados1 = Taxiways.FindAll(a =>
a.FX.Equals(Taxiway.FX));
                    Seleccionados1 = Seleccionados1.FindAll(a =>
a.FY.Equals(Taxiway.FY));
                    Seleccionados2 = Taxiways.FindAll(a =>
a.IX.Equals(Taxiway.FX));
                    Seleccionados2 = Seleccionados2.FindAll(a =>
a.IY.Equals(Taxiway.FY));
                    Seleccionados1.AddRange(Seleccionados2);//We look
for all the perpendicular taxiway that match at the end of a desired
taxiway
                    for (int t = 0; t < Seleccionados1.Count; t++)
                    {
                        if (Seleccionados1[t].Name.Contains("T") ==
false)//We make sure the taxiway is perpendicular
                        {

GraphMultiplier[Seleccionados1[t].Dijkstra, Taxiway.Dijkstra] =
Double.MaxValue;//Set a high value to go from those taxiways to the
desired one
                        }
                    }
                }
            }
            Seleccionados = Taxiways.FindAll(a =>
a.Name.Contains("J"));//Add all the taxiways that its name contains a
J
            for (int p = 0; p < Seleccionados.Count(); p++)
            {//For each one of the taxiways in the list, we look for
the adjoining taxiway
                Taxiway = Seleccionados.Find(a =>
a.FX.Equals(Seleccionados[p].IX));
                if (Taxiway != null && Taxiway.Name.Contains("J7_sup")
&& Taxiway.Name.Contains("J7_cent"))//if the taxiway is not named one
of the previously mentioned
                {
                    GraphMultiplier[Taxiway.Dijkstra,
Seleccionados[p].Dijkstra] = Double.MaxValue;//Set a high value to go
from those taxiways to the desired one
                    Seleccionados1 = Taxiways.FindAll(a =>
a.FX.Equals(Taxiway.IX));
                    Seleccionados1 = Seleccionados1.FindAll(a =>
a.FY.Equals(Taxiway.IY));
                    Seleccionados2 = Taxiways.FindAll(a =>
a.IX.Equals(Taxiway.IX));
                    Seleccionados2 = Seleccionados2.FindAll(a =>
a.IY.Equals(Taxiway.IY));
                    Seleccionados1.AddRange(Seleccionados2);//We look
for all the perpendicular taxiway that match at the beginning of a
desired taxiway
                    for (int t = 0; t < Seleccionados1.Count; t++)
                    {
                        if (Seleccionados1[t].Name.Contains("J") ==
false)//We make sure the taxiway is perpendicular
```

```
                        {

GraphMultiplier[Seleccionados1[t].Dijkstra, Taxiway.Dijkstra] =
Double.MaxValue;//Set a high value to go from those taxiways to the
desired one
                        }
                    }
                }
            }
            Seleccionados = Taxiways.FindAll(a =>
a.Name.Contains("L"));//Add all the taxiways that its name contains a
L
            for (int p = 0; p < Seleccionados.Count(); p++)
            {
                Taxiway = Seleccionados[p];
                if (Taxiway != null && Taxiway.Name.Contains("LS") ==
false && Taxiway.Name.Contains("LE") == false &&
Taxiway.Name.Contains("LW") == false && Taxiway.Name.Contains("to") ==
false)
                {//For each one of the taxiways in the list, we look
for the adjoining taxiway
                    Taxiway = Seleccionados.Find(a =>
a.IX.Equals(Seleccionados[p].FX));
                    if (Taxiway != null &&
Taxiway.Name.Contains("L11") == false)
                    {
                        GraphMultiplier[Taxiway.Dijkstra,
Seleccionados[p].Dijkstra] = Double.MaxValue;//Set a high value to go
from those taxiways to the desired one
                        Seleccionados1 = Taxiways.FindAll(a =>
a.FX.Equals(Taxiway.FX));
                        Seleccionados1 = Seleccionados1.FindAll(a =>
a.FY.Equals(Taxiway.FY));
                        Seleccionados2 = Taxiways.FindAll(a =>
a.IX.Equals(Taxiway.FX));
                        Seleccionados2 = Seleccionados2.FindAll(a =>
a.IY.Equals(Taxiway.FY));
                        Seleccionados1.AddRange(Seleccionados2);//We
look for all the perpendicular taxiway that match at the end of a
desired taxiway
                        for (int t = 0; t < Seleccionados1.Count; t++)
                        {
                            if (Seleccionados1[t].Name.Contains("L")
== false && Seleccionados1[t].Name.Contains("L8_dret") == false)//We
make sure the taxiway is perpendicular
                            {

GraphMultiplier[Seleccionados1[t].Dijkstra, Taxiway.Dijkstra] =
Double.MaxValue;//Set a high value to go from those taxiways to the
desired one
                            }
                        }
                    }
                }
            }
            Seleccionados = Taxiways.FindAll(a =>
a.Name.Contains("Q"));//Add all the taxiways that its name contains a
L
            for (int p = 0; p < Seleccionados.Count(); p++)
            {//For each one of the taxiways in the list, we look for
the adjoining taxiway
```

```
                Taxiway = Seleccionados[p];
                if (Taxiway != null && Taxiway.Name.Contains("QS") ==
false)
                {
                    Taxiway = Seleccionados.Find(a =>
a.FX.Equals(Seleccionados[p].IX));
                    if (Taxiway != null &&
Taxiway.Name.Contains("Q11") == false)
                    {
                        GraphMultiplier[Taxiway.Dijkstra,
Seleccionados[p].Dijkstra] = Double.MaxValue;//Set a high value to go
from those taxiways to the desired one
                        Seleccionados1 = Taxiways.FindAll(a =>
a.FX.Equals(Taxiway.IX));
                        Seleccionados1 = Seleccionados1.FindAll(a =>
a.FY.Equals(Taxiway.IY));
                        Seleccionados2 = Taxiways.FindAll(a =>
a.IX.Equals(Taxiway.IX));
                        Seleccionados2 = Seleccionados2.FindAll(a =>
a.IY.Equals(Taxiway.IY));
                        Seleccionados1.AddRange(Seleccionados2);//We
look for all the perpendicular taxiway that match at the beginning of
a desired taxiway
                        for (int t = 0; t < Seleccionados1.Count; t++)
                        {
                            if (Seleccionados1[t].Name.Contains("Q")
== false && Taxiway.Name.Contains("Q8") == false)//We make sure the
taxiway is perpendicular
                            {

GraphMultiplier[Seleccionados1[t].Dijkstra, Taxiway.Dijkstra] =
Double.MaxValue;//Set a high value to go from those taxiways to the
desired one
                            }
                        }
                    }
                }
            }
        }

        public void MultiplyMatrix(double[,] GraphMultiplier,
double[,] graph)
        {
            if (GraphMultiplier.Length == graph.Length)//We check that
the matrices are of the same size
            {
                for (int i = 0; i < GraphMultiplier.GetLength(0); i++)
                {
                    for (int j = 0; j < GraphMultiplier.GetLength(0);
j++)
                    {
                        graph[i,j]=GraphMultiplier[i,j]*graph[i,j];
//We multiply the original matrix point by point with the multiplicity
matrix in order to define the direction of the taxiways.
                    }
                }
            }
            this.graph1 = graph;//We store the resulting matrix in the
local variable graph1
        }
```

```
        public void CreateMatrix(List<Gate>Gates, List<Runway>Runways,
List<taxiways>Taxiways)
        {//All stands, taxiways and runways are entered as parameters.
            double[,] graph = new double[this.Dijkstra1,
this.Dijkstra1];//We create the dijkstra matrix
            for (int m = 0; m < this.Dijkstra1; m++)
            {
                Boolean find = false;
                if ((Gates.Find(a => a.Dijkstra.Equals(m))!=null) &&
find==false)//We configure the cost of getting from a stand to its
nearest taxiway and vice-versa
                {
                    graph[m, m] = 0;
                    Gate = Gates.Find(a => a.Dijkstra.Equals(m));
                    CalculateLineEquation(Gate.X, Gate.Y, Gate.FX,
Gate.FY);//The pushback line equation is calculated
                    double Slope1 = Slope;
                    double YIntersect1 = YIntersect;
                    double dist=double.MaxValue;
                    double TotalDistance1=0;
                    double TotalDistance2=0;
                    double TotalDistance3 = 0;
                    double TotalDistance4 = 0;//Variables to determine
which taxiway is closest
                    double[] position1=new double[2];//Vector to save
the x and y coordinates
                    int taxi=0;
                    double distanciaTaxi = double.MaxValue;
                    for (int p = 0; p < Taxiways.Count(); p++)//we go
through all the taxiways at the airport
                    {
                        CalculateLineEquation(Taxiways[p].IX,
Taxiways[p].IY, Taxiways[p].FX, Taxiways[p].FY); //Taxiway line
equation is calculated
                        double[] position = Punto_2rectas(Slope,
YIntersect, Slope1, YIntersect1);//Calculate the position where the
taxiway crosses and the pushback of the stand
                        double distancia = Math.Sqrt(((position[0] -
Gate.FX) * (position[0] - Gate.FX)) + ((position[1] - Gate.FY) *
(position[1] - Gate.FY))); //We calculate the distance between the end
of the pushback and the previously calculated intersection
                        TotalDistance1 = Math.Sqrt(((position[0] -
Taxiways[p].IX) * (position[0] - Taxiways[p].IX)) + ((position[1] -
Taxiways[p].IY) * (position[1] - Taxiways[p].IY)));//Calculate the
distance between the start of the taxiway and the intersection
                        TotalDistance2 = Math.Sqrt(((position[0] -
Taxiways[p].FX) * (position[0] - Taxiways[p].FX)) + ((position[1] -
Taxiways[p].FY) * (position[1] - Taxiways[p].FY)));//Calculate the
distance between the end of the taxiway and the intersection
                        TotalDistance3 = Math.Sqrt(((position[0] -
Gate.X) * (position[0] - Gate.X)) + ((position[1] - Gate.Y) *
(position[1] - Gate.Y)));//We calculate the distance between the stand
and the intersection
                        TotalDistance4 = Math.Sqrt(((position[0] -
Gate.FX) * (position[0] - Gate.FX)) + ((position[1] - Gate.FY) *
(position[1] - Gate.FY)));//We calculate the distance between the end
of the pushback and the intersection
                        if ((distancia <= dist) && ((TotalDistance1 <=
Taxiways[p].Hypotenuse) && (TotalDistance2 <=
Taxiways[p].Hypotenuse+0.01)) && TotalDistance4 < TotalDistance3)//We
look to see if the distance is less than the one we had saved
```

```
                        {
                            dist = distancia;
                            if (TotalDistance1 < TotalDistance2)
                            {
                                distanciaTaxi = TotalDistance1;
                            }
                            else
                            {
                                distanciaTaxi = TotalDistance2;
                            }
                            taxi = p;//We save the taxiway
                            position1 = position;
                        }
                    }
                    double TotalDistance = Math.Sqrt(((position1[0] -
Gate.X) * (position1[0] - Gate.X)) + ((position1[1] - Gate.Y) *
(position1[1] - Gate.Y)));//Distance the plane will have to travel
                    double time = TotalDistance / (3 * (0.514444) *
530.65 / 2690); //We calculate the time from the distance (in pixels)
and the speed(from knots to m / s to pixels / s)
                    graph[m, Taxiways[taxi].Dijkstra] = time;//We
assign the obtained value to the matrix in its corresponding position
                    time = TotalDistance1 / (Taxiways[taxi].velocity *
(0.514444) * 530.65 / 2690);
                    graph[Taxiways[taxi].Dijkstra, m] = time;
                    find = true;
                }
                else if ((Taxiways.Find(a => a.Dijkstra.Equals(m)) !=
null) && find == false)//We configure the cost of getting from a
taxiway to antoher taxiway
                {
                    graph[m, m] = 0;//We set the value of going from a
taxiway to itself to 0
                    Taxiway = Taxiways.Find(a =>
a.Dijkstra.Equals(m));
                    List<taxiways> Coinciden=new List<taxiways>();
                    taxiways Taxiway1=new taxiways();

                    if ((Taxiways.Find(a => a.IX.Equals(Taxiway.FX))
!= null) && (Taxiways.Find(a => a.IY.Equals(Taxiway.FY)) != null))
                    {
                        Coinciden = Taxiways.FindAll(a =>
a.IX.Equals(Taxiway.FX));
                        Coinciden = Coinciden.FindAll(a =>
a.IY.Equals(Taxiway.FY));//We look for all taxiways that start when a
specific taxiway ends.
                        if (Coinciden.Find(a =>
a.Dijkstra.Equals(Taxiway.Dijkstra)) != null)
                        {
                            Taxiway1=Coinciden.Find(a =>
a.Dijkstra.Equals(Taxiway.Dijkstra));
                            Coinciden.Remove(Taxiway1);//We remove the
specific taxiway from the list
                        }
                        for (int i=0; i<Coinciden.Count(); i++)//For
each of the taxiways on the list
                        {
                            if (Taxiway.time<0)
                            {
```

```
                                Taxiway.time= Taxiway.Hypotenuse /
((15 * 0.514444) * 530.65 / 2690); //15 is the velocity in knots,
other are conversion factors
                            }
                            graph[m, Coinciden[i].Dijkstra] =
Taxiway.time;//We add the cost (time in  seconds) of going from one
taxiway to its contiguous one in the matrix
                        }
                    }
                    if ((Taxiways.Find(a => a.FX.Equals(Taxiway.IX))
!= null) && (Taxiways.Find(a => a.FY.Equals(Taxiway.IY)) != null))
                    {
                        Coinciden = Taxiways.FindAll(a =>
a.FX.Equals(Taxiway.IX));
                        Coinciden = Coinciden.FindAll(a =>
a.FY.Equals(Taxiway.IY));//We look for all taxiways that ends when a
specific taxiway starts.
                        if (Coinciden.Find(a =>
a.Dijkstra.Equals(Taxiway.Dijkstra)) != null)
                        {
                            Taxiway1=Coinciden.Find(a =>
a.Dijkstra.Equals(Taxiway.Dijkstra));
                            Coinciden.Remove(Taxiway1);//We remove the
specific taxiway from the list
                        }
                        for (int i=0; i<Coinciden.Count(); i++)//For
each of the taxiways on the list
                        {
                            if (Taxiway.time < 0)
                            {
                                Taxiway.time = Taxiway.Hypotenuse /
((15 * 0.514444) * 530.65 / 2690); //15 is the velocity in knots,
other are conversion factors
                            }
                            graph[m, Coinciden[i].Dijkstra] =
Taxiway.time;//We add the cost (time in  seconds) of going from one
taxiway to its contiguous one in the matrix
                        }
                    }

                    if ((Taxiways.Find(a => a.IX.Equals(Taxiway.IX))
!= null) && (Taxiways.Find(a => a.IY.Equals(Taxiway.IY)) != null))
                    {
                        Coinciden = Taxiways.FindAll(a =>
a.IX.Equals(Taxiway.IX));
                        Coinciden = Coinciden.FindAll(a =>
a.IY.Equals(Taxiway.IY));//We look for all taxiways that starts when a
specific taxiway starts.
                        if (Coinciden.Find(a =>
a.Dijkstra.Equals(Taxiway.Dijkstra)) != null)
                        {
                            Taxiway1=Coinciden.Find(a =>
a.Dijkstra.Equals(Taxiway.Dijkstra));
                            Coinciden.Remove(Taxiway1);//We remove the
specific taxiway from the list
                        }
                        for (int i=0; i<Coinciden.Count(); i++)//For
each of the taxiways on the list
                        {
                            if (Taxiway.time < 0)
                            {
```

```
                                Taxiway.time = Taxiway.Hypotenuse /
((15 * 0.514444) * 530.65 / 2690); //15 is the velocity in knots,
other are conversion factors
                            }
                            graph[m, Coinciden[i].Dijkstra] =
Taxiway.time;//We add the cost (time in  seconds) of going from one
taxiway to its contiguous one in the matrix
                        }

                    }
                    if ((Taxiways.Find(a => a.FX.Equals(Taxiway.FX))
!= null) && (Taxiways.Find(a => a.FY.Equals(Taxiway.FY)) != null))
                    {
                        Coinciden = Taxiways.FindAll(a =>
a.FX.Equals(Taxiway.FX));
                        Coinciden = Coinciden.FindAll(a =>
a.FY.Equals(Taxiway.FY));//We look for all taxiways that ends when a
specific taxiway ends.
                        if (Coinciden.Find(a =>
a.Dijkstra.Equals(Taxiway.Dijkstra)) != null)
                        {
                            Taxiway1=Coinciden.Find(a =>
a.Dijkstra.Equals(Taxiway.Dijkstra));
                            Coinciden.Remove(Taxiway1);//We remove the
specific taxiway from the list
                        }
                        for (int i=0; i<Coinciden.Count(); i++)//For
each of the taxiways on the list
                        {
                            if (Taxiway.time < 0)
                            {
                                Taxiway.time = Taxiway.Hypotenuse /
((15 * 0.514444) * 530.65 / 2690); //15 is the velocity in knots,
other are conversion factors
                            }
                            graph[m, Coinciden[i].Dijkstra] =
Taxiway.time;//We add the cost (time in  seconds) of going from one
taxiway to its contiguous one in the matrix
                        }
                    }


                    find = true;
                }
                else if ((Runways.Find(a => a.Dijkstra.Equals(m)) !=
null) && find == false)//We configure the cost of getting from a
taxiway to a runway
                {
                    graph[m, m] = 0;//We set the value of going from a
runway to itself to 0
                    Runway = Runways.Find(a => a.Dijkstra.Equals(m));
                    for (int p = 0; p < Taxiways.Count(); p++)//For
each one of the taxiways
                    {
                        if (Taxiways[p].Name.Contains("to") &&
Taxiways[p].FX==Runway.IX && Taxiways[p].FY==Runway.IY)//If the
taxiway contains "to" and ends at the runway threshold
                        {
                                        double TotalDistance1 =
Math.Sqrt(((Runway.IX - Taxiways[p].IX) * (Runway.IX -
```

```
Taxiways[p].IX)) + ((Runway.IY - Taxiways[p].IY) * (Runway.IY -
Taxiways[p].IY)));
                                        double TotalDistance2 =
Math.Sqrt(((Runway.IX - Taxiways[p].FX) * (Runway.IX -
Taxiways[p].FX)) + ((Runway.IY - Taxiways[p].FY) * (Runway.IY -
Taxiways[p].FY)));
                                        if (TotalDistance1 <
TotalDistance2)
                                        {
                                            double time =
TotalDistance2 / (Taxiways[p].velocity * (0.514444) * 530.65 / 2690);
//From the distances calculated above, calculate the time it will take
to travel it.

      graph[Taxiways[p].Dijkstra, m] = time;
                                        }
                                        else
                                        {
                                            double time =
TotalDistance1 / (15 * (0.514444) * 530.65 / 2690);//From the
distances calculated above, calculate the time it will take to travel
it.

      graph[Taxiways[p].Dijkstra, m] = time;
                                        }

                    }
                }
                find = true;
            }

        }
        CreateMatrixToMultiply(Gates, Runways, Taxiways);
        MultiplyMatrix(GraphMultiplier, graph);
    }

    public void SetDijkstra(int Dijkstra)
    {
        this.Dijkstra1 = Dijkstra;
    }

    public int GetDijkstra()
    {
        return (this.Dijkstra1);
    }

    public void CalculateLineEquation(double IX, double IY, double
FX, double FY)//We calculate the line equation from the initial and
final coordinates
    {
        this.Slope = (IY - FY) / (IX - FX);
        this.YIntersect = -this.Slope * IX + IY;
    }

    public double[] Punto_2rectas(double Slope, double YIntersect,
double Slope1, double Yintersect1)//We calculate the point of
intersection from two line equations
    {
        double[] position = new double[2];
        position[0] = ((YIntersect - Yintersect1) / (Slope1 -
Slope));
```

```
            position[1] = position[0] * Slope + YIntersect;
            return position;
        }
    }
}
```

A.2 Write excel function

```
// Function to write in an excel all the variables to later analyse
the program perfomance
//Code adapted from the webpage:
https://www.delftstack.com/es/howto/csharp/write-to-excel-in-csharp/

Excel.Application myexcelApplication = new Excel.Application();
Excel.Workbook myexcelWorkbook;
Excel.Worksheet myexcelWorksheet;

private void Simulador_Load(object sender, EventArgs e)
{
      HoraLocal = new DateTime(2018, 8, 1, 0, 00, 0);
      myexcelWorkbook = myexcelApplication.Workbooks.Add();
      myexcelWorksheet =
(Excel.Worksheet)myexcelWorkbook.Sheets.Add();
}

private void EscribirExcel()
{
      if (myexcelApplication != null)
      {
            //Write the title of each one of the parameters
            int x = 0;
            myexcelWorksheet.Cells[1, 1] = "id";
            myexcelWorksheet.Cells[1, 2] = "Arrival Date";
            myexcelWorksheet.Cells[1, 3] = "Callsign";
            myexcelWorksheet.Cells[1, 4] = "ETA";
            myexcelWorksheet.Cells[1, 5] = "ToA";
            myexcelWorksheet.Cells[1, 6] = "Airport origin";
            myexcelWorksheet.Cells[1, 7] = "Model";
            myexcelWorksheet.Cells[1, 8] = "Stand";
            myexcelWorksheet.Cells[1, 9] = "Departure Date";
            myexcelWorksheet.Cells[1, 10] = "Callsign";
            myexcelWorksheet.Cells[1, 11] = "ETD";
            myexcelWorksheet.Cells[1, 12] = "ToD";
            myexcelWorksheet.Cells[1, 13] = "Airport destination";
            myexcelWorksheet.Cells[1, 14] = "Time arrival stand";
            myexcelWorksheet.Cells[1, 15] = "Time departure stand";
            myexcelWorksheet.Cells[1, 16] = "Distance arrival";
            myexcelWorksheet.Cells[1, 17] = "Distance departure";
            myexcelWorksheet.Cells[1, 18] = "Time running arrival";
            myexcelWorksheet.Cells[1, 19] = "Time running departure";
            myexcelWorksheet.Cells[1, 20] = "Stop INT arrival";
            myexcelWorksheet.Cells[1, 21] = "Stop INT departure";
            myexcelWorksheet.Cells[1, 22] = "Stop Front arrival";
            myexcelWorksheet.Cells[1, 23] = "Stop Front departure";
            for (int i=2; i<=AirplaneListFINAL.Count+1;i++)
            {//Write the corresponding values to each one of the
airplanes
                  if (AirplaneListFINAL[x].IDArrival ==
myexcelWorksheet.Cells[i, 3].Value2 &&
```

```
AirplaneListFINAL[x].IDDeparture == myexcelWorksheet.Cells[i,
10].Value2)
                    {//if is in the same excel position as the list of
the program
                          myexcelWorksheet.Cells[i, 1] = i;
                          myexcelWorksheet.Cells[i, 5] =
AirplaneListFINAL[x].TimeArrival.ToString("HH:mm");
                          myexcelWorksheet.Cells[i, 6] =
AirplaneListFINAL[x].AirportOrigin;
                          myexcelWorksheet.Cells[i, 12] =
AirplaneListFINAL[x].Tiempo_Salida.ToString("HH:mm");
                          myexcelWorksheet.Cells[i, 13] =
AirplaneListFINAL[x].AirportDestination;
                          myexcelWorksheet.Cells[i, 14] =
AirplaneListFINAL[x].Tiempo_Llegada_Gate.ToString("HH:mm");
                          myexcelWorksheet.Cells[i, 15] =
AirplaneListFINAL[x].Tiempo_Salida_Gate.ToString("HH:mm");
                          myexcelWorksheet.Cells[i, 16] =
AirplaneListFINAL[x].DistanciaTotalLlegada/scale;
                          myexcelWorksheet.Cells[i, 17] =
AirplaneListFINAL[x].DistanciaTotalSalida/scale;
                          myexcelWorksheet.Cells[i, 18] =
AirplaneListFINAL[x].tiempo_corriendo_llegada;
                          myexcelWorksheet.Cells[i, 19] =
AirplaneListFINAL[x].tiempo_corriendo_salida;
                          myexcelWorksheet.Cells[i, 20] =
AirplaneListFINAL[x].tiempo_parado_interseccion_llegada;
                          myexcelWorksheet.Cells[i, 21] =
AirplaneListFINAL[x].tiempo_parado_interseccion_salida;
                          myexcelWorksheet.Cells[i, 22] =
AirplaneListFINAL[x].tiempo_parado_delante_llegada;
                          myexcelWorksheet.Cells[i, 23] =
AirplaneListFINAL[x].tiempo_parado_delante_salida;
                    }
                    else
                    {//We search for the position which is a certain
airplane in the excel
                          for (int c=0; c<AirplaneListFINAL.Count; c++)
                          {
                                if(AirplaneListFINAL[c].IDArrival ==
myexcelWorksheet.Cells[i, 3].Value2 &&
AirplaneListFINAL[c].IDDeparture == myexcelWorksheet.Cells[i,
10].Value2)
                                {
                                      myexcelWorksheet.Cells[i, 1] = i;
                                      myexcelWorksheet.Cells[i, 5] =
AirplaneListFINAL[c].TimeArrival.ToString("HH:mm");
                                      myexcelWorksheet.Cells[i, 6] =
AirplaneListFINAL[c].AirportOrigin;
                                      myexcelWorksheet.Cells[i, 12] =
AirplaneListFINAL[x].Tiempo_Salida.ToString("HH:mm");
                                      myexcelWorksheet.Cells[i, 13] =
AirplaneListFINAL[x].AirportDestination;
                                      myexcelWorksheet.Cells[i, 14] =
AirplaneListFINAL[x].Tiempo_Llegada_Gate.ToString("HH:mm");
                                      myexcelWorksheet.Cells[i, 15] =
AirplaneListFINAL[x].Tiempo_Salida_Gate.ToString("HH:mm");
                                      myexcelWorksheet.Cells[i, 16] =
AirplaneListFINAL[x].DistanciaTotalLlegada / scale;
                                      myexcelWorksheet.Cells[i, 17] =
AirplaneListFINAL[x].DistanciaTotalSalida / scale;
```

```
                                    myexcelWorksheet.Cells[i, 18] =
AirplaneListFINAL[x].tiempo_corriendo_llegada;
                                    myexcelWorksheet.Cells[i, 19] =
AirplaneListFINAL[x].tiempo_corriendo_salida;
                                    myexcelWorksheet.Cells[i, 20] =
AirplaneListFINAL[x].tiempo_parado_interseccion_llegada;
                                    myexcelWorksheet.Cells[i, 21] =
AirplaneListFINAL[x].tiempo_parado_interseccion_salida;
                                    myexcelWorksheet.Cells[i, 22] =
AirplaneListFINAL[x].tiempo_parado_delante_llegada;
                                    myexcelWorksheet.Cells[i, 23] =
AirplaneListFINAL[x].tiempo_parado_delante_salida;
                                }
                            }
                        }
                        x++;
                    }
                    //Close and save the excel file

      myexcelApplication.ActiveWorkbook.SaveAs(@"C:\Users\Usuario\Desk
top\copia de
seguridad\Universitat\TFG\TFG_versio2\TFG_versio1\bin\Debug\ResultatFi
nal.xls", Excel.XlFileFormat.xlWorkbookNormal);
                    myexcelWorkbook.Close();
                    myexcelApplication.Quit();
            }
}
```

A.3 Read taxiways excel and save in local variables function

```
//Function to read taxiways from an excel and save them in local
variables
//Code adapted from the webpage: https://coderwall.com/p/app3ya/read-
excel-file-in-c

Excel.Application xlApp = new Excel.Application();
Excel.Workbook xlWorkbook =
xlApp.Workbooks.Open(@"C:\Users\Usuario\Desktop\copia de
seguridad\Universitat\TFG\TFG_versio1\TFG_versio1\bin\Debug\Excel_TAXI
S_1.xlsx"); //Open the excel file
Excel._Worksheet xlWorksheet = xlWorkbook.Sheets[1]; //Read the first
sheet
Excel.Range xlRange = xlWorksheet.UsedRange;

int rowCount = xlRange.Rows.Count; //Count the number of rows
int colCount = xlRange.Columns.Count; //Count the number of columns


for (int i = 2; i <= rowCount; i++)//for each row in the excel file
{
      taxiways Taxiway1 = new taxiways(); //For each row we create a
new taxiway
      Taxiway1.Crear_taxi(xlRange.Cells[i, 1].Value2,
Convert.ToDouble(xlRange.Cells[i, 2].Value2),
Convert.ToDouble(xlRange.Cells[i, 3].Value2),
Convert.ToDouble(xlRange.Cells[i, 4].Value2),
Convert.ToDouble(xlRange.Cells[i, 5].Value2)); //We load the excel
info and put it on a fnction to create the taxiway
```

```
      TaxiList1.Add(Taxiway1); //We add this taxiway to a list with
all the taxiways
      Taxiway1.Dijkstra = DijkstraCont; //Set the position of the
taxiway inside the Dijkstra matrix
      DijkstraCont++;
}
TaxiList.SetTaxiList(TaxiList1);
xlWorkbook.Close();
```

ANNEX B.

B.1 Excel obtained with variables to analyse

id	Arrival Date	Callsign	ETA	ToA	Airport origin	Model	Stand	Departure Date	Callsign	ETD	ToD	Airport destination	Time arrival stand	Time departure stand	Distance arrival	Distance departure	Time running arrival	Time running departure	Stop INT arrival	Stop INT departure	Stop Front arrival	Stop Front departure
2	29/07/2018	VLG3913	16:25	16:25	PMI	320	234	01/08/2018	VLG3248	5:25	5:30	SPC	5:17	5:17	0	2762.2065	0	492	0	8	0	311
3	31/07/2018	VLG2211	8:50	8:50	SVQ	320	181	01/08/2018	VLG3522	19:50	19:47	IBZ	19:35	19:35	0	4871.435701	0	697	0	0	0	0
4	31/07/2018	RYR6385P	12:55	12:55	BIO	737	102	04/08/2018	RYR6341	4:05	0:00	FCO	0:00	0:00	0	0	0	0	0	0	0	0
5	01/08/2018	IBI6757	13:40	13:48	TFN	737	117	01/08/2018	IBK5602	14:10	14:08	DUS	13:53	13:53	1002.826297	6433.503523	227	932	0	0	0	0
6	31/07/2018	VLG2124	14:05	14:05	AGP	321	314	01/08/2018	VLG3902	5:00	4:57	PMI	4:49	4:49	0	3189.225568	0	496	0	0	0	0
7	31/07/2018	RYR3040	16:10	16:10	PRG	737	106	01/08/2018	RYR2918	4:15	4:12	BRU	3:56	3:57	0	6884.985139	0	943	0	0	0	0
8	31/07/2018	VLG1315	17:05	17:05	VLC	319	284	02/08/2018	VLG8036	3:55	0:00	ORY	0:00	0:00	0	0	0	0	0	0	0	0
9	31/07/2018	DLH1136	17:35	17:35	FRA	321	254	01/08/2018	DLH1139	4:50	4:47	FRA	4:40	4:40	0	2549.288445	0	452	0	0	0	0
10	31/07/2018	VLG1295	17:40	17:40	LCG	320	240	02/08/2018	VLG1701	15:00	0:00	MAD	0:00	0:00	0	0	0	0	0	0	0	0
11	31/07/2018	RYR6395	18:40	18:40	DUB	737	135	01/08/2018	RYR6374	4:30	4:30	BVA	4:13	4:13	0	5632.168304	0	946	0	0	0	37
12	31/07/2018	DLH1816	19:20	19:20	MUC	321	252	01/08/2018	DLH1809	7:10	7:06	MUC	7:00	7:00	0	2593.527213	0	389	0	0	0	0
13	31/07/2018	IBE2100	20:15	20:15	MAD	321	200	01/08/2018	IBE0801	6:00	5:58	MAD	5:47	5:47	0	3899.12178	0	665	0	0	0	0
14	31/07/2018	BAW482	20:20	20:20	LHR	320	214	01/08/2018	BAW477	5:10	5:07	LHR	4:59	4:59	0	3470.018146	0	522	0	0	0	0
15	31/07/2018	VLG3729	20:20	20:20	MAH	320	322	01/08/2018	VLG3738	4:35	4:33	MAH	4:24	4:24	0	3037.192144	0	551	0	0	0	0
16	31/07/2018	IBK5503	20:20	20:20	LGW	737	119	01/08/2018	IBK5510	5:40	5:46	EDI	5:22	5:22	0	6389.508098	0	1071	0	0	0	372
17	31/07/2018	EZY4419	20:25	20:25	LYS	320	101	01/08/2018	EZY1404	4:40	4:37	GVA	4:20	4:21	0	7073.239584	0	995	0	0	0	0
18	31/07/2018	EZY3921	20:30	20:30	CDG	320	95	01/08/2018	EZY8570	5:10	5:11	LGW	4:52	4:52	0	7313.825695	0	1131	0	0	0	0
19	31/07/2018	VLG6337	20:40	20:40	MPP	321	300	01/08/2018	VLG3710	5:00	4:59	MAH	4:50	4:50	0	3084.394961	0	517	0	0	0	0
20	31/07/2018	SWR1950	20:40	20:40	ZRH	320	425	01/08/2018	SWR1951	4:25	4:24	ZRH	4:15	4:16	0	2562.565961	0	538	0	0	0	0
21	31/07/2018	VLG7825	20:40	20:40	LGW	321	302	01/08/2018	VLG7822	5:35	5:43	LGW	5:23	5:24	0	3325.931719	0	579	0	85	0	571
22	31/07/2018	EZY1079	20:40	20:40	BSL	320	100	01/08/2018	EZY1076	5:40	5:48	BSL	5:22	5:22	0	7116.668054	0	1129	0	0	0	407
23	31/07/2018	KLM1681	20:45	20:45	AMS	737	220	01/08/2018	KLM1662	4:20	4:19	AMS	4:08	4:08	0	3422.011026	0	656	0	0	0	0
24	31/07/2018	THY1851	20:45	20:45	IST	321	270	01/08/2018	THY1852	3:45	3:40	IST	3:33	3:33	0	2685.373053	0	418	0	0	0	0
25	31/07/2018	VLG3919	20:45	20:45	PMI	320	414	01/08/2018	VLG902P	3:10	3:04	LPA	2:59	2:59	0	2285.12385	0	339	0	0	0	0
26	31/07/2018	AFR1448	20:55	20:55	CDG	320	222	01/08/2018	AFR1449	4:15	4:14	CDG	4:05	4:05	0	2991.331177	0	527	0	0	0	0
27	31/07/2018	VLG1673	20:55	20:55	SCQ	320	312	01/08/2018	VLG1515	5:20	5:22	NCE	5:10	5:10	0	3149.262596	0	529	0	0	0	183
28	31/07/2018	VLG6307	20:55	20:55	FCO	321	238	01/08/2018	VLG8460	5:15	5:15	LIS	5:04	5:04	0	2652.917193	0	474	0	31	0	126
29	31/07/2018	RYR3893	20:55	20:55	BUD	737	157A	01/08/2018	RYR9811	4:20	4:21	STN	4:06	4:06	0	5281.190577	0	888	0	0	0	0
30	31/07/2018	RYR7823	20:55	20:55	PIK	737	105	01/08/2018	RYR8136	4:40	4:39	IBZ	4:23	4:23	0	6896.46464	0	967	0	0	0	0
31	31/07/2018	RYR3143	20:55	20:55	VCE	737	122	01/08/2018	RYR6396	6:50	6:48	SVQ	6:32	6:32	0	6117.041649	0	885	4	0	0	8
32	31/07/2018	DLH1138	21:00	21:00	FRA	321	256	01/08/2018	DLH1137	6:00	6:00	FRA	5:51	5:51	0	2503.868189	0	465	0	0	0	46
33	31/07/2018	EZY8581	21:05	21:05	LGW	320	96	01/08/2018	EZY6026	6:10	6:07	BRS	5:50	5:50	0	7351.262339	0	1014	0	0	0	0
34	31/07/2018	VLG3519	21:10	21:10	IBZ	321	334	01/08/2018	VLG2474	3:35	3:29	SPU	3:23	3:23	0	2642.454689	0	373	0	0	0	0
35	31/07/2018	AZA078	21:15	21:15	FCO	321	216	01/08/2018	AZA075	4:25	4:26	FCO	4:15	4:15	0	3342.36879	0	603	24	0	0	84
36	31/07/2018	RYR6356	21:15	21:15	BVA	737	104	01/08/2018	RYR128	4:40	4:41	HAM	4:24	4:24	0	6860.381285	0	989	0	0	0	0
37	31/07/2018	VLG8991	21:20	21:20	BRU	320	330	01/08/2018	VLG7834	5:00	5:00	LGW	4:53	4:53	0	2722.15606	0	472	0	0	0	11
38	31/07/2018	VLG1309	21:20	21:20	MRS	320	236	01/08/2018	VLG3712	5:20	5:25	MAH	5:12	5:12	0	2699.860003	0	577	0	0	0	186
39	31/07/2018	VLG2118	21:20	21:20	AGP	320	171	01/08/2018	VLG1702	5:30	5:36	VGO	5:15	5:15	0	4874.806649	0	835	3	0	0	386
40	31/07/2018	IBK5523	21:20	21:20	OSL	737	116	01/08/2018	IBK5620	6:00	6:02	HEL	5:45	5:45	0	6407.78281	0	958	0	0	0	35
41	31/07/2018	IBK5575	21:25	21:25	CPH	737	118	01/08/2018	IBK5690	4:35	4:31	DBV	4:16	4:16	0	6421.019363	0	941	0	0	0	0
42	31/07/2018	VLG3719	21:30	21:30	MAH	321	310	01/08/2018	VLG6400	5:05	5:02	VCE	4:53	4:53	0	3105.193918	0	527	8	0	0	15
43	31/07/2018	VLG8023	21:35	21:35	ORY	320	282	01/08/2018	VLG1302	4:50	4:49	ALC	4:42	4:42	0	2652.854652	0	436	0	0	0	0
44	31/07/2018	IBK5555	21:35	21:35	ARN	737	115	01/08/2018	IBK5550	6:10	6:05	ARN	5:50	5:51	0	6372.261672	0	887	0	0	0	0
45	31/07/2018	VLG3547	21:35	21:35	IBZ	320	172	01/08/2018	VLG7010	5:00	5:04	MAD	4:51	4:51	0	4917.857956	0	754	0	0	0	13
46	01/08/2018	RYR8140	21:40	21:43	IBZ	737	131	01/08/2018	RYR3063	22:10	22:05	TFN	21:48	21:52	1546.805963	5843.552174	273	796	0	0	0	0
47	31/07/2018	VLG1518	21:40	21:40	NCE	320	280	01/08/2018	VLG7723	3:55	3:52	DBV	3:45	3:46	0	2619.728095	0	388	0	0	0	0
48	31/07/2018	VLG7816	21:45	21:45	EDI	320	332	01/08/2018	VLG6168	5:10	5:13	BIA	5:03	5:03	0	2672.56767	0	555	0	0	0	34
49	31/07/2018	RYR6945	21:45	21:45	WMI	737	107	01/08/2018	RYR6332	4:30	4:28	SCQ	4:09	4:09	0	6822.495258	0	1042	0	0	0	78
50	01/08/2018	RYR9814	21:50	21:58	STN	737	133	01/08/2018	RYR6310	22:20	22:18	FUE	22:05	22:05	1685.661071	5724.891235	363	786	0	0	0	0
51	31/07/2018	VLG6403	21:50	21:50	VCE	320	266	01/08/2018	VLG6207	5:10	5:16	GVA	5:06	5:06	0	2897.590799	0	490	0	0	0	133
52	31/07/2018	VLG7841	21:50	21:50	LHR	320	410	01/08/2018	VLG8744	5:30	5:37	MLA	5:22	5:22	0	2470.799486	0	525	0	0	0	406
53	31/07/2018	VLG8657	21:55	21:55	PRG	320	260	01/08/2018	VLG8012	5:00	5:06	ORY	4:58	4:58	0	2528.308062	0	486	0	0	0	0
54	31/07/2018	VLG3901	21:55	21:55	PMI	321	412	01/08/2018	VLG2115	5:20	5:27	AGP	5:14	5:14	0	2432.416938	0	425	60	0	0	317
55	31/07/2018	VLG8463	22:00	22:00	LIS	321	202	01/08/2018	VLG8300	5:10	5:18	AMS	5:05	5:05	0	3847.55274	0	632	30	0	0	129
56	31/07/2018	RYR6366	22:00	22:00	BGY	737	113	01/08/2018	RYR2744	4:20	4:17	VNO	4:01	4:01	0	6837.700458	0	992	0	0	0	0
57	31/07/2018	RYR6986	22:00	22:00	FCO	737	111	01/08/2018	RYR9366	4:15	4:16	BLQ	4:00	4:00	0	6669.637076	0	965	0	0	0	0
58	31/07/2018	RYR4546	22:05	22:05	OPO	737	134	01/08/2018	RYR2243	6:00	6:03	RAK	5:48	5:48	0	5687.529098	0	939	0	0	0	0
59	31/07/2018	VLG6503	22:10	22:10	NAP	320	262	01/08/2018	VLG8652	4:25	4:23	PRG	4:14	4:14	0	2572.256431	0	543	0	0	0	0
60	31/07/2018	RYR6303	22:10	22:10	FUE	737	114	01/08/2018	RYR6353	4:15	4:10	BGY	3:55	3:56	0	6428.90909	0	890	0	0	0	0
61	31/07/2018	VLG8305	22:10	22:10	AMS	321	230	01/08/2018	VLG2010	5:20	5:29	GRX	5:14	5:14	0	2855.300282	0	511	128	0	0	351
62	31/07/2018	VLG1291	22:20	22:20	LCG	320	264	01/08/2018	VLG6246	5:00	4:55	ZRH	4:48	4:48	0	2744.247574	0	408	0	0	0	0
63	31/07/2018	VLG6771	22:20	22:20	PSA	320	248	01/08/2018	VLG8476	5:50	5:53	OPO	5:41	5:41	0	2223.905313	0	513	0	0	0	222
64	31/07/2018	VLG2017	22:20	22:20	GRX	321	232	01/08/2018	VLG2974	5:10	5:20	NTE	5:08	5:08	0	2818.484068	0	511	73	0	0	179
65	31/07/2018	TAP1048	22:20	22:20	LIS	319	218	01/08/2018	TAP1049	4:40	4:44	LIS	4:34	4:34	0	3435.800819	0	576	0	0	0	0

B.2 Excel of stand data

Name	IX	IY	FX	FY
298	254	643	259	653
296	263	640	268	649
295	266	638	271	647
294	270	636	275	646
292	278	633	284	643
291	283	630	288	641
290	288	628	293	638
288	294	625	299	635
287	298	623	303	633
286	303	621	308	631
285	313	618	317	627
284	313	618	317	627
282	321	613	327	624
281	324	611	330	622
280	328	609	333	620
278	336	602	346	612
277	340	600	355	608
276	377	594	359	593
274	334	587	351	581

B.3 Taxiways Data Excel

Name	IX	IY	FX	FY
L14	199	528	231	514
L13	231	514	280	492
L12	280	492	310	479
L11_vert_sup	300	455	305	467
L11_vert_cent	305	467	310	479
L11_vert_inf	310	479	323	509
L11_horit	323	509	349	498
L10	349	498	370	489
L9	370	489	392	481
L8_esq	392	481	412	470
L8_dret	412	470	440	460
K11_esq	290	668	304	662
K11_dret	304	662	312	659
K10	312	659	336	647
K9_esq	336	647	538	638
K9_dret	538	638	377	628
K8	377	628	414	612
K7	414	612	458	592
K6	458	592	492	574

B.4 Excel of data of the different aircraft models

Model	Length	Width	Takeoff Weight	Landing Weight	Takeoff Distance	Landing Distance
319	33.8	34.1	75500	62500	1950	1450
320	37.57	35.8	78000	66000	2090	1400
321	44.51	34.1	93500	77800	1707	1540
332	58.8	60.3	230000	182000	2770	1730
333	63.6	60.3	242000	187000	2770	1730
346	75.36	63.45	344730	240404	3140	2000
359	66.9	64	280000	207000	2438	1219
388	72.73	79.75	575000	395000	2749	1524
733	33.4	28.9	62820	52880	2286	1433
734	36.4	28.9	68039	56245	2540	1540
735	31.01	28.88	68000	49895	2650	1400
737	33.63	35.79	70080	58604	2000	1400
738	39.5	34.31	78245	65317	2316	1372
739	42.1	35.79	85139	71350	2300	1700
744	70.66	64.9	396890	295740	2815	1905
752	47.3	38	99800	89800	2070	1400
763	54.94	47.57	158758	136078	2408	1798
767	61.4	51.9	204120	158757	2896	2134
772	63.7	60.9	297550	213180	2438	1554
777	73.86	64.8	351535	260816	3000	1800
788	57	60	227930	172000	2600	1700
789	62.81	60.12	254011	192776	2800	1520
E90	36.24	28.72	50300	43000	2056	1323
E95	38.65	28.72	52290	45800	1432	1275

ANNEX C.

C.5 Barcelona-El Prat airport stands