



UNIVERSITAT POLITÈCNICA DE CATALUNYA  
BARCELONATECH  
Escola d'Enginyeria de Telecomunicació  
i Aeroespacial de Castelldefels

# TREBALL DE FI DE GRAU

**TFG TITLE: Airport Digital Mock Up**

**DEGREE: Masters Degree in Aerospace Science and Technology**

**AUTHOR: Girish Shiva Prasanna Raju Sardar**

**:**

**SUPERVISOR: Jordi Pons Prats**

**DATE: October 24, 2022**



**Title :** Airport Digital Mock Up

**Author:** Girish Shiva Prasanna Raju Sardar

**Supervisor:** Jordi Pons Prats

**Date:** October 24, 2022

## Overview

The accessibility of the Aeronautical Information is available in format such as AIXM, these formats are most suitable for trained eyes and intended for the purpose of Aerodrome monitoring, Landing & Takeoff procedures and Navigation systems on full fledge devices designed and built for AIXM only.

Let's say if we want to develop an Aerodrome game where we can land an aeroplane in an airport whose characteristics are close to the real ones on a cross platform friendly format, it is not possible with the current available formats. In this thesis we take advantage of ENAIRE Spain's Air Navigation and Aeronautical Information service provider, availing Aeronautical Information Publication service <https://aip.enaire.es/AIP/#LEPA/LESJ>, where they provide data regarding to Aerodromes in Spain in the form of PDF's and CSV's.

The pdfs are usually used by pilots and co pilots as a reference to identify hotspots, taxiways status and obstacles inside the aerodrome, CSV files serve as a data to feed GIS platforms such as QGIS and mostly used by researchers and other people who are involved in aeronautical domain to identify the obstacles in and around the Aerodromes.

We used Python libraries such as Pandas, Camelot to extract and convert data from PDF which are in the form of tables. The extracted information which contains Geo Locations and other critical information such as area, altitude and Runway's important properties such as Clear way, Strip dimensions & geographic reference point is combined with the Obstacles data such as Geo Location and other characteristics such as altitude, Lighting, Marking and other available information of obstacles such as Trees, vegetation, Signal Towers into a geojson format.

The Generated output geojson files are viewed on a 2-Dimensional map on <http://geojson.io/#map=2/20.0/0.0>, which has two split panes where we can view the geojson information and its geolocation against each other.



I firstly thank my Parents for helping me to pursue my dreams of being an engineer and to solve problems, I thank my Master's coordinator Ricard, my Supervisor Jordi and all my Professors who have helped me to attain knowledge to solve engineering problems. I thank all my dearest friends who have extended their support when it was necessary.



# CONTENTS

<b>CHAPTER 1. Introduction</b> . . . . .	<b>1</b>
1.1. Motivation . . . . .	1
1.2. Aim and Objectives . . . . .	1
1.3. Structure of the Thesis . . . . .	2
1.4. State of Art . . . . .	3
1.4.1. ENAIRE . . . . .	3
1.5. How to obtain required AIP documents . . . . .	4
<b>CHAPTER 2. Data Formats</b> . . . . .	<b>7</b>
2.1. AIXM . . . . .	7
2.1.1. GML . . . . .	9
2.1.2. AIXM Code Example . . . . .	10
2.1.3. Disadvantages of AIXM . . . . .	13
2.2. Shape Files . . . . .	13
2.2.1. .shp files . . . . .	14
2.2.2. .shx files . . . . .	14
2.2.3. .dbf . . . . .	14
2.2.4. Disadvantages of Shape files . . . . .	14
2.3. GeoJson . . . . .	14
2.3.1. Disadvantages of GeoJson . . . . .	16
<b>CHAPTER 3. Approach and Solution</b> . . . . .	<b>17</b>
3.1. GeoJson over Other Formats . . . . .	17
3.1.1. GeoJson advantages . . . . .	17
3.1.2. Other Formats Disadvantages . . . . .	18
3.2. Tools of choice . . . . .	18
3.2.1. Stream Method . . . . .	20
3.2.2. Lattice Method . . . . .	21
3.3. Converting the obstacles CSV file data to json . . . . .	22
3.4. GeoJSON package . . . . .	23

<b>3.5. Complete solution architecture.</b>	<b>25</b>
3.5.1. Automated	25
3.5.2. Manual actions	26
<b>3.6. How to use Automated Program</b>	<b>26</b>
<b>3.7. How to Perform Manual Actions</b>	<b>27</b>
<b>CHAPTER 4. SourceCode Architecture</b>	<b>29</b>
<b>4.1. Runways and Terminal Stands Data Extractions</b>	<b>29</b>
<b>4.2. Obstacles Data Extraction</b>	<b>34</b>
<b>4.3. Files Organisation</b>	<b>35</b>
4.3.1. Source Code files	35
4.3.2. Data files	36
<b>CHAPTER 5. Output</b>	<b>37</b>
<b>5.1. Mallorca Airport</b>	<b>37</b>
5.1.1. Program Generated Output	37
5.1.2. Manually Generated Output	42
<b>5.2. Coruña Airport</b>	<b>44</b>
5.2.1. Program Generated Output	44
5.2.2. Manual Generated Output	49
<b>5.3. Seville Airport</b>	<b>50</b>
5.3.1. Program Generated Output	50
5.3.2. Manual Generated Output	56
<b>Conclusions</b>	<b>59</b>
<b>Bibliography</b>	<b>61</b>



# LIST OF FIGURES

1.1 Runway details table in Aerodrome document . . . . .	5
1.2 AIP website . . . . .	5
1.3 Airport files . . . . .	5
2.1 AIXM- Feature . . . . .	7
2.2 AIXM- FeatureToObject . . . . .	8
3.1 gejson default view . . . . .	20
3.2 lattice method default view1 . . . . .	21
3.3 lattice method default view2 . . . . .	21
3.4 lattice method default view3 . . . . .	21
3.5 lattice method default view4 . . . . .	22
3.6 lattice method default view5 . . . . .	22
3.7 Block Digram of Automated steps. . . . .	25
3.8 [Left]- Runways Thresholds, [Center]- Terminal Stands ,[Right]- Obstacles . . . . .	25
3.9 Block Digram of Manual steps. . . . .	26
3.10[Left]- Taxiways, [Right]- Hotspots . . . . .	26
4.1 Source Code Files Sturcture . . . . .	36
4.2 Data Files Sturcture . . . . .	36
5.1 Mallorca Airport Runways Thresholds . . . . .	37
5.2 Mallorca Airport Terminal Stands . . . . .	39
5.3 Mallorca Airport Obstacles . . . . .	42
5.4 Mallorca Airport Taxiways . . . . .	43
5.5 Mallorca Airport Hotspots . . . . .	43
5.6 Coruna Airport Runways . . . . .	44
5.7 Coruna Airport Terminalstands . . . . .	45
5.8 Coruna Airport Obstacles . . . . .	47
5.9 Coruna Airport Taxiways . . . . .	50
5.10Seville Airport Runways . . . . .	51
5.11Seville Airport Terminal Stands . . . . .	52
5.12Seville Airport Obstacles . . . . .	54
5.13Seville Airport Taxiways . . . . .	57
5.14Seville Airport Hotspots . . . . .	57



# CHAPTER 1. INTRODUCTION

## 1.1. Motivation

Since my childhood I have always been passionate about technology, it might be because my dad is a software engineer and since I have memory, I remember him showing me how to develop things and how to find solutions for the different technical problems that can arise. I am not sure the reason but as soon as I had to decide what to do in my bachelors, I was sure I wanted to study software and telecommunications engineer. It was during university that I realized how passionate I was about space, and I decided to do my master's in aerospace science and technology. During the classes I had on the master, I learned a lot of new skills and acquired new and useful data, more concrete data related to how the airport data works. When I got more interested in the topic and willing to research for my side, I realized that the tools are not user friendly, and I needed some special tools to be able to gather the data. With this issue I realized how important will it be to create a unique platform that was user friendly and gathered all the information for everyone.

My main motivation to write this thesis is to make data accessible and at the same time ensure that airports can have a more user-friendly platforms to gather and display the data.

## 1.2. Aim and Objectives

The goal of this thesis is to make more accessible and cross platform supporting data of Aerodrome information such as Runways, Taxiways, Terminal stands, and Obstacles in and surrounding the aerodromes. A limited amount of information is available at official Information exchange (AIXM) for the commercial purposes and not the everything is open source; however, we have ENAIRE's AIS where they provide data regarding to Aerodromes in Spain in the form of PDF's and CSV's.

This accessibility of the data regarding aerodrome information will help new applications to surface, which will be more accessible to people involved in aeronautical field and people that could be interested in the data. But this availability will be without the need of special equipment. To reach this goal the key aspect is to digitalize the information sourced from the aeronautical ENAIRE service provider, in a format that is cross platform friendly so everyone can have access without special tools.

To help reach this accessibility the aim is to develop a code which converts the current data format or directly for data sourced that we have for aerodrome information to an alternative chosen format that is cross platform friendly, and everyone can have access to it regardless of the tools.

To reach this main goal, the thesis objectives have been determined as the following ones:

- Know currently used data format and its sources.
- Propose alternative to the currently used data format.
- Find Advantages and Disadvantages of Proposed format over other current format.

- Develop code which converts current format or directly from the data source to alternative chosen format.
- Provide examples as proof of work for the developed code.

### 1.3. Structure of the Thesis

The structure of the thesis follows a concrete order to ensure the understanding of the topic being analyzed. It is divided in five main chapters with subchapters according to the development of the point topic.

The first point of chapter one is focused on the motivation that arise me to write this topic for the master's in aerospace science and technology thesis. With a further overview of the aim and the objectives that will be discussed and analyzed during all the paper. To finish with the introduction chapter, a detailed explanation of ENAIRE's AIP and AIS services is displayed. ENAIRE is the Public Business Entity that manages air navigation in Spain and Western Sahara, certified to provide route, approach and aerodrome control services.

During the second chapter the reader will be able to find the main data formats available their disadvantages and the following chapter contains the proposal with the key advantages and disadvantages of the proposed format over counter formats and a further analysis of the tools of choice. The key tools are Camelot Python package, with two methods, Stream and Lattice which are deeply explained in the subsections. And the other key tool is GeoJSON which is explained as a tool and later develop as a package in the point four of the chapter two. Along chapter two the reader will also be able to understand how to covert the obstacles CSV file data to json and an overview of the complete solution architecture, understanding the data format proposal for the extracted information

In chapter four the SoruceCode architecture is explained in detail to have a better understanding of the runways and terminal stands of the data extraction, as well as the obstacles. In addition to the files organization with the source code files and the data files used to do the coding.

For chapter five the output has been displayed explaining the geojson information and its geolocation with three examples of the airports of La Coruña, Palma de Mallorca and Sevilla. This three airports will have the analysis of the Runways, Taxiways, Terminal stands, and Obstacles in and surrounding the aerodromes applying the code and the manual steps.

Finally, the conclusion will be discussed together with the raised points from the first chapter reading aim and objectives of the thesis to see if they have been fulfilled and the key outcomes extracted from this thesis analysis. Together with the pack up of all the annexes and the sources used to do the investigation.

## 1.4. State of Art

### 1.4.1. ENAIRE

ENAIRE is the navigations and aeronautical information service provider in Spain. It is a public business entity reporting to Ministry of Transports & Mobility. ENAIRE provides services such as en-route approach, Aerodrome ATC services along with flight information, alerts and consulting services. In many such services ENAIRE also provides a services named as Aeronautical Information service (AIS) which provides the advice and informations required for operational safety, regularity and efficiency of air navigation. In this Aeronautical Information service, following types of information is provided,

- General : Five Sections of information useful for administration and explanatory kind of information is provided.
- En-Route : Seven sections of information concerning Airspace uses and Air Traffic Services Rules and Regulations and important airspace description is provided.
- Aerodromes : Information divided into four sections of Aerodromes, heliports of Spain and its Territories which is used for Geographical and administrative needs, Physical characteristics and associated Charts.

The above mentioned Categories are further divided into,

#### 1.4.1.1. *General Category*

1. Gen 0 Contains General Information such as Records of AIP Amendments and supplements and CheckList of AIP pages.
2. Gen 1 which contains National Regulations and Requirements, such as Designated Authorities, Entry transit and departure of Aircraft, Passengers, Crew and Cargo.
3. Gen 2 contains Tables and Codes regarding Measuring systems, Aircraft markings and abbreviations, symbols, Indicators, Conversion of units of measurements used in Aeronautical Information Products
4. Gen 3 displays Services such as AIS, Charts, Air Traffic services, Communication and Navigation services, Search and Rescue and Operative air traffic management.
5. Gen 4 has the information regarding Charges for Aerodromes, Heliports and Air Navigation services.

#### 1.4.1.2. *En-Route*

1. ENR 1 contains General Rules and Procedures such as Visual ,Instrument Flight Rules, ATS airspace classifications and description, Holding approach and departure procedures, Flight planning, Unlawful Interference and Incidents related information in PDF format.

2. ENR 2 contains Air Traffic Services Airspace information regarding the Spanish and other regulated airspace along with Air Traffic services contingency planning.
3. ENR 3 contains Air Traffic services routes such as Navigation Routes, Upper and Lower routes, Helicopter routes and also holding pattern incompatibilities in terminal areas.
4. ENR 4 displays Radio navigation aids /systems. which include Special Navigation systems, Global Navigation satellite system, Aeronautical ground lights and Name-code designators for significant points.
5. ENR 5 Navigation warnings details such as Air Navigation obstacles, Aerial Sporting and recreational activities, Bird Migration and areas with sensitive fauna, Restricted areas to photographic flight and other activities of dangerous nature and other potential hazards.
6. ENR 6 displays the actual En-route charts which contains TMA for major and busy airports such as Barcelona, Madrid, Galicia and Sevilla and other details such as Mandatory use of Transponder zones charts.

#### 1.4.1.3. Aerodromes AD

1. AD 1 displays the Introduction documents regarding Airports and Heliports availability, Rescue and fire fighting services, runway surface condition assessment, snow plans and Status of certification of Aerodromes and Heliports.
2. AD 2 Displays the list of Airports Index.
3. AD 3 Display the list of Heliports Index.

As we are looking at how AIP documents are organised in AIP website, Lets utilise the opportunity how to obtain the documents needed to feed to the automated program which was developed for this thesis.

## 1.5. How to obtain required AIP documents

As an example lets chose A Coruña Airport and obtain the following files,

1. AD 2 Aerodrome Data.
2. AD 2 item : 10 Aerodrome obstacles.
3. AD 2 PDC 1.

Download the above mentioned list of files in PDF format except for Aerodrome Obstacles file which will be available in CSV format.

Make sure in you find the table with the runways details as shown in the figure below.

12. CARACTERÍSTICAS FÍSICAS DE LA PISTA				RUNWAY PHYSICAL CHARACTERISTICS						
RWY	Orientación Direction	DIM (m)	THR PSN	THR ELEV TDZ ELEV	SWY (m)	CWY (m)	Franja (m) Strip (m)	OFZ	RESA (m)	RWY/SWY SFC PCN
03	030.94°GEO 033° MAG	2188 x 45	431730.11N 0082308.74W	THR: 100 m / 329 ft	No	300 x 150	2308 x 150	No	240 x 150 (1)	RWY: ASPH PCN 39/F/B/W/T SWY: No
21 (2) (3)	210.95°GEO 213°MAG	2188 x 45 (4)	431830.91N 0082218.84W	THR: 82.8 m / 272 ft TDZ: 85.6 m / 281 ft	No	300 x 150	2248 x 150 (5)	Sí / Yes	240 x 150	RWY: ASPH PCN 39/F/B/W/T SWY: No

Figure 1.1: Runway details table in Aerodrome document

Please follow below steps to Download the files,

1. Go to AIP website : <https://aip.enaire.es/AIP/AIP-en.html>
2. After the page load Click on Part 3 - Aerodromes (AD) as shown in below figure.

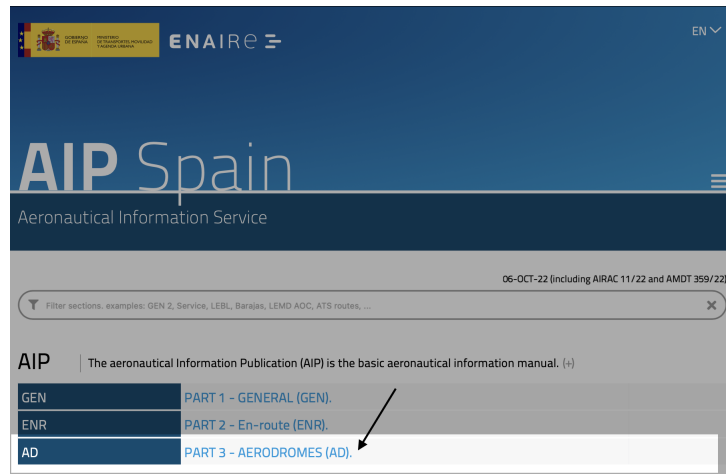


Figure 1.2: AIP website

3. It displays the list of Aerodromes, Please select the airport of interest and click, it will take you to the respective Airport documents.
4. Now Click on Aerodrome, Aerodrome obstacles and ADC files individually and download them.




AD 2 LECO	<a href="#">Aerodrome data.</a>	  
AD 2 10 LECO	<a href="#">Item 10: AERODROME OBSTACLES.</a>	  
AD 2 LECO ADC 1	<a href="#">ADC</a>	  

Figure 1.3: Airport files





# CHAPTER 2. DATA FORMATS

## 2.1. AIXM

AIXM is information exchange medium for Aeronautics standardised under EURO CONTROL, ICAO & EC. It is designed using UML to share information in the form of Features, attributes and Data types. It strongly depends on the Concept of Temporality (timely changing data), the data is used to represent in GML (Geographical Markup Language) to represent the AIXM data graphically. UML is a Standard Modelling language in Software Engineering domain, It helps to visually represent a design system or a business implementation.

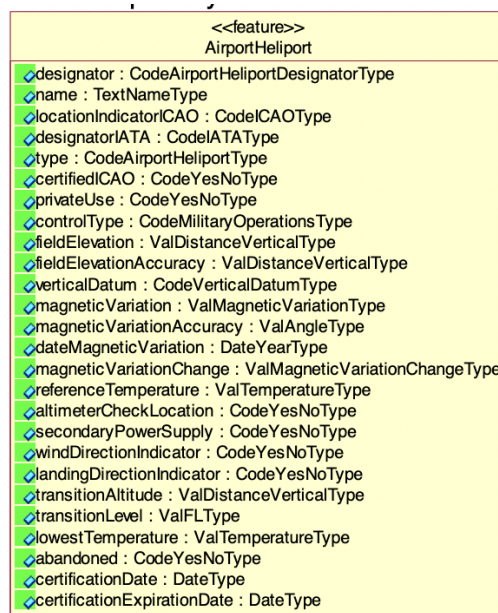


Figure 2.1: AIXM- Feature

AIXM has objects and attributes known as properties, All the Objects and attributes are connected to features, Features are fundamental in AIXM and describe real world entities, Features are dynamic meaning they change in time hence features are represented in 'Time slices', Figure:2.1 is a time slice Feature where we can see different characteristics, which further have a relation to an object which describes in detail about the feature property with which it is associated. To explain in a simple example using Figure:2.2, An Aerodrome is a Feature and the city name in which the Aerodrome is located is an Object and the life of this object is only associated with the Feature it is associated with outside of this Feature the object 'city name' does not exist In this way various details of an Aerodrome are represented using objects.

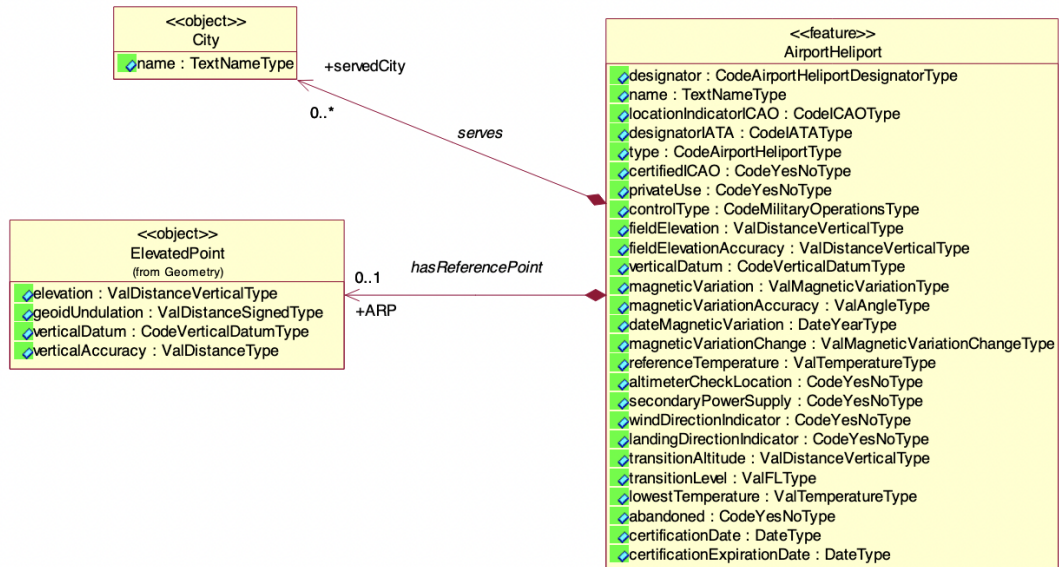


Figure 2.2: AIXM- FeatureToObject

AIXM has a reduced form which can be readable which would be helpful for airmen, there is also an extensive form which is not so great for reading but very useful for developing applications. Latest format of AIXM includes GML into it and represents the objects with ids and co-ordinates of location. There are many parses in open-source community to parse XML format of AIXM into python and other open-source GIS platforms 'QGIS' which is commonly used by airmen.

ENAIRES provides information about aerodrome obstacles and en routes in AIXM format only which is strict in the schema and not very flexible, AIXM based on XML language.

#### 2.1.0.1. XML

XML - Extensible Markup Language is a software, hardware independent tool for storing data and transporting data. It transfers the data in plain text format so that it is readable by machines, humans without requiring any additional equipment. The information can be distinguished by using tags and enclosing the data in between those tags as required. Below figure code snippet an example of XML.

```

<note>
  <date>2015-09-01</date>
  <hour>08:30</hour>
  <to>Tove</to>
  <from>Jani</from>
  <body>Don't forget me this weekend!</body>
</note>
  
```

The above code snippet is a Mail, where it contains date, time, Receiver, Sender and the Message. It starts with a note tag, '<note>' and ends with '</note>', please bear that the closing tag has '/' added to indicate that the note tag is closing tag. Everything else

is in between the note tag like an envelope. The data and time are enclosed in data and hour tags followed by receiver information in between 'to' and 'from' tags then finally the message body is added in between 'body' tag. In this way a format can be proposed and ask the candidates you share the information to follow the same format which arises the concept of 'Schema'. A Schema is a definition of the structure of document it is used to define how the elements and attributes can appear in a document and how many number and types of elements and their default values, a XML schema is also known as XSD which stands for XML Structure Definition, Some of syntax rules are listed below,

1. The document must begin with XML declaration shown in code snippet below,

```
\<?xml version="1.0" encoding="UTF-8" standalone="no" ?\>
```

2. Start tags must have matching end tags, example : "i>note; i/>note;"
3. All elements are case sensitive.
4. All attribute values must be quoted.
5. All Elements must be properly nested.

In this way, in XML we can transfer information in text format with a defined structure called as schema or XSD in the case of schema.

### 2.1.1. GML

GML- Geography Markup Language, it is written in XML but the rules are defined by Open Geospatial Consortium OGC to represent Geographical features. It serves as Modelling language and exchange format for geographic systems and across the geographic transactions on internet. GML contains a set of Primitives which are used to build schemas, these primitives include, Feature, Geometry, Topology, Coordinate Reference System, Time, Unit of Measure, Directions, Observations, Coverage. Now let see how a single Geographic coordinates point is represented in GML,

```
<gml:Point gml:id="p21" srsName="http://www.opengis.net/def/crs/EPSSG/0/4326">
  <gml:coordinates>45.67, 88.56</gml:coordinates>
</gml:Point>
```

In the above code snippet you can see that a single point is represented using two tags, which are "gml coordinates","gml Point" In the same way instead of point we can also have a Line string which are continuous points forming as a string, code snippet below shows it,

```
<gml:LineString gml:id="p21" srsName="http://www.opengis.net/def/crs/EPSSG/0/4326">
  <gml:coordinates>45.67, 88.56 55.56,89.44</gml:coordinates>
</gml:LineString >
```

## 2.1.2. AIXM Code Example

AIXM uses both the concepts of XML and GML and take advantage to represent Aeronautical data which includes, Aerodromes and NavAid details. Below code snippet is a part of AIXM code which does not represent any physical entity but just as an example.

```
<message:AIXMBasicMessage xmlns:message="http://www.aixm.aero/schema/5
    .1.1/message"
  xmlns:gts="http://www.isotc211.org/2005/gts"
  xmlns:gco="http://www.isotc211.org/2005/gco"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:gml="http://www.opengis.net/gml/3.2"
  xmlns:gss="http://www.isotc211.org/2005/gss"
  xmlns:aixm="http://www.aixm.aero/schema/5.1.1"
  xmlns:gssr="http://www.isotc211.org/2005/gssr"
  xmlns:gmd="http://www.isotc211.org/2005/gmd"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.aixm.aero/schema/5.1.1/message http://
    www.aixm.aero/schema/5.1.1/message/
    AIXM_BasicMessage.xsd"

  gml:id="M0000001">
  <gml:boundedBy>
    <gml:Envelope srsName="urn:ogc:def:crs:EPSG::4326">
      <gml:lowerCorner>-32.08861111111111 -47.0</gml:lowerCorner>
      <gml:upperCorner>57.6908159699999996 52.42833333333333</gml:
        upperCorner>
    </gml:Envelope>
  </gml:boundedBy>
  <aixm:messageMetadata>
    <gmd:MD_Metadata id="MD_DONLON_MESSAGE">
      <gmd:characterSet>
        <gmd:MD_CharacterSetCode
          codeList="http://standards.iso.org/ittf/
            PubliclyAvailableStandards/
            ISO_19139_Schemas/resources/
            Codelist/gmxCodelists.xml#
            MD_CharacterSetCode"
          codeListValue="utf8">utf8</gmd:MD_CharacterSetCode>
      </gmd:characterSet>
      <gmd:contact>
        <gmd:CI_ResponsibleParty>
          <gmd:organisationName>
            <gco:CharacterString>EUROCONTROL</gco:CharacterString>
          </gmd:organisationName>
          <gmd:contactInfo>
            <gmd:CI_Contact>
              <gmd:onlineResource>
                <gmd:CI_OnlineResource>
                  <gmd:linkage>
                    <gmd:URL>http://www.eurocontrol.int</gmd:URL>
                  </gmd:linkage>
                </gmd:CI_OnlineResource>
              </gmd:onlineResource>
            </gmd:CI_Contact>
          </gmd:contactInfo>
          <gmd:role>
            <gmd:CI_RoleCode
```



```

        <gmd:CI_OnlineResource>
          <gmd:linkage>
            <gmd:URL>www.aixm.aero</gmd:URL>
          </gmd:linkage>
          <gmd:name>
            <gco:CharacterString>AIXM 5.1
              website</gco:CharacterString>
          </gmd:name>
        </gmd:CI_OnlineResource>
      </gmd:onlineResource>
    </gmd:CI_Contact>
  </gmd:contactInfo>
  <gmd:role>
    <gmd:CI_RoleCode
      codeList="http://standards.iso.org/ittf/
        PubliclyAvailableStandards/
        ISO_19139_Schemas/resources/
        Codelist/gmxCodelists.xml#
        CI_RoleCode"
      codeListValue="pointOfContact">pointOfContact</gmd:
        CI_RoleCode>
    </gmd:role>
  </gmd:CI_ResponsibleParty>
</gmd:pointOfContact>
<gmd:resourceConstraints>
  <gmd:MD_Constraints>
    <gmd:useLimitation>
      <gco:CharacterString>To be used for demonstration
        purposes only! Any
        operational usage is prohibited!</gco:CharacterString
      >
    </gmd:useLimitation>
  </gmd:MD_Constraints>
</gmd:resourceConstraints>
<gmd:language>
  <gco:CharacterString>eng</gco:CharacterString>
</gmd:language>
<gmd:topicCategory>
  <gmd:MD_TopicCategoryCode>transportation</gmd:
    MD_TopicCategoryCode>
</gmd:topicCategory>
<gmd:extent>
  <gmd:EX_Extent>
    <gmd:geographicElement>
      <gmd:EX_GeographicBoundingBox>
        <gmd:westBoundLongitude>
          <gco:Decimal>-47.0</gco:Decimal>
        </gmd:westBoundLongitude>
        <gmd:eastBoundLongitude>
          <gco:Decimal>52.4283333333333</gco:Decimal>
        </gmd:eastBoundLongitude>
        <gmd:southBoundLatitude>
          <gco:Decimal>57.690815969999996</gco:Decimal>
        </gmd:southBoundLatitude>
        <gmd:northBoundLatitude>
          <gco:Decimal>-32.0886111111111</gco:Decimal>
        </gmd:northBoundLatitude>
      </gmd:EX_GeographicBoundingBox>
    </gmd:geographicElement>
  </gmd:EX_Extent>
</gmd:extent>

```

```
        </gmd:geographicElement>
      </gmd:EX_Extent>
    </gmd:extent>
  </gmd:MD_DataIdentification>
</gmd:identificationInfo>
</gmd:MD_Metadata>
</aixm:messageMetadata>
```

The above code snippet is an example of AIXM to represent few geographic coordinates which are nested way into nearly 6 to 7 level of elements. The document should always have a header which describes the version of AIXM and Enclose all the propriety information before starting to add the actual information.

### 2.1.3. Disadvantages of AIXM

1. A separate front end software is needed to view the data as it is only a transfer data language and does not have any functional abilities.
2. It is difficult to develop applications because the AIXM data is proprietary and is available as Enterprise version and monetised.
3. The Data source of AIXM is also from ENAIRE's AIS and AIP services, which is freely available to public.
4. The development and updates to the AIXM versions are very slow due to its dependency on Enterprise Hardware so frequent updates cannot be pushed.
5. Some data such as NavAids are required for Airmen but for general purpose applications they are not useful unless specifically needed.

## 2.2. Shape Files

Another alternative to AIXM is 'Shape files', developed by ESRI - Environmental Systems Research Institute to use it for their own proprietary products such ArcGIS application which is Geographic Information systems application. Shape file is a vector storage format in digital form to store geographic location and associated information as attributes. It stores geometry data in primitive geometry shapes like points, lines and polygons. The shape file format consists of collection of files which have following extensions '.shp', '.shx', '.dbf' which are mandatory for a shape file to satisfy its functionality there are other files which compliment with additional information but not strictly necessary on the functionality stand point of view. You can have only 1 type of shape in a single file and each different shape is stored as a layer and is disconnect from the each layer. A maximum of 2 Giga Bytes of file size can be tolerated with this format.

Let's dive in detail about the mandatory files,

### 2.2.1. .shp files

'shp' file is the main file and contains the feature geometry data. The data is stored as vector coordinates, the internal binary file which is a file which is associated with any format file usually containing the data in the format of 1's and 0's or machine level language which is useful for the computer, contains the key information regarding the shape such as variable length of the coordinates and type of variable it is stored in such as 'int', 'float', 'double'. each shape has its own type of variable.

### 2.2.2. .shx files

'shx' stands for Shape Index format, which contains the positional index of the feature geometry, .shx file contains the first 100 bytes of the .shp file information to avoid gaps in the file while editing or using the .shp file in GIS platform. However if the .shx file is corrupted or lost, we can retain the file using .shp file.

### 2.2.3. .dbf

.dbf stands for shape file attribute format, this file store the attribute information about the geometry in the main file, for example if we have a single coordinate point in .shp file, we can store the information regarding the temperature or any other information relevant for the purpose. This file serves as a database file and has 255 maximum number of fields

On a final note about the shape files, Shape files were most popular with ArcGis which is freemium tool for GIS Geographic Information Services applications, however they posed huge problems with the capacity being limited to 2GB files sizes and difficult maintainability due to multiple file formats enclosed to have full functionality, which takes us to the next alternative GeoJson.

### 2.2.4. Disadvantages of Shape files

1. This format was invented and intended in the scope of proprietary software usage.
2. It has multiple files for single functionality which requires high maintenance to avoid corrupting of files.
3. The file size is limited to 2 GigaBytes which makes it to have limited information in a single file.
4. The actual geometry is in a different file and its attributes or properties are stored in a different file so consuming more memory.

## 2.3. GeoJson

GeoJson is an open standard format for representing Geographical Features along with non-spatial attributes. GeoJson is based on JSON format a widely used format across the



internet for web applications. It is developed and maintained by a group of Developers. GeoJson is a single file which contains the location data represented in different shapes such as Point, line, Polygon, Multi polygon, MultiLine String. Unlike counter part formats, the information is coordinates centric, the main information is directly shown on the first level of object as coordinates and other information is stored in properties associated with the coordinates. Below code snippet shows an example of geoJson format.

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "geometry": {
        "type": "Point",
        "coordinates": [102.0, 0.5]
      },
      "properties": {
        "prop0": "value0"
      }
    },
    {
      "type": "Feature",
      "geometry": {
        "type": "LineString",
        "coordinates": [
          [102.0, 0.0], [103.0, 1.0], [104.0, 0.0], [105.0, 1.0]
        ]
      },
      "properties": {
        "prop0": "value0",
        "prop1": 0.0
      }
    },
    {
      "type": "Feature",
      "geometry": {
        "type": "Polygon",
        "coordinates": [
          [
            [100.0, 0.0], [101.0, 0.0], [101.0, 1.0],
            [100.0, 1.0], [100.0, 0.0]
          ]
        ]
      },
      "properties": {
        "prop0": "value0",
        "prop1": { "this": "that" }
      }
    }
  ]
}
```

In the above code snippet you can see that, there are different Geometric features in a single file with their associated attributes stored as properties. The points are stored in [x,y], [x,y,z] or [longitude, latitude] formats if using longitude and latitude they must be in decimal degrees. Main advantage of GeoJson is that we can enclose multiple geometric

shapes as a geometric collection.

### **2.3.1. Disadvantages of GeoJson**

1. It is developed by group of developers unlike counter formats which have been developed and maintained by formal organisations.
2. It does not support Topology of the geography it is representing.
3. If the file size increases over 10 Giga Bytes the loading of file into memory takes longer times to load and some times also causing part of the file to corrupt.
4. Loading time is bigger when compared to other formats for the same amount of data.

In the Next chapter you will see the proposal of a Data format GeoJson, its advantages over other formats.

## CHAPTER 3. APPROACH AND SOLUTION

Our goal is to provide information available from AIP documents in a digital format easily accessible across multiple platforms be it web applications or mobile applications. To achieve that we need to find a data structure which is cross platform friendly, In search of it we found JSON format may be a closely flexible format suitable for us but we have to define a data structure that is schema for the json files we generate, but an already modified version or schema defined json objects known as GeoJson specially designed keeping in mind for use cases where geographic location information is centric. GeoJson has its own schema to represent geographic information using different objects conforming to World Geodetic System 1984, and units of decimal degrees.

The main objects we are interested in are FeatureCollection, Features, Point. Each of the objects mentioned previously combined can used to represent the each geographic point in decimal degrees with additional properties and allowing us to add more information and being valid under the schema of GeoJson.

```
{
  "features": [
    {
      "geometry": {
        "coordinates": [-8.385761111111111, 43.29169722222222],
        "type": "Point"
      },
      "properties": {
        "ClearWay": { "length": "300 ", "width": " 150" },
        "Dimensions": { "length": "2188 ", "width": " 45" },
        "Direction": { "Geographic": "030.94", "Magnetic": " 033 " },
        "OFZ": "No",
        "RESA": "240 x 150 ",
        "RWY": "03",
        "RWY/SWY SFC PCN": "RWY: ASPH PCN 39/F/B/W/T SWY: No",
        "StopWay": "No",
        "Strip": { "length": "2308 ", "width": " 150" },
        "Threshold & Touchdown Elevation": "THR: 100 m / 329 ft"
      },
      "type": "Feature"
    },
  ],
  "type": "FeatureCollection"
}
```

For example the above code snippet shows how the threshold of La coruña airport is represented using objects Point, Features and FeatureCollection.

### 3.1. GeoJson over Other Formats

#### 3.1.1. GeoJson advantages

1. GeoJson is an open source format and has cross platform support and flexible.
2. It is easy to add additional information to the exiting geographic points as properties.

3. It allows to the information represented to be used across various Geographic information services like QGIS and others.
4. The data conversion from other formats such as shape files is possible.
5. Has a large community to have support in different issues.

### 3.1.2. Other Formats Disadvantages

1. The counter part formats are not fully open source which makes it to depend on the proprietary softwares such as ArcGIS to use the files or worse the data itself will not be available for public like AIXM.
2. The additional information addition requires additional hierarchy or a separate file itself is required to show them.
3. Limited size availability in case of shape files which is a maximum of 2Giga Bytes.
4. An additional computing power is always required when trying to load the files or worse a designated software has to be developed to just view files which is the case of AIXM format.

## 3.2. Tools of choice

To be able to gather and digitalize the aerodrome data different tools are needed.

First, to collect the data it is needed a tool to extract tables from PDF's, for that reason we use Python and its open-source libraries in order to achieve our goal of digitizing the aerodrome data. To extract information from PDF for runways and terminal stands information and further process it according to our needs, the open-source library known as 'Camelot' is used to extract the data from tables. Below we can see an example, extracted from the official Camelot webpage, of how it extracts the data from PDF's tables:

```
import camelot

tables = camelot.read_pdf('foo.pdf')

tables
<TableList n=1>

tables.export('foo.csv', f='csv', compress=True) # json, excel, html,
                                                markdown, sqlite

tables[0]
<Table shape=(7, 7)>

tables[0].parsing_report
{
  'accuracy': 99.02,
  'whitespace': 12.24,
  'order': 1,
  'page': 1
}
```

```
}  
  
tables[0].to_csv('foo.csv') # to_json, to_excel, to_html, to_markdown,  
                           to_sqlite  
  
tables[0].df # get a pandas DataFrame!
```

There are several table extractions libraries and tools that might work better in some aspects than Camelot in a qualitative analysis but on a general overview, Camelot has a determined down steps with which makes up for them using one or more of the configuration parameters (See in Appendix the comparison with other PDF table extraction libraries and tools)

The main reasons why Camelot was selected as a tool is because it gives you control over the table extraction process with tweakable settings, the ones used are stream and lattice methods which will be explained later. Another aspect to highlight is regarding the metrics as it can discard bad tables based on metrics like accuracy and whitespace, without having to manually look at each table, automatizing the data selection without need to be double checking the extraction. Camelot offers user configurability so that user can tweak the settings of data extraction and concentrate on extracting the data of which is primary interest. Finally, internally Camelot uses Pandas to covert the extracted data into data frames, which seamlessly integrates into ETL and data analysis workflows, nevertheless it can also export tables to multiple formats, which include CSV, JSON, Excel, HTML, Markdown, and Sqlite.

It is important to bear in mind that Camelot only works with text-based PDFs and not scanned documents that is why is so important that we extract the documents from ENAIRE in the correct format.

In addition, another important tool of choice is also used GeoJson.io as the file viewer for GeoJSON files. This tool is key as it cross platform friendly, and can be displayed the GeoJSON files across any devise and this was one of the main goals. The alternatives for GeoJSON files will be shape formats and JSON formats, but they where not picked because JSON format is not centered with location coordinate information and shape format it will only have the shape of the geometry and the other information cannot be added, for instance the characteristics of runways or stop clear way.

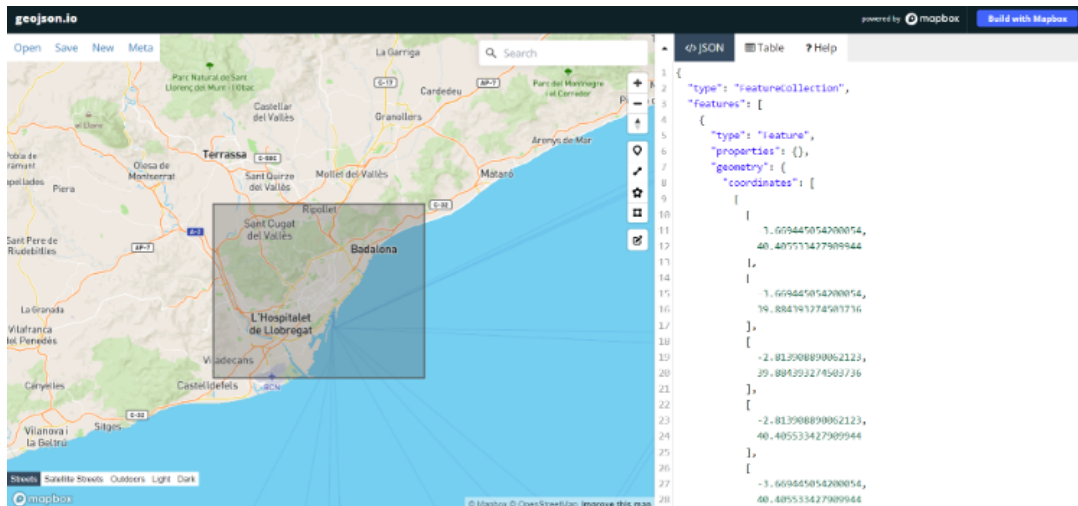


Figure 3.1: geojson default view

As mentioned, there are two methods in which Camelot extracts data. Nevertheless, the main approach is to determine the cells of the table and their boundaries so that each cell can then be converted to a format the user chooses to.

### 3.2.1. Stream Method

Let's analyze in detail each method to better understand the advantage configuration of stream and lattice, the name of these two methods come from tableau. Stream method is more of a guess work, it can be used to parse tables that have whitespaces between cells to simulate a table structure. It is built on top of PDFMiner's functionality of grouping characters on a page into words and sentences, using margins. The edges of the words are first determined to form a guess of text row and then the white spaces and then a guess is made to determine number of columns depending on the number of text rows. Finally, a table is formed using the text rows and column ranges, the words found on the page are assigned to table cells. Let's analyze it step by step:

1. Words on the PDF page are grouped into text rows based on their y axis overlaps.
2. Text edges are calculated and then used to guess interesting table areas on the PDF page.
3. The number of columns inside each table area are then guessed. This is done by calculating the mode of number of words in each text row. Based on this mode, words in each text row are chosen to calculate a list of column x ranges.
4. Words that lie inside/outside the current column x ranges are then used to extend the current list of columns.
5. Finally, a table is formed using the text rows' y ranges and column x ranges and words found on the page are assigned to the table's cells based on their x and y coordinates.

### 3.2.2. Lattice Method

Lattice method is more deterministic than a guess work, the previous stream method explained. It can be used to parse tables that have demarcated lines between cells, and it can automatically parse multiple tables present on a page. It starts by converting the PDF page to an image using ghostscript, and then processes it to get horizontal and vertical line segments by applying a set of morphological transformations (erosion and dilation) using OpenCV.

In lattice method the line segments of the table are detected first and then line intersections with the individual cells of the table are determined and corresponding data of each cell is determined, by overlapping the detected line segments and “and”ing their pixel intensities. Table boundaries are computed by overlapping the detected line segments again, this time by “or”ing their pixel intensities.

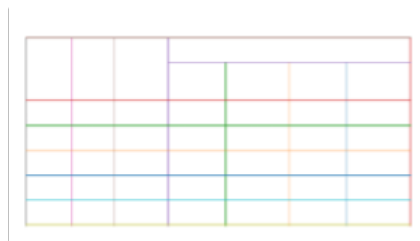


Figure 3.2: lattice method default view1

Cycle Name	KI (1/km)	Distance (mi)	Percent Fuel Savings			
			Improved Speed	Decreased Accel	Eliminate Stops	Decreased Idle
2012_2	3.30	1.3	5.9%	9.5%	29.2%	17.4%
2145_1	0.68	11.2	2.4%	0.1%	9.5%	2.7%
4234_1	0.59	58.7	8.5%	1.3%	8.5%	3.3%
2032_2	0.17	57.8	21.7%	0.3%	2.7%	1.2%
4171_1	0.07	173.9	58.1%	1.6%	2.1%	0.5%

Figure 3.3: lattice method default view2

Cycle Name	KI (1/km)	Distance (mi)	Percent Fuel Savings			
			Improved Speed	Decreased Accel	Eliminate Stops	Decreased Idle
2012_2	3.30	1.3	5.9%	9.5%	29.2%	17.4%
2145_1	0.68	11.2	2.4%	0.1%	9.5%	2.7%
4234_1	0.59	58.7	8.5%	1.3%	8.5%	3.3%
2032_2	0.17	57.8	21.7%	0.3%	2.7%	1.2%
4171_1	0.07	173.9	58.1%	1.6%	2.1%	0.5%

Figure 3.4: lattice method default view3

The previous figure can be an example of how this method first detects the lattice (corners) of each cell and boundaries and then each of the line segments of the rows and columns then the line segments are called and translated to the PDF page coordinate space to form a representation of table. Since dimensions of the PDF page and its image vary, the detected table boundaries, line intersections, and line segments are scaled and translated to the PDF page’s coordinate space, and a representation of the table is created.

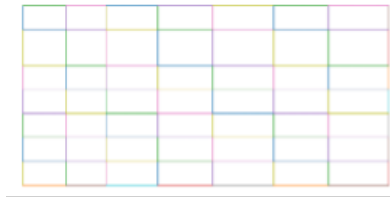


Figure 3.5: lattice method default view4

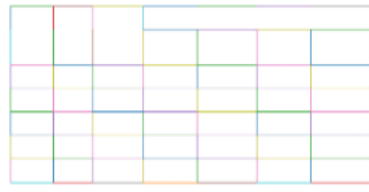


Figure 3.6: lattice method default view5

With the line segments detected individual cells can be detected which also includes spanning cells. Finally the data is assigned to the cells based on their locations (x and y coordinates) on the PDF page.

### 3.3. Converting the obstacles CSV file data to json

The Aeronautical Information Publication (AIP) is the authority of a state that publishes all the documentation containing aeronautical information of a lasting character essential to air navigation. Through the AIP we are able to extract the obstacles data, but is always provided with a CSV file, and we can directly parse the CSV files into json format using pandas library. Pandas library is a Python Package library used for working with data sets, it has functions for analyzing, cleaning, exploring, and manipulating data. Below we can see an example of the complete code to convert CSV to JSON in Python:

```
import csv
import json

# Function to convert a CSV to JSON
# Takes the file paths as arguments
def make_json(csvFilePath, jsonFilePath):

    # create a dictionary
    data = {}

    # Open a csv reader called DictReader
    with open(csvFilePath, encoding='utf-8') as csvf:
        csvReader = csv.DictReader(csvf)

        # Convert each row into a dictionary
        # and add it to data
        for rows in csvReader:
```



```

    # Assuming a column named 'No' to
    # be the primary key
    key = rows['No']
    data[key] = rows

    # Open a json writer, and use the json.dumps()
    # function to dump data
    with open(jsonFilePath, 'w', encoding='utf-8') as jsonf:
        jsonf.write(json.dumps(data, indent=4))

# Driver Code

# Decide the two file paths according to your
# computer system
csvFilePath = r'Names.csv'
jsonFilePath = r'Names.json'

# Call the make_json function
make_json(csvFilePath, jsonFilePath)

```

As we are relying in Camelot to convert the tables from PDF to Jason, we also rely on Pandas library to convert CSV files into Jason. From the CSV file it is needed to convert it to json format, because JSON is a standard text-based format for representing structured data based on JavaScript object syntax. Afterwards it is needed to convert from JSON to GeoJson to ensure the most user platform friendly for the final application of the data, selecting only the information we need and is wanted to be displayed.

Since the data is comma separated it's easy to form a structured format of the CSV data. It uses a standard method of opening the file and reading the CSV data and then assigning it to an object key which are the table headers, and it runs this logic in a for loop for the length of elements in the CSV file and creates as many objects there are in the CSV file.

### 3.4. GeoJSON package

We are using GeoJson open source package to form a GeoJson data using data extracted from the pdf files. The main components we are using from the package are Point, Feature, Feature Collection. In the below code snippet we can see the format of each component.

```

{
  "features": [
    {
      "geometry": {
        "coordinates": [2.7425, 39.56217569444444],
        "type": "Point"
      },
      "properties": {
        "ClearWay": { "length": "60 ", "width": " 150" },
        "Dimensions": { "length": "3270 ", "width": " 45" },
        "Direction": { "Geographic": "238.58", "Magnetic": " 238"
          },
      },
    },
  ],
}

```

```

        "OFZ": "No",
        "RESA": "240 x 150 ",
        "RWY": "24R ",
        "RWY/SWY SFC PCN": "RWY: ASPH PCN 80/F/A/W/T SWY: No",
        "StopWay": "No",
        "Strip": { "length": "3390 ", "width": " 300 " },
        "Threshold & Touchdown Elevation": "THR: 2.5 m / 8 ft \
                                           nTDZ: 3.2 m / 10 ft"
    },
    "type": "Feature"
},
{
    "geometry": {
        "coordinates": [2.7358083333333333, 39.543180555555555],
        "type": "Point"
    },
    "properties": {
        "ClearWay": { "length": "60 ", "width": " 150" },
        "Dimensions": { "length": "3000 ", "width": " 45" },
        "Direction": { "Geographic": "058.58", "Magnetic": " 058"
                      },
        "OFZ": "No",
        "RESA": "240 x 150 ",
        "RWY": "06R ",
        "RWY/SWY SFC PCN": "RWY: ASPH PCN 82/F/A/W/T SWY: No",
        "StopWay": "No",
        "Strip": { "length": "3120 ", "width": " 300 " },
        "Threshold & Touchdown Elevation": "THR: 8 m / 25 ft \nTDZ
                                           : No"
    },
    "type": "Feature"
},
],
"type": "FeatureCollection"
}

```

A GeoJson Object can be understood easily using Top bottom approach, The first level of the object is a type-FeatureCollection, which is a collection of Features in array with key name 'features', The next level in the object is individual Feature which contains another nested object but with a geometry

## 3.5. Complete solution architecture.

### 3.5.1. Automated

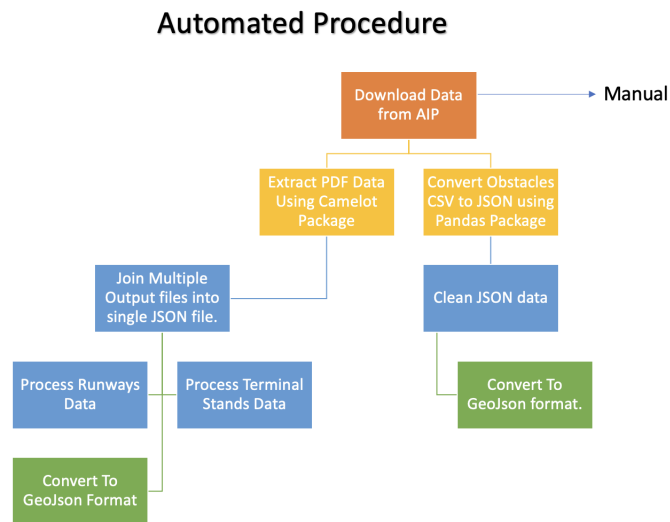


Figure 3.7: Block Diagram of Automated steps.

The above figure: 3.7 shows the steps involved in the automated procedure, however the first step is manual as the pdf files and csv files corresponding to the airports should be downloaded and fed to the code. The python program generates the geojson output files in the directory geojson which will be automatically created by the program.

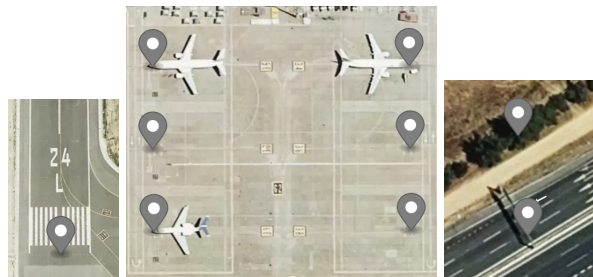


Figure 3.8: [Left]- Runways Thresholds, [Center]- Terminal Stands ,[Right]- Obstacles

The following geojson files are obtained by the automated procedure,

- (a) Runway's Threshold Points.
- (b) Terminal Stands.
- (c) Obstacles in and around Airport.

### 3.5.2. Manual actions

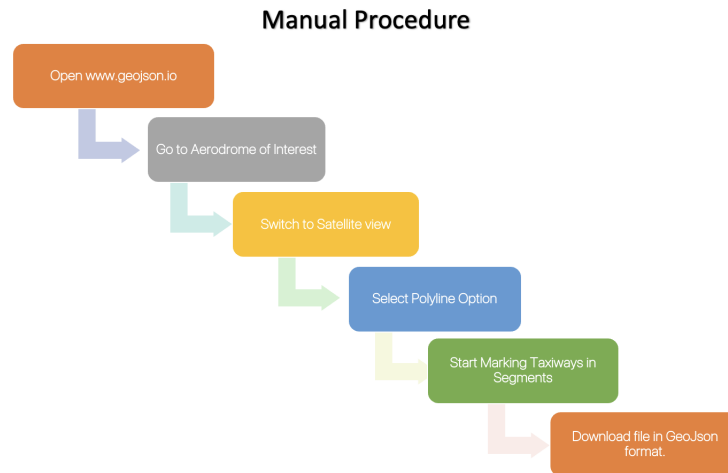


Figure 3.9: Block Diagram of Manual steps.

The above figure: 3.9 shows the steps involved in generating gejson files for taxiways and hotspots manually.



Figure 3.10: [Left]- Taxiways, [Right]- Hotspots

The following geojson files can be obtained using manual procedure.

- (a) Taxiways.
- (b) Hotspots.

## 3.6. How to use Automated Program

- (a) Download Runways information pdf and terminal stands and obstacles files and rename them by adding runways, terminalStands and obstacles to the respective existing file names like shown in figure- 3.7.
- (b) Move the renamed files to a newly created directory with the name of the airport inside data and inputData directory (data/inputData)

- (c) Now run main.py file and enter the airport name for which you want to extract the data, the output data will be generated into JSON files and GeoJSON files into json, geoJson directories respectively.
- (d) Now open geojson.io and select satellite view.

### **3.7. How to Perform Manual Actions**

- (a) Go to the destination airport you or load runways GeoJson file which will show threshold points.
- (b) Now select poly line option and mark the taxi lines starting from the runway to the terminal stands or apron intersections.
- (c) After you have your taxiways, you can delete the threshold points of the runways to have an isolated taxiways data only.
- (d) Save the file to airportName + taxiways.geojson file to the GeoJson file directory where you have rest of the GeoJson files corresponding to the airport. after finishing that we use GeoJson.io and plot taxi lines manually using satellite view.



# CHAPTER 4. SOURCECODE ARCHITECTURE

## 4.1. Runways and Terminal Stands Data Extractions

Firstly we need to ask user input for the details of the airport, Airport Name, Runway and Terminal Stands PDF page number and table numbers. we will discuss how runways and terminal stands data is extracted and later obstacles as obstacles data is extracted directly from a CSV file.

```
class AskUserInput :  
  
    def askUserAirportDetails(self):  
        airport = {}  
        airport['airportName'] = input(  
            'Enter Airport Directory Name (Case Sensitive): '  
        )  
        return airport  
  
    def askUserRunwayDetails(self, airportName):  
        airportName['runwaysTablePageNumber'] = list(  
            input('Enter the Runways table page Number/ Numbers  
                use : '))  
        airportName['runwaysTableNumber'] = list(  
            input('Enter table number / numbers in the  
                corresponding  
                Runways page : '))  
        return airportName  
  
    def askUserTerminalStandDetails(self, airport):  
        airport['terminalStandsTablePageNumber'] = list(  
            input('Enter the Terminals Stands table page Number/  
                Numbers use : '))  
        airport['terminalStandsTableNumber'] = list(input(  
            'Enter table number/ numbers in the corresponding  
                terminals Stand  
                page : '))  
        return airport
```

For that we created a class AskUserInput which will create an object which will contain all the details of the page numbers and table numbers of runways and terminal stands in their respective pdfs. This class is instantiated in the main.py file.

```
userInputs = AskUserInput()  
airportDetails = userInputs.askUserAirportDetails()  
airportDetails = userInputs.askUserRunwayDetails(airportDetails)  
airportDetails = userInputs.askUserTerminalStandDetails(  
    airportDetails)
```

With the help of another class LoadFiles, which contains basic methods such as list of files in directory and some important methods which helps to load files of the given aerodrome. One of the important methods from LoadFiles class is findAllInputFiles.

```

def findAllInputFiles(self, listOfFiles):
    files = {
        'runways': '',
        'obstacles': '',
        'terminalStands': ''
    }
    for file in listOfFiles:
        if file.endswith('runways.pdf'):
            files['runways'] = file
        if file.endswith('terminalStands.pdf'):
            files['terminalStands'] = file
        if file.endswith('obstacles.csv'):
            files['obstacles'] = file
    return files

```

loadFilesMethod it checks in the list of files in the given directory end with certain condition and add that respective file to the object. This method is instantiated in the main.py file to load all the respective files into the object named airportFiles. The both objects airportFiles and airportDetails combined are used in the further parts of program. First they are fed to executeCamelot function which extracts information from the pdf.

```

from fileSystem import LoadFiles
import camelot

def executeCamelot(airportName, pdfName, pageNumber, tableNumber):

    fileName = pdfName.split('.pdf')[0]
    inputFiles = LoadFiles()
    basePath = inputFiles.getBasePath()
    inputFilesPath = basePath+'data/'+airportName+'/inputData/'
    jsonPath = basePath+'data/'+airportName+'/json/'
    inputFiles.createDirectoryIfNotExists(jsonPath)
    allTables = []
    if len(pageNumber) == 1:
        pdfTables = camelot.read_pdf(inputFilesPath+pdfName,
                                     pageNumber[0])
        pdfTables[int(tableNumber[0])].to_json(jsonPath+fileName+
                                               pageNumber[0]+'.json')
    else:
        for n in range(len(pageNumber)):
            allTables.append(camelot.read_pdf(
                inputFilesPath+pdfName, pageNumber[n]))
            allTables[n][int(tableNumber[n])].to_json(jsonPath+
                                                       fileName+
                                                       pageNumber[n]+'.
                                                       json')

```

The executeCamelot function internally uses camelot package to read the pdfs. using the arguments pdf name, page numbers and table numbers. First it checks for the length of PageNumbers argument to decide if tables extracted is from only one page or multiple ones. if the the length is only one then it will execute camelot function only once if the length of the page numbers is higher than 1 then it iterates on the length of pageNumbers and appends the data to an array, then the data from array is written to individual json files into the json Directory. The executeCamelot



function is called in the main.py file individually for runways, terminalsStands.

```

%% Runways
executeCamelot (airportDetails ['airportName'], airportFiles ['
                                runways'],
                airportDetails ['runwaysTablePageNumber'],
                                airportDetails [
                                '
                                runwaysTableNumber
                                '])

airport = airportDetails ['airportName']
runwayFileName = files.removeFileExtension (airportFiles ['runways']
                                             , '.pdf')
allRunways = joinJsonFiles.joinRunwayJsonFiles (airportDetails,
                                                  airportFiles)
runwaysGeoJson = cgj.convertRunwaysJsonToGeoJson (allRunways)
%% Terminals
executeCamelot (airport, airportFiles ['terminalStands'],
                airportDetails ['terminalStandsTablePageNumber'],
                                airportDetails [
                                '
                                terminalStandsTableNumber
                                '])

terminalStandsFileName = files.removeFileExtension (
    airportFiles ['terminalStands'], '.pdf')
allTerminalStands = joinJsonFiles.joinTerminalJsonFiles (
    airportDetails, airportFiles)
terminalStandsGeoJson = cgj.convertTerminalStandsJsonToGeojson (
    allTerminalStands)

```

after calling executeCamelot function the results data is passed on to the joinRunwayJsonFiles or joinTerminalJsonFiles functions.

```

import readJsonFiles

def joinRunwayJsonFiles (airportDetails, airportFiles):
    allRunways = []
    if (len(airportDetails ['runwaysTablePageNumber']) > 1):
        runwayFiles = readJsonFiles.getRunwayJsonFiles (
            airportDetails, airportFiles)
        for n in range(0, len(runwayFiles)):
            allRunways.extend(runwayFiles [n])
        return allRunways
    else:
        return (readJsonFiles.getRunwayJsonFiles (airportDetails,
                                                    airportFiles))

def joinTerminalJsonFiles (airportDetails, airportFiles):
    allTerminalStands = []
    if (len(airportDetails ['terminalStandsTablePageNumber']) > 1):
        terminalStandsFiles = readJsonFiles.getTerminalStandsFiles
            (
                airportDetails, airportFiles)
        for n in range(0, len(terminalStandsFiles)):
            allTerminalStands.extend(terminalStandsFiles [n])
        return allTerminalStands
    else:

```

```

return (readJsonFiles.getTerminalStandsFiles (
                                airportDetails,
                                airportFiles))

```

The functions `joinRunwayJsonFiles` `joinTerminalJsonFiles` reads the generated json files from `executeCamelot` function and combines them into one object and pass it to the `convertRunwaysJsonToGeojson` and `convertTerminalStandsJsonToGeojson` functions, these functions internally use `TransformRunwayData` and `TransformTerminalData` Classes.

```

tod = TransformObstaclesData ()
trd = TransformRunwayData ()
ttd = TransformTerminalData ()
tdc = TransformDataCommon ()

def convertRunwaysJsonToGeoJson (data) :
    jsonData = trd.changeRunwayDataKeys (data)
    validJsonData = trd.removeRunwayInvalidData (jsonData)
    cleanJson = trd.cleanRunwayData (validJsonData)
    thresholds = trd.getRunwayThresholds (cleanJson)
    thresholds = tdc.convertCoordinatesToDMS (thresholds)
    features = tdc.createGeoJsonPointFeatures (thresholds)
    geojson = trd.combineRunwayPointAndProperties (features,
                                                    cleanJson)

    return geojson

def convertTerminalStandsJsonToGeojson (data) :
    jsonData = ttd.changeTerminalStandsDataKeys (data)
    validJsonData = ttd.removeTerminalStandsInvalidData (jsonData)
    cleanJson = ttd.cleanTerminalStandsData (validJsonData)
    coordinates = ttd.getCoordinates (cleanJson)
    coordinates = tdc.convertCoordinatesToDMS (coordinates)
    features = tdc.createGeoJsonPointFeatures (coordinates)
    geojson = ttd.combineTerminalStandsPointsAndProperties (
                                                    features, cleanJson)

    return geojson

```

A series of important methods are used in order to generate the Geojson data from the generated json data. such as removing the ASCII characters as camelot function reads the pdf data's special characters in ASCII code. depending on the context we might need to remove the ASCII code or include it in the data and many such exceptions are achieved using separate methods designated in each of the classes. Internally these classes use another class `ExtractUsableDataFromJson` which has the methods `removeSpanishCharacters`, `removeUniCode` and other useful ones, the most important methods in this class is `coordinates` which will extract the coordinates.

```

def coordinates (self, threshold) :
    data = self.replaceNewline (threshold, "")
    if "N" in data:
        data = data.split ('N')
        coordinates = {
            "latitude": str (data [0] + 'N'),
            "longitude": str (data [1])

```

```

    }
    return coordinates
if "S" in data:
    data = data.split('S')
    coordinates = {
        "latitude": str(data[0]+'S'),
        "longitude": str(data[1])
    }
return coordinates

```

The extracted coordinates will be returned in the form of object which has separate Longitude and Latitude Keys and value pairs. This method is very critical because later a geojson object which depends on geographic information. These extracted coordinates are then converted to Degrees Minutes and Seconds with the help of TransformCommonData class as this method is not specific to only runways and terminals it is added to the class which is common to all.

```

def convertCoordinatesToDMS(self, coordinates):
    identifiers = ['latitude', 'longitude']
    for n in coordinates.keys():
        coordinates[n] = validateCoordinates(coordinates[n])
        coordinates[n][identifiers[0]] = convertDmsToDd(
            coordinates[n][identifiers[0]])
        coordinates[n][identifiers[1]] = convertDmsToDd(
            coordinates[n][identifiers[1]])
    return coordinates

```

The converted coordinates are used to form a geojson object known as 'geometry' which is combined with another object called as 'Feature' which will have key name 'properties' where the rest of the information extracted corresponding to that geographic point is added. All the features are added to the global object 'FeaturesCollection'.

In this way all the information corresponding to the coordinates provided in the pdf's are used to form an object which can be used various GIS platforms.

```

{
  "geometry": {
    "coordinates": [2.7107275, 39.54714763888889],
    "type": "Point"
  },
  "properties": {
    "ClearWay": { "length": "60 ", "width": " 150" },
    "Dimensions": { "length": "3270 ", "width": " 45" },
    "Direction": { "Geographic": "058.56", "Magnetic": " 058"
                  },
    "OFZ": "No",
    "RESA": "240 x 150 ",
    "RWY": "06L ",
    "RWY/SWY SFC PCN": "RWY: ASPH PCN 80/F/A/W/T SWY: No",
    "StopWay": "No",
    "Strip": { "length": "3390 ", "width": " 300 " },
    "Threshold & Touchdown Elevation": "THR: 4.6 m / 15 ft \
                                         nTDZ: 4.6 m / 15 ft"
  },
  "type": "Feature"
}

```

## 4.2. Obstacles Data Extraction

Obstacles data is available in AIP in the form of CSV files, (Comma Separated Values), which can be downloaded and converted to JSON and then to GeoJSON. We used Pandas library package to convert the csv format data to json.

```
def csvToJson(self, airportName, csvName):
    basePath = self.getBasePath()
    csvPath = basePath+'data/'+airportName+'/inputData/'+
                csvName+'.csv'
    jsonPath = basePath+'data/'+airportName+'/json/'+csvName+'
                .json'
    obstaclesDataFrame = pd.read_csv(csvPath, sep=';')
    obstaclesDataFrame.to_json(jsonPath)
```

The converted JSON data is then cleaned using the TransformObstaclesData class to form a usable information to represent in GeoJSON format. Unlike Runways and TerminalStands, the number of obstacles listed in the CSV files is very large so minimum number of operations on this data would result in faster processing times.

```
class TransformObstaclesData:

    def combineObstaclesPointAndProperties(self, features,
                                          properties):
        collection = []
        for n in features.keys():
            m = int(n)
            del properties[m]['Coordinates']
            features[n]['properties'] = properties[m]
            collection.append(features[n])
        return FeatureCollection(collection)

    def getKeysAndValues(self, obstacles):
        keys = list(obstacles.keys())
        values = list(obstacles.values())
        return keys, values

    def getEnglishKeys(self, originalKeys):
        englishKeys = []
        for n in range(len(originalKeys)):
            if '_' in originalKeys[n]:
                englishKeys.append(originalKeys[n].split('_')[1])
            else:
                englishKeys.append(originalKeys[n])
        return englishKeys

    def mergeNewKeys(self, newKeys, obstaclesValues):
        newObstaclesData = {}
        for n in range(len(obstaclesValues)):
            newObstaclesData[newKeys[n]] = obstaclesValues[n]
        return newObstaclesData

    def combineCoordinates(self, latitudes, longitudes):
        coordinates = {}
        allCoordinates = {}
        listLength = (len(latitudes))
        if (len(latitudes) == len(longitudes)):
```

```
        for n in range(listLength):
            coordinates[n] = {'latitude': latitudes[n],
                              'longitude': longitudes[n]}
            allCoordinates[str(n)] = coordinates[n]
    else:
        print('Mismatched Array Lengths :', 'lat:', len(
            latitudes), ',', 'long:', len(longitudes))
    return allCoordinates

def addNewCoordinatesToObstacles(self, obstacles, coordinates)
    :
    obstacles['Coordinates'] = coordinates
    del obstacles['Longitude']
    del obstacles['Latitude']
    return obstacles

def convertCoordinatesToDms(self, coordinates):
    convertedCoordinates = []
    convertedCoordinates.append(tdc.convertCoordinatesToDMS(
        coordinates))

    return convertedCoordinates

def mergeObstacleData(self, obstacles):
    allObstacles = []
    categories = list(obstacles.keys())
    length = len(list(obstacles['Type'].values()))
    for m in range(length):
        mergedObstacles = {}
        for n in categories:
            mergedObstacles[n] = obstacles[n][str(m)]
        allObstacles.append(mergedObstacles)
    return allObstacles
```

## 4.3. Files Organisation

### 4.3.1. Source Code files

The source code files are organised in a way that the source code is separate from all the other data files and output files. below you can see the file structure of the source code. Main.py is the files which needs to be executed which depends on packages, Camelot, Pandas, json and fs(fileSystem).

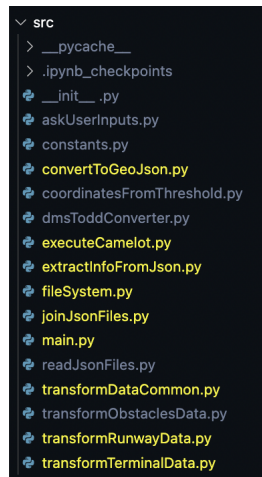


Figure 4.1: Source Code Files Sturcture

### 4.3.2. Data files

The data files are organised according to the name of the aerodromes, each aerodrome directory consists of a compulsory inputFiles directory which has pdf files of runways, terminal stands and obstacles CSV file. The naming of the files generated by the program follows the same names but adds few differences such as adding page numbers at the end and adding if it is a runway or terminal stands file. The other directories are generated by the program itself namely json and geojson directories.

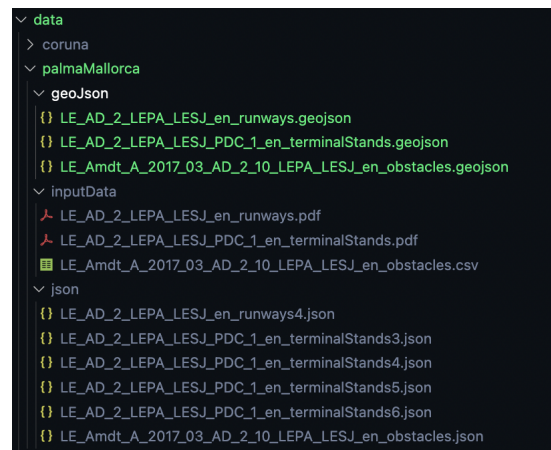


Figure 4.2: Data Files Sturcture

The geoJson directory will have only three files generated by the program, the file names end with runways, terminalsStands, obstacles with the extension of '.geojson' the manually generated files, have the same extension but the names end with taxiways and centreline. The json directory will have multiple files with similar names but ends with the page number of the pdf from which the data is extracted.

# CHAPTER 5. OUTPUT

## 5.1. Mallorca Airport

Palma de Mallorca Airport also known as Son Sant Joan Airport is an international airport located on the Island of Palma de Mallorca on the East of Spain. At peak this airport handles nearly 30 million Passengers making it the 3rd busiest airport in Spain after Barcelona and Madrid Airports. This Airport underwent great transformations from 1920's starting as an Airmail Line for Spanish Government and then in 1960's The international airport was created as the traffic was massively increased due to the tourists.

### 5.1.1. Program Generated Output

#### 5.1.1.1. Runways

The extracted data from runways pdf saved in runways.geojson file. we are using geojson.io to show the generated data instead developing a new frontend. Follow the steps below to use geojson.io.

- (a) Goto [www.geojson.io](http://www.geojson.io)
- (b) click on Open and then File.
- (c) Use the Program Generated GeoJson file,



Figure 5.1: Mallorca Airport Runways Thresholds

```
{
  "features": [
    {
      "geometry": {
        "coordinates": [2.7107275, 39.54714763888889],

```





```

    "type": "Feature"
  },
  {
    "geometry": {
      "coordinates": [2.761525694444444, 39.555338888888888],
      "type": "Point"
    },
    "properties": {
      "ClearWay": null,
      "Dimensions": { "length": "3000 ", "width": " 45 " },
      "Direction": { "Geographic": "238.60", "Magnetic": " 238"
                    },
      "OFZ": "Yes",
      "RESA": "240 x 150 ",
      "RWY": "24L ",
      "RWY/SWY SFC PCN": "RWY: ASPH  PCN 82/F/A/W/T  SWY: No",
      "StopWay": "No",
      "Strip": { "length": "3120 ", "width": " 300 " },
      "Threshold & Touchdown Elevation": "THR: 2.4 m / 8 ft \
                                          nTDZ: 4.2 m / 14 ft"
    }
  },
  "type": "Feature"
}
],
"type": "FeatureCollection"
}

```

Figure: 5.1 shows the threshold points and the above code snippet is the geojson data for the threshold, The data contains the geographic location and other properties such as Dimensions, Directions and if the corresponding runways have stopways and its touch down elevations.

#### 5.1.1.2. Terminal Stands



Figure 5.2: Mallorca Airport Terminal Stands

```
{
  "features": [
    {
      "geometry": {
        "coordinates": [2.722633333333333, 39.548019444444444],
        "type": "Point"
      },
      "properties": {
        "Exit": "R",
        "Max ACFT": "B753",
        "Nose To": "E",
        "Ramp": "R3",
        "Remarks": "INCOMP. 02B",
        "Stand": "02"
      },
      "type": "Feature"
    },
    {
      "geometry": {
        "coordinates": [2.722625, 39.548355555555555],
        "type": "Point"
      },
      "properties": {
        "Exit": "A",
        "Max ACFT": "20m (4)",
        "Nose To": "\u2013",
        "Ramp": "R3",
        "Remarks": "INCOMP. 02",
        "Stand": "02B"
      },
      "type": "Feature"
    },
    {
      "geometry": {
        "coordinates": [2.723391666666667, 39.54835],
        "type": "Point"
      },
      "properties": {
        "Exit": "R",
        "Max ACFT": "A321",
        "Nose To": "E",
        "Ramp": "R3",
        "Remarks": "",
        "Stand": "03"
      },
      "type": "Feature"
    },
    {
      "geometry": {
        "coordinates": [2.723791666666667, 39.548541666666665],
        "type": "Point"
      },
      "properties": {
        "Exit": "R",
        "Max ACFT": "A321",
        "Nose To": "E",
        "Ramp": "R3",
      }
    }
  ]
}
```

```
    "Remarks": "",
    "Stand": "04"
  },
  "type": "Feature"
},
{
  "geometry": {
    "coordinates": [2.7239916666666667, 39.549],
    "type": "Point"
  },
  "properties": {
    "Exit": "A",
    "Max ACFT": "30m (4)",
    "Nose To": "\u2013",
    "Ramp": "R3",
    "Remarks": "",
    "Stand": "05"
  },
  "type": "Feature"
},
{
  "geometry": {
    "coordinates": [2.7252361111111111, 39.549288888888889],
    "type": "Point"
  },
  "properties": {
    "Exit": "R",
    "Max ACFT": "B738",
    "Nose To": "E",
    "Ramp": "R3",
    "Remarks": "\u2013",
    "Stand": "06"
  },
  "type": "Feature"
},
{
  "geometry": {
    "coordinates": [2.726088888888889, 39.549580555555555],
    "type": "Point"
  },
  "properties": {
    "Exit": "R",
    "Max ACFT": "B753",
    "Nose To": "E",
    "Ramp": "R4",
    "Remarks": "400 Hz - A/C (5)",
    "Stand": "08"
  },
  "type": "Feature"
},
{
  "geometry": {
    "coordinates": [2.726775, 39.549727777777775],
    "type": "Point"
  },
  "properties": {
    "Exit": "R",
    "Max ACFT": "B738",
```

```

      "Nose To": "E",
      "Ramp": "R4",
      "Remarks": "400 Hz - A/C (5)",
      "Stand": "10"
    },
    "type": "Feature"
  },
  ],
  "type": "FeatureCollection"
}

```

The above figure - 5.2 is the example of TerminalStands data generated from the program. To view the terminal stands data it is recommended to use satellite view as other variants may look like the points are out of place. The above code snippet contains a part of the output generated as there are more 100 terminals stands in Palma Mallorca airport.

### 5.1.1.3. Obstacles



Figure 5.3: Mallorca Airport Obstacles

The above figure - 5.3 shows the obstacles, as you can see in the right picture the buildings which are taller are marked as obstacles, it is recommended to zoom in and out on the maps as usually there are many obstacles listed in the original CSV files.

### 5.1.2. Manually Generated Output

We collect the points of Taxiways as the information from AIP is not available about the exact taxiways on the runways. So we use geojson.io's mapbox tool to add Linestrings along the length of taxiways, A most suitable way to collect the taxi ways is by making segments like if the taxiway splits into two, having a segment of two individual splits is better than having to repeat the entire taxiway. In this way we reduce the data redundancy and overall size of the GeoJson file.

### 5.1.2.1. Taxiways

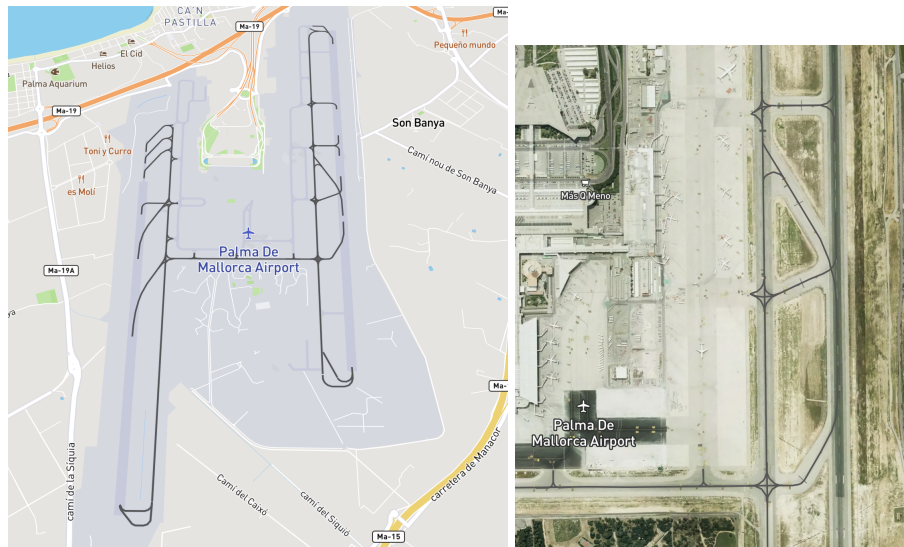


Figure 5.4: Mallorca Airport Taxiways

The above figure- 5.4 represents the manually extracted taxiways, It is recommended to use satellite view while collecting the points of taxiways and to view mapbox view is the best. This manually gathered information is an addition to the program generated information and with more clear information from AIP can help us to automate this data gathering as well.

### 5.1.2.2. Hotspots



Figure 5.5: Mallorca Airport Hotspots

Depending upon the traffic the Airport deals with a region or regions in the airport are designated as hotspots as it is essential to minimise the unwanted and unauthorised activities in that zone. The Airport of Palma Mallorca has 4 hotspots which are shown in the figure : 5.5

## 5.2. Coruña Airport

Coruña Airport previously known as Alvedro Airport is located in northwest of Spain, in 2021 this airport served nearly 600,000 passengers. This airport construction started in 1953, it is not a typical busy airport but its average annual yearly passengers are a little over 1 million. It consists two runways and corresponding taxiways, heliways and corresponding heliways connecting them.

### 5.2.1. Program Generated Output

#### 5.2.1.1. Runways

Coruña has two runways The below image shows the Program generated output of the threshold locations,



Figure 5.6: Coruna Airport Runways

The code snippet below shows the Geojson object,

```
{
  "features": [
    {
      "geometry": {
        "coordinates": [-8.385761111111111, 43.29169722222222],
        "type": "Point"
      },
      "properties": {
        "ClearWay": { "length": "300 ", "width": " 150" },
        "Dimensions": { "length": "2188 ", "width": " 45" },
        "Direction": { "Geographic": "030.94", "Magnetic": " 033" },
        "OFZ": "No",
        "RESA": "240 x 150 ",
        "RWY": "03",
        "RWY/SWY SFC PCN": "RWY: ASPH PCN 39/F/B/W/T SWY: No",
        "StopWay": "No",
        "Strip": { "length": "2308 ", "width": " 150" },

```

```

    "Threshold & Touchdown Elevation": "THR: 100 m / 329 ft"
  },
  "type": "Feature"
},
{
  "geometry": {
    "coordinates": [-8.3719, 43.30858611111111],
    "type": "Point"
  },
  "properties": {
    "ClearWay": { "length": "300 ", "width": " 150 " },
    "Dimensions": { "length": "2188 ", "width": " 45 " },
    "Direction": { "Geographic": "210.95", "Magnetic": " 213"
                  },
    "OFZ": "Yes",
    "RESA": "240 x 150",
    "RWY": "21 ",
    "RWY/SWY SFC PCN": "RWY: ASPH  PCN 39/F/B/W/T  SWY: No",
    "StopWay": "No",
    "Strip": { "length": "2248 ", "width": " 150 " },
    "Threshold & Touchdown Elevation": "THR: 82.8 m / 272 ft \
                                         nTDZ: 85.6 m / 281 ft"
  },
  "type": "Feature"
}
],
"type": "FeatureCollection"
}

```

### 5.2.1.2. Terminal Stands

Coruna has approximately 16 terminal stands serving the airport for few airlines such as Air Europa, Iberia and Vueling namely, below code snippet shows a few of them in geojson format.



Figure 5.7: Coruna Airport Terminalstands

```
{
```

```
"features": [  
  {  
    "geometry": {  
      "coordinates": [-8.381672222222223, 43.301116666666665],  
      "type": "Point"  
    },  
    "properties": {  
      "Exit": "A",  
      "Max ACFT": "A321",  
      "Nose To": "\u2013",  
      "Ramp": "12",  
      "Remarks": "INCOMP. 11A, H1, H2",  
      "Stand": "1"  
    },  
    "type": "Feature"  
  },  
  {  
    "geometry": {  
      "coordinates": [-8.382516666666668, 43.301224999999995],  
      "type": "Point"  
    },  
    "properties": {  
      "Exit": "A",  
      "Max ACFT": "A321",  
      "Nose To": "\u2013",  
      "Ramp": "13",  
      "Remarks": "\u2013",  
      "Stand": "1"  
    },  
    "type": "Feature"  
  },  
  {  
    "geometry": {  
      "coordinates": [-8.381922222222222, 43.301188888888889],  
      "type": "Point"  
    },  
    "properties": {  
      "Exit": "A",  
      "Max ACFT": "AS55",  
      "Nose To": "\u2013",  
      "Ramp": "H1",  
      "Remarks": "INCOMP. 11A, 12",  
      "Stand": "1"  
    },  
    "type": "Feature"  
  },  
  {  
    "geometry": {  
      "coordinates": [-8.381719444444444, 43.300963888888889],  
      "type": "Point"  
    },  
    "properties": {  
      "Exit": "A",  
      "Max ACFT": "S61",  
      "Nose To": "\u2013",  
      "Ramp": "H2",  
      "Remarks": "INCOMP. 11A, 12",  
      "Stand": "1"  
    }  
  }  
]
```



```

    },
    "type": "Feature"
  },
  {
    "geometry": {
      "coordinates": [-8.37906388888889, 43.30328055555555],
      "type": "Point"
    },
    "properties": {
      "Exit": "A",
      "Max ACFT": "A139",
      "Nose To": "\u2013",
      "Ramp": "H3",
      "Remarks": "INCOMP. 6",
      "Stand": "0"
    }
  },
  "type": "Feature"
},
"FeatureCollection"
}

```

### 5.2.1.3. Obstacles

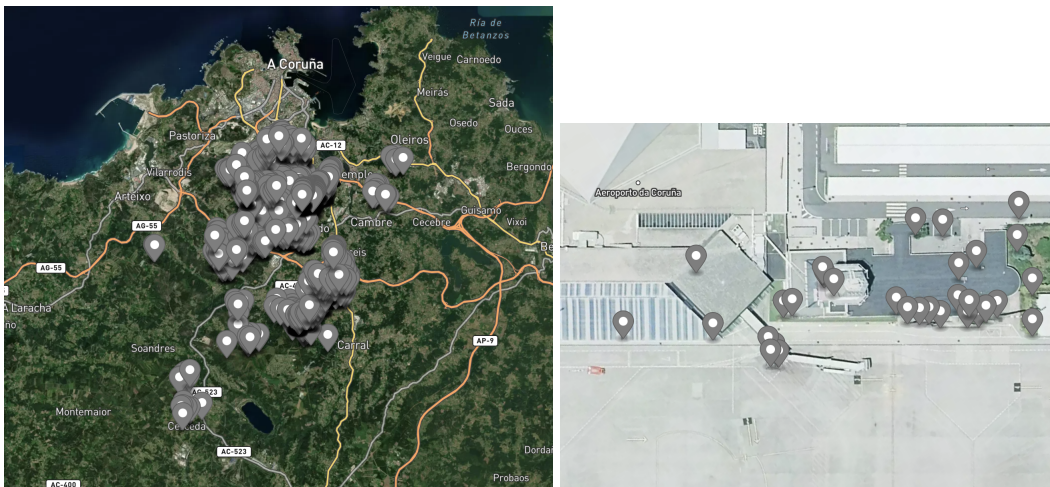


Figure 5.8: Coruna Airport Obstacles

```

{
  "features": [
    {
      "geometry": {
        "coordinates": [-8.436890027777778, 43.24280588888889],
        "type": "Point"
      },
      "properties": {
        "AMDT": "AIRAC - 03/20",
        "Elevation": 432.275,
        "Frangibility": "No",
        "GFID": "C768699B-D156-4CF6-976F-5AA9F4D6F95A",

```

```

    "GeometryType": "Puntual/Point",
    "Height": 0.0,
    "HorzAccuracy": 0.2,
    "HorzConfLevel": 95,
    "HorzRefSyst": "WGE",
    "HorzResolution": 0.001,
    "Identifier": "LECO-OBS-00193-001-2015",
    "Integrity": "Si/Yes",
    "IntegrityRemark": null,
    "Lighting": "No",
    "Marking": "No",
    "MeasurementDate": 20150211,
    "MeasurementType": "Radiaci\u00f3n/Radiation",
    "OriginatorId": "AENA",
    "PUB": "2020-03-12",
    "SourceId": "CONSULTOP",
    "Type": "Cota/Natural Highpoint",
    "UOM": "m",
    "VertAccuracy": 0.2,
    "VertConfLevel": 95,
    "VertRefSyst": "MSL",
    "VertResolution": 0.001,
    "WEF": "2020-04-23"
  },
  "type": "Feature"
},
{
  "geometry": {
    "coordinates": [-8.435313694444444, 43.240531527777776],
    "type": "Point"
  },
  "properties": {
    "AMDT": "AIRAC - 03/20",
    "Elevation": 426.462,
    "Frangibility": "No",
    "GFID": "68E40D47-1878-4ACF-AD5B-2ADB0195A9D9",
    "GeometryType": "Superficial/Polygon",
    "Height": 20.976,
    "HorzAccuracy": 0.2,
    "HorzConfLevel": 95,
    "HorzRefSyst": "WGE",
    "HorzResolution": 0.001,
    "Identifier": "LECO-OBS-00190-001-2015",
    "Integrity": "Si/Yes",
    "IntegrityRemark": null,
    "Lighting": "No",
    "Marking": "No",
    "MeasurementDate": 20150211,
    "MeasurementType": "Radiaci\u00f3n/Radiation",
    "OriginatorId": "AENA",
    "PUB": "2020-03-12",
    "SourceId": "CONSULTOP",
    "Type": "Vegetaci\u00f3n/Vegetation",
    "UOM": "m",
    "VertAccuracy": 0.2,
    "VertConfLevel": 95,
    "VertRefSyst": "MSL",
    "VertResolution": 0.001,

```

```

        "WEF": "2020-04-23"
    },
    "type": "Feature"
},
{
    "geometry": {
        "coordinates": [-8.437487166666667, 43.24416572222222],
        "type": "Point"
    },
    "properties": {
        "AMDT": "AIRAC - 03/20",
        "Elevation": 426.025,
        "Frangibility": "No",
        "GFID": "8DAA2580-6094-43EF-A01A-77084BA1FBB2",
        "GeometryType": "Puntual/Point",
        "Height": 0.0,
        "HorzAccuracy": 0.2,
        "HorzConfLevel": 95,
        "HorzRefSyst": "WGE",
        "HorzResolution": 0.001,
        "Identifier": "LECO-OBS-00191-001-2015",
        "Integrity": "Si/Yes",
        "IntegrityRemark": null,
        "Lighting": "No",
        "Marking": "No",
        "MeasurementDate": 20150211,
        "MeasurementType": "Radiaci\u00f3n/Radiation",
        "OriginatorId": "AENA",
        "PUB": "2020-03-12",
        "SourceId": "CONSULTOP",
        "Type": "Cota/Natural Highpoint",
        "UOM": "m",
        "VertAccuracy": 0.2,
        "VertConfLevel": 95,
        "VertRefSyst": "MSL",
        "VertResolution": 0.001,
        "WEF": "2020-04-23"
    },
    "type": "Feature"
},
],
"type": "FeatureCollection"
}

```

The above figures: 5.8 show the obstacles which were generated by the automated program, the image on the right is an example how the obstacles are reported inside the airport itself, the airport buildings and Traffic Control tower as obstacles.

### 5.2.2. Manual Generated Output

The following data has been secured manual procedure as explained previously and there are no Hotspots for this airport.

### 5.2.2.1. Taxiways



Figure 5.9: Coruna Airport Taxiways

## 5.3. Seville Airport

Seville Airport also known as San Pablo Airport, is the 6th busiest airport located in south of Spain. It has a capacity to handle 6 million passengers per year and is undergoing a construction in the vision of expanding to accommodate 10 million passengers per year. The main operations of the airport began in the year 1919 when the municipal government of Seville handed a plot of land to Military and Aeronautical Society. Presently the airport has combined operations for commercial airlines and as base for defence operations.

### 5.3.1. Program Generated Output

#### 5.3.1.1. Runways

Seville airport has two runways which are shared by commercial purposes and defence purposes as well.



Figure 5.10: Seville Airport Runways

```

{
  "features": [
    {
      "geometry": {
        "coordinates": [-5.912083333333333, 37.417874999999995],
        "type": "Point"
      },
      "properties": {
        "ClearWay": { "length": "60 ", "width": " 150" },
        "Dimensions": { "length": "3364 ", "width": " 45" },
        "Direction": { "Geographic": "089.74 ", "Magnetic": " 091
          " },
        "OFZ": "No",
        "RESA": "90 x 150",
        "RWY": "09",
        "RWY/SWY SFC PCN": "RWY: ASPH PCN 82 F/D/W/T SWY: No",
        "StopWay": "No",
        "Strip": { "length": "3484 ", "width": " 300" },
        "Threshold & Touchdown Elevation": "THR: 25.3 m / 83 ft \
          nTDZ: 25.9 m / 85 ft"
      },
      "type": "Feature"
    },
    {
      "geometry": {
        "coordinates": [-5.874102777777778, 37.41800277777777],
        "type": "Point"
      },
      "properties": {
        "ClearWay": { "length": "60 ", "width": " 150 " },
        "Dimensions": { "length": "3364 ", "width": " 45" },
        "Direction": { "Geographic": "269.77 ", "Magnetic": " 271
          " },
        "OFZ": "No",
        "RESA": "240 x 150",
        "RWY": "27",
        "RWY/SWY SFC PCN": "RWY: ASPH PCN 82 F/D/W/T SWY: No",
        "StopWay": "No",
        "Strip": { "length": "3484 ", "width": " 300" },

```

```

      "Threshold & Touchdown Elevation": "THR: 33.8 m / 111 ft \
                                         nTDZ: 33.8 m / 111 ft"
    },
    "type": "Feature"
  }
],
"type": "FeatureCollection"
}

```

the above code snippet shows the program generated runways geojson object and the plotting the same on geojson.io is shown in [5.10](#)

### 5.3.1.2. Terminal Stands

Seville Airport has 61 terminals stands, in which 23 stands are self manoeuvring.

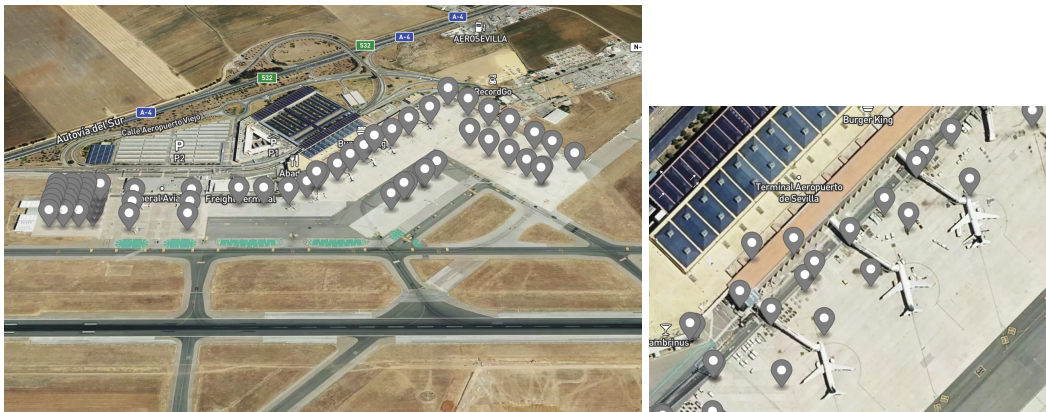


Figure 5.11: Seville Airport Terminal Stands

```

{
  "features": [
    {
      "geometry": {
        "coordinates": [-5.900663888888889, 37.42133611111111],
        "type": "Point"
      },
      "properties": {
        "Exit": "R",
        "Max ACFT": "B739/A321",
        "Nose To": "\u2013",
        "Ramp": "R-4",
        "Remarks": "INCOMP. 30",
        "Stand": "01"
      },
      "type": "Feature"
    },
    {
      "geometry": {
        "coordinates": [-5.900386111111112, 37.42161111111111],
        "type": "Point"
      },
      "properties": {

```

```
    "Exit": "R",
    "Max ACFT": "B739/A321",
    "Nose To": "\u2013",
    "Ramp": "R-4",
    "Remarks": "INCOMP. 30",
    "Stand": "02"
  },
  "type": "Feature"
},
{
  "geometry": {
    "coordinates": [-5.899908333333332, 37.421908333333334],
    "type": "Point"
  },
  "properties": {
    "Exit": "R",
    "Max ACFT": "B739/A321",
    "Nose To": "\u2013",
    "Ramp": "R-4",
    "Remarks": "\u2013",
    "Stand": "03"
  },
  "type": "Feature"
},
{
  "geometry": {
    "coordinates": [-5.899586111111111, 37.42222222222222],
    "type": "Point"
  },
  "properties": {
    "Exit": "R",
    "Max ACFT": "B739/A321",
    "Nose To": "\u2013",
    "Ramp": "R-4",
    "Remarks": "\u2013",
    "Stand": "04"
  },
  "type": "Feature"
},
{
  "geometry": {
    "coordinates": [-5.899233333333333, 37.422516666666667],
    "type": "Point"
  },
  "properties": {
    "Exit": "R",
    "Max ACFT": "B739/A321",
    "Nose To": "\u2013",
    "Ramp": "R-4",
    "Remarks": "\u2013",
    "Stand": "05"
  },
  "type": "Feature"
},
{
  "geometry": {
    "coordinates": [-5.898941666666667, 37.422855555555555],
    "type": "Point"
  },
```

```

    },
    "properties": {
      "Exit": "A/R",
      "Max ACFT": "B739/A321",
      "Nose To": "\u2013",
      "Ramp": "R-4",
      "Remarks": "\u2013",
      "Stand": "06"
    },
    "type": "Feature"
  },
  ],
  "type": "FeatureCollection"
}

```

The above code snippet shows some of the terminal stands out of 61 terminal stands in geojson format, the figure:5.11 shows the same on geojson.io

### 5.3.1.3. Obstacles

A total number of 309 obstacles data had been extracted from the csv files provided by ENAIRE's AIP service.



Figure 5.12: Seville Airport Obstacles

```

{
  "features": [
    {
      "geometry": {
        "coordinates": [-5.907265111111111, 37.420270861111106],
        "type": "Point"
      },
      "properties": {
        "ACCION": "INSERT",
        "AMDT": "AIRAC - 07/22",
        "Elevation": 26.742,
        "Frangibility": "No",
        "GFID": "847082BB-2447-41FD-8BB9-30F8FBC0842C",
        "GeometryType": "Puntual/Point",
        "Height": 1.968,

```



```
    "HorzAccuracy": 0.2,
    "HorzConfLevel": 95,
    "HorzRefSyst": "WGE",
    "HorzResolution": 0.001,
    "Identifier": "LEZL-OBS-00019-000-2018",
    "Integrity": "Si/Yes",
    "IntegrityRemark": null,
    "Lighting": "No",
    "Marking": "No",
    "MeasurementDate": 20181009,
    "MeasurementType": "Radiaci\u00f3n/Radiation",
    "OriginatorId": "AENA",
    "PUB": "2022-05-05T00:00:00Z",
    "SourceId": "CONSULTOP",
    "Type": "Sistema El\u00e9ctrico/Electrical System",
    "UOM": "m",
    "VertAccuracy": 0.2,
    "VertConfLevel": 95,
    "VertRefSyst": "MSL",
    "VertResolution": 0.001,
    "WEF": "2022-06-16T00:00:00Z"
  },
  "type": "Feature"
},
{
  "geometry": {
    "coordinates": [-5.873295277777778, 37.41800602777778],
    "type": "Point"
  },
  "properties": {
    "ACCION": "INSERT",
    "AMDT": "AIRAC - 07/22",
    "Elevation": 34.475,
    "Frangibility": "No",
    "GFID": "B628F23B-8E9F-4B15-8A73-368AC86E60F5",
    "GeometryType": "Puntual/Point",
    "Height": 0.86,
    "HorzAccuracy": 0.006,
    "HorzConfLevel": 95,
    "HorzRefSyst": "WGE",
    "HorzResolution": 0.001,
    "Identifier": "LEZL-OBS-00033-000-2018",
    "Integrity": "Si/Yes",
    "IntegrityRemark": null,
    "Lighting": "No",
    "Marking": "Si/Yes",
    "MeasurementDate": 20181011,
    "MeasurementType": "GPS",
    "OriginatorId": "AENA",
    "PUB": "2022-05-05T00:00:00Z",
    "SourceId": "CONSULTOP",
    "Type": "Antena/Antenna",
    "UOM": "m",
    "VertAccuracy": 0.01,
    "VertConfLevel": 95,
    "VertRefSyst": "MSL",
    "VertResolution": 0.001,
    "WEF": "2022-06-16T00:00:00Z"
  }
}
```

```

    },
    "type": "Feature"
  },
  {
    "geometry": {
      "coordinates": [-5.839117888888889, 37.42873680555555],
      "type": "Point"
    },
    "properties": {
      "ACCION": "INSERT",
      "AMDT": "AIRAC - 07/22",
      "Elevation": 94.125,
      "Frangibility": "No",
      "GFID": "FB1DCEA1-B854-4E13-9268-0142CF73EA89",
      "GeometryType": "Puntual/Point",
      "Height": 43.655,
      "HorzAccuracy": 0.2,
      "HorzConfLevel": 95,
      "HorzRefSyst": "WGE",
      "HorzResolution": 0.001,
      "Identifier": "LEZL-OBS-00034-000-2018",
      "Integrity": "Si/Yes",
      "IntegrityRemark": null,
      "Lighting": "Si/Yes",
      "Marking": "Si/Yes",
      "MeasurementDate": 20181003,
      "MeasurementType": "Radiaci\u00f3n/Radiation",
      "OriginatorId": "AENA",
      "PUB": "2022-05-05T00:00:00Z",
      "SourceId": "CONSULTOP",
      "Type": "Antena/Antenna",
      "UOM": "m",
      "VertAccuracy": 0.2,
      "VertConfLevel": 95,
      "VertRefSyst": "MSL",
      "VertResolution": 0.001,
      "WEF": "2022-06-16T00:00:00Z"
    },
    "type": "Feature"
  },
],
"type": "FeatureCollection"
}

```

In the right figure: [5.12](#), you can clearly see the types of obstacles which have been reported, a vegetation, Trees and Towers namely. The above code snippet shows some of the obstacles out of 309 obstacles in geojson format.

### 5.3.2. Manual Generated Output

Seville airport is undergoing construction so the accuracy and fidelity of the taxiways data is low, never the less with the available maps on geojson.io we could extract the main taxiways and hotspots using manual procedure.

### 5.3.2.1. Taxiways

The taxiways in this airport are shared for commercial and defence purposes, some taxiways are restricted for commercial aeroplanes and designated only for defence aeroplanes.

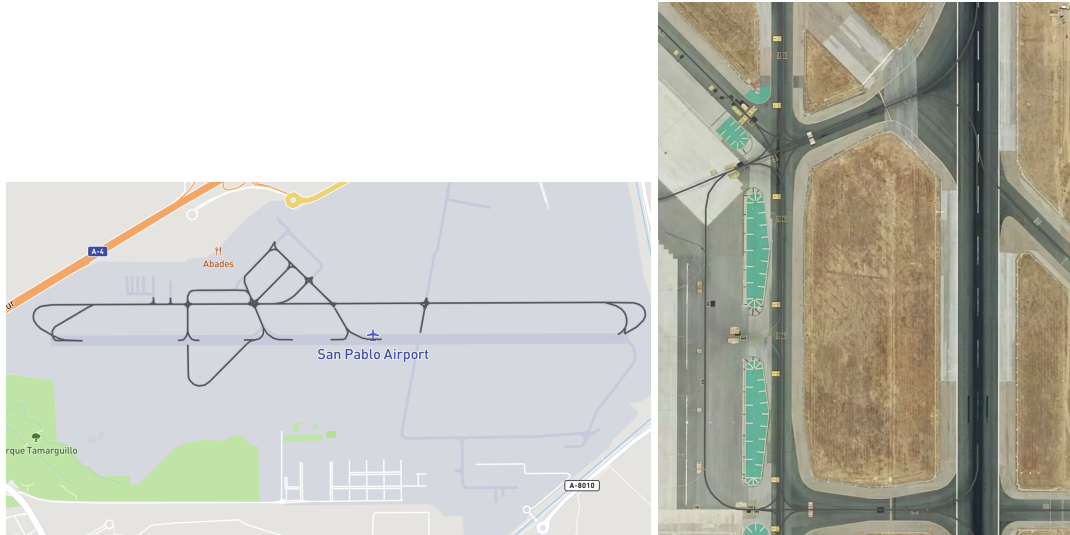


Figure 5.13: Seville Airport Taxiways

### 5.3.2.2. Hotspots

The hotspots in this airport are mainly near the commercial operations as for the defence purpose there's only a little or no traffic mostly.



Figure 5.14: Seville Airport Hotspots



# CONCLUSIONS

The main idea is to provide an alternative to the AIXM format files and the unavailability of the information about the aerodromes in that format also opened an opportunity to introduce a new format with a data structure very flexible, The GeoJSON format has a strict data structure but we can modify the data structure to our needs and generate a new schema and name it as aixm.json how ever to achieve this a strong and robust design approach should be employed which is another topic. For now we have created a base template for generating digital data confined to Informations provided my AIP of Spanish airports, the source code is very flexible to add more methods to modify the data accordingly and keep on improving the original program of digitising the aerodrome data. The another aspect is that there is an additional exit point other than GeoJson, all information is also available in JSON files so it is possible to use that exit point if required for other applications.



# BIBLIOGRAPHY

5. GeoJson Documentation - <https://www.rfc-editor.org/rfc/rfc7946>
5. AIXM Documentation- <https://www.aixm.aero/>
5. Python Camelot Documentation - <https://camelot-py.readthedocs.io/en/master/>
5. Python Pandas Documentation - <https://pandas.pydata.org/docs/>
5. XML declaration - [https://xmlwriter.net/xml\\_guide/xml\\_declaration.shtml](https://xmlwriter.net/xml_guide/xml_declaration.shtml)
5. AIXM XSD mapping - [https://aixm.aero/sites/aixm.aero/files/imce/AIXM51/aixm\\_uml\\_to\\_aixm\\_xsd\\_mapping-1.1.pdf](https://aixm.aero/sites/aixm.aero/files/imce/AIXM51/aixm_uml_to_aixm_xsd_mapping-1.1.pdf)
5. AIXM structure diagram and introduction - <https://aixm.aero/sites/aixm.aero/files/imce/AIXM51/aixm5.1-highlevel.pdf>
5. Fictitious AIXM code - [https://github.com/aixm/donlon/blob/master/EA\\_AIP\\_DS\\_FULLL\\_20170701.xml](https://github.com/aixm/donlon/blob/master/EA_AIP_DS_FULLL_20170701.xml)
5. Python syntax - <https://www.w3schools.com/python/default.asp>