# PlaneSpotting Documentation

*Release 0.0.1*

**Shoumik Dey and & Andreas Hornig**

**Aug 25, 2019**

# Contents:

PlaneSpotting is a software package written in python3 which can decode ADS-B(Automatic Dependant Survelliance Broadcast) from the output provided by dump1090 (a C based Mode-S decoder which is designed for RTL-SDR devices).

The RTL-SDR devices record at 1090 MHz and outputs a binary (.dat) file. This file is processed by dump1090 which provides us with the raw ADS-B frames from that recording in 'mlat' AVR format(ascii hexdump of the frames with the sample position). The processed files serves as the input for this project.

All ADS-B frames do not contain position information in them. To be specific only frames with Downlink Format(DF) 17-18 and Type Code(TC) 9-18 carry the location as the payload. Thus, the next part of this project is to find the location of the plane at the point when a non-position carrying frame is sent using multilateration.

Multilateration is a method by which the location of an aircraft (can be any moving object which broadcasts a signal) using multiple ground stations and the Time Difference of Arrival(TDOA) of a signal to each ground station.

# CHAPTER 1

# Getting Started

## 1.1 Requirements

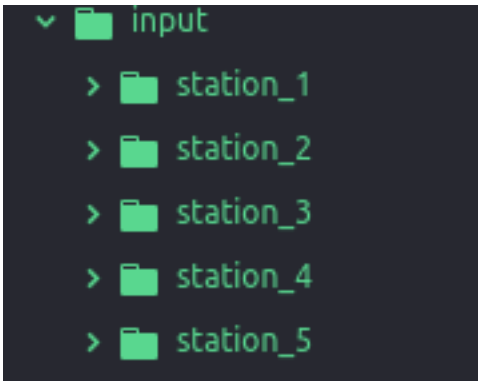
These are the mandatory libraries required:

- numpy
- argparse
- sqlite3
- gzip
- json

## 1.2 Installation

Clone the repositiory into a folder. Currently, the program is set to handle files from 5 stations and the location coordinates are preprogrammed. You can change the coordinates in the **gs_data.json** file present in the **planespotting** folder.

The input folder should be present in the root directory with the name **'input'**




The directory structure should be like this and the input files from each station should be stored inside these station directories.

Run the program executing **python3 main.py** in the command line. The output files will be stored in the **data** folder (if it doesn't exists, it will be created) These files in the data folder contain the decoded data from all the frames. The console output gives shows the frames occuring thrice or more in different station files.

CHAPTER 2

# PlaneSpotting: Technical Documentation

## 2.1 Utility functions

`planespotting.utils.`**`bin2np`**(*binarystr*)

Convert binary string to numpy array.

**Parameters** **`binarystr`** (*String*) – The message in binary

**Returns** Numpy array

**Return type** Numpy.ndarray

`planespotting.utils.`**`const_frame`**()

Retuns the json meta data template when called. The keys in the dictionary contain null value.

**Returns** JSON

**Return type** Python dictionary

`planespotting.utils.`**`const_frame_data`**()

This function, when called, returns a blank template of the data segment of the entire JSON structure

**Returns** JSON

**Return type** Python String

`planespotting.utils.`**`create_folder`**(*path*)

Creates a folder/directory at a given path. Often required when a directory which does not exist and is required for dumping the output

**Parameters** **`path`** (*String*) – Path to the location where the folder/directory is to be created.

`planespotting.utils.`**`get_all_files`**(*filename*)

This function returns the path to all the files present in a directory/folder.

**Parameters** **`filename`** – Path to the directory/folder

**Returns** File paths

**Return type** Python list

`planespotting.utils.`**`get_one_file`**(*filename*)
To open a specific file for a code segment, this returns the single file path in an unit list.

**Parameters** **`filename`** (*`String (path)`*) – Path to the specific file

**Returns** File path (singular)

**Return type** Python list

`planespotting.utils.`**`hex2bin`**(*hexstr*)
Convert a hexdecimal string to binary string, with zero fillings.

`planespotting.utils.`**`hexToBin`**(*hexdec*)
Coverts hexadecimal code to 56 bit binary string.

**Parameters** **`hexdec`** (*`string`*) – Hexadecimal String

**Returns** Decimal string

**Return type** String

`planespotting.utils.`**`hexToDec`**(*hexdec*)
Converts hexadecimal code to binary of the length 4x of the hexadecimal string.

**Parameters** **`hexdec`** (*`string`*) – Hexadecimal String

**Returns** Decimal string

**Return type** String

`planespotting.utils.`**`is_binary`**(*file*)
Checks if the file type is binary or text. It opens the file in text mode initially and and handles if an exception is thrown. If an exception occurs, then the file is non-binary and vice-versa.

**Parameters** **`file`** (*`string`*) – The file of which the type is to be checked.

**Returns** File type (text/binary)

**Return type** Boolean

`planespotting.utils.`**`load_file_jsonGzip`**(*filename*)
Used to load and read a gzipped JSON file present a particular location.

**Parameters** **`filename`** (*`String`*) – Name of the .gz file to be opened along with it's path

`planespotting.utils.`**`load_json`**(*filename*)
Used to load and read a JSON file present a particular location.

**Parameters** **`filename`** (*`String`*) – Name of the .json file to be opened along with it's path

`planespotting.utils.`**`np2bin`**(*npbin*)
Convert a binary numpy array to string.

**Parameters** **`npbin`** (*`Numpy.ndarray`*) – Each bit of the message stored as an element in the array.

**Returns** Binary message in string

**Return type** String

`planespotting.utils.`**`store_file`**(*path*, *file*, *data*)
Used to store a file and put the data in that file at a particular location. If the file does not exist, then the file is created and stored.

**Parameters**

- **`path`** (*`String`*) – Location where the file is to be stored

- **file** (*String*) – The name of the file
- **data** (*JSON (Python dictionary)*) – The contents of the file that is to be saved

`planespotting.utils.store_file_jsonGzip`(*path*, *file*, *data*)

It stores the files by gzipping it to save memory. The file, if it does not exist at the specific location, then it is created and the data is saved in it.

**Parameters**

- **path** (*String*) – Location where the file is to be stored
- **file** (*String*) – The name of the file
- **data** (*JSON (Python dictionary)*) – The contents of the file that is to be saved

## 2.2 File loading and transfer

### 2.2.1 PlaneSpotting main

## 2.3 Frame Identification and Decoding

`planespotting.identifiers.identifier1`(*df*, *tc*)

ADS-B type identifier function for Downlink Format 17 - 18 & Type Code 1 - 4 These messages contain Aircraft identification data

**Parameters**

- **df** (*Integer*) – Downlink Format
- **tc** (*Integer*) – Downlink Format

**Returns** True/False

**Return type** Boolean

`planespotting.identifiers.identifier10`(*df*, *tc*)

ADS-B type identifier function for Downlink Format 4 These messages contain Altitude data

**Parameters**

- **df** (*Integer*) – Downlink Format
- **tc** (*Integer*) – Downlink Format

**Returns** True/False

**Return type** Boolean

`planespotting.identifiers.identifier11`(*df*, *tc*)

ADS-B type identifier function for Downlink Format 5 These messages contain Long ACAS messages

**Parameters**

- **df** (*Integer*) – Downlink Format
- **tc** (*Integer*) – Downlink Format

**Returns** True/False

**Return type** Boolean

`planespotting.identifiers.`**`identifier2`**(*df*, *tc*)

ADS-B type identifier function for Downlink Format 17 - 18 & Type Code 5 - 8 These messages contain Surface Position data

**Parameters**

- **df** (*Integer*) – Downlink Format
- **tc** (*Integer*) – Downlink Format

**Returns** True/False

**Return type** Boolean

`planespotting.identifiers.`**`identifier3`**(*df*, *tc*)

ADS-B type identifier function for Downlink Format 17 - 18 & Type Code 9-18 These messages contain Airborne Position with Barometric Altitude data

**Parameters**

- **df** (*Integer*) – Downlink Format
- **tc** (*Integer*) – Downlink Format

**Returns** True/False

**Return type** Boolean

`planespotting.identifiers.`**`identifier4`**(*df*, *tc*)

ADS-B type identifier function for Downlink Format 17 - 18 & Type Code 19 These messages contain Airborne Velocity data

**Parameters**

- **df** (*Integer*) – Downlink Format
- **tc** (*Integer*) – Downlink Format

**Returns** True/False

**Return type** Boolean

`planespotting.identifiers.`**`identifier5`**(*df*, *tc*)

ADS-B type identifier function for Downlink Format 17 - 18 & Type Code 20-22 These messages contain Airborne Position with GNSS altitude data

**Parameters**

- **df** (*Integer*) – Downlink Format
- **tc** (*Integer*) – Downlink Format

**Returns** True/False

**Return type** Boolean

`planespotting.identifiers.`**`identifier6`**(*df*, *tc*)

ADS-B type identifier function for Downlink Format 17 - 18 & Type Code 31 These messages contain Operation Status data

**Parameters**

- **df** (*Integer*) – Downlink Format
- **tc** (*Integer*) – Downlink Format

**Returns** True/False

**Return type** Boolean

`planespotting.identifiers.`**`identifier7`**(*df*, *tc*)

ADS-B type identifier function for Downlink Format 20 - 21 These messages contain Enhanced Mode-S ADS-B data

**Parameters**

- **df** (*Integer*) – Downlink Format
- **tc** (*Integer*) – Downlink Format

**Returns** True/False

**Return type** Boolean

`planespotting.identifiers.`**`identifier8`**(*df*, *tc*)

ADS-B type identifier function for Downlink Format 5 These messages contain SQUAWK/IDENT data

**Parameters**

- **df** (*Integer*) – Downlink Format
- **tc** (*Integer*) – Downlink Format

**Returns** True/False

**Return type** Boolean

`planespotting.identifiers.`**`identifier9`**(*df*, *tc*)

ADS-B type identifier function for Downlink Format 0 These messages contain Short Air to Air ACAS messages

**Parameters**

- **df** (*Integer*) – Downlink Format
- **tc** (*Integer*) – Downlink Format

**Returns** True/False

**Return type** Boolean

## 2.4 Data Extraction from ADS-B frames

## 2.5 Airborne Parameters Calculation

`planespotting.calculator.`**`NL`**(*lat*)

This function returns the number of latitude zones from the latitude angle. The returned value lies inbetween [1, 59]

**Parameters** **lat** (*Float*) – The latitude angle

**Returns** Number of latitude zones

**Return type** Integer

`planespotting.calculator.`**`calculate_position`**(*all_seen_planes*, *data*)

This function calculates the position of all the icaos successively, both using the 'Globally unambigous' method and the 'Locally unambigous method' Even after several corrective checks on the frames, there is still a possibility for the frames to carry wrong data. Thus, the calculated position is verified by both the steps and an average of all the positions is mantained which acts as a reference position.

**Parameters**

- **all_seen_planes** (*Python List*) – List of unique icao addresses present in the recording
- **data** (*Python dictionary*) – JSON containing the entire set of frame data found in the recording

**Returns** The updated data dictionary with the locations (only for position frames)

**Return type** Python dictionary

`planespotting.calculator.`**`calculate_velocity`**(*data*)

Calculation of velocity from DF 17-18 & TC 19 frames.

**Parameters** **data** – JSON containing all the frames and data keys

**Returns** JSON with the velocity key filled up

**Return type** Python Dictionary

`planespotting.calculator.`**`convert_position`**(*data*)

Conversion of geographical coordinates present inside data to cartesian coordinates

**Parameters** **data** – JSON containing all the frames received and the decoded data stored in each key.

**Returns** JSON with the coordinates updated

**Return type** Python dictionary

`planespotting.calculator.`**`get_cartesian_coordinates`**(*lat=0.0*, *lon=0.0*, *alt=0.0*, *meter=True*)

Converts Geographical coordinates to cartesian coordinates.

**Parameters**

- **lat** (*Float*) – Latitude coordinate in degrees
- **lon** (*Float*) – Longitude coordinate in degrees
- **alt** (*Float*) – Altitude in feet
- **meter** (*Boolean*) – State variable for determining return unit (metres/feet)

**Returns** Latitude, Longitude, Altitude in metres

**Return type** Float

`planespotting.calculator.`**`get_geo_coordinates`**(*x*, *y*, *z*)

Converts cartesian coordinates to geographical coordinates.

**Parameters**

- **x** (*Float*) – Latitude in metres
- **y** (*Float*) – Longitude in metres
- **z** (*Float*) – Altitude in metres

**Returns** Latitude, longitude, altitude in geographic coordinates.

**Return type** Float

`planespotting.calculator.`**`get_meanposition`**(*data*, *relevant_planes_id*, *hit_counter_global*, *latitudeMean_global*, *longitudeMean_global*)

Calculation of the mean of all the decoded positions (lat, lon). Returns the mean (lat, lon) This mean(lat, lon) is used as the reference position for the locally unambigous method.

**Parameters**

- **data** (*Python List*) – Contains all the frames seen in the recording.
- **relevant_planes_id** (*Python List*) – List containing frames which contain airborne position data.
- **hit_counter_global** (*Long*) – Number of frames considered during average calculation at each iteration.
- **latitudeMean_global** (*Float*) – Mean of all calculated latitudes which gets updated after each iteration.
- **longitudeMean_global** (*Float*) – Mean of all calculated longitudes which gets updated after each iteration.

**Returns** Latitude, Longitude

**Return type** Float

`planespotting.calculator.`**`lat_index`**(*lat_cpr_even*, *lat_cpr_odd*)

Used to calculate the latitude index using the odd and even frame latitudes.

**Parameters**

- **lat_cpr_even** (*Float*) – The latitude data from an even frame
- **lat_cpr_odd** (*Float*) – The longitude data from the odd frame with the same icao as the even frame

**Returns** Latitude index

**Return type** Float

`planespotting.calculator.`**`latitude`**(*lat_cpr_even*, *lat_cpr_odd*, *t_even*, *t_odd*)

Calculation of the latitude of a given aircraft with a set of successively received odd and even pair of frames. This type of calculation is also known as 'Globally unambigous position calculation' in which you need an odd and even frames from the same icao received one after the another. The order at which the frames at which the frames were received also determines the methodology of the calculation. For more info: https://mode-s.org/decode/adsb/airborne-position.html

**Parameters**

- **lat_cpr_even** (*Float*) – Latitude data from the even frame
- **lat_cpr_odd** (*Float*) – Latitude data from the odd frame
- **t_even** (*Long*) – Time/Sample Position of the even message in the recording
- **t_odd** (*Long*) – Time/Sample Position of the odd message in the recording

**Returns** Latitude in degrees

**Return type** Float

`planespotting.calculator.`**`longitude`**(*long_cpr_even*, *long_cpr_odd*, *t_even*, *t_odd*, *nl_lat*)

Calculation of the Longitude coordinate with the same set of successively received odd and even pair of frames as used in the latitude() function. In this function, the order of the odd and even frames is considered for calculation

**Parameters**

- **long_cpr_even** (*Float*) – Latitude data from the even frame
- **long_cpr_odd** (*Float*) – Latitude data from the odd frame
- **t_even** (*Long*) – Time/Sample Position of the even message in the recording
- **t_odd** (*Long*) – Time/Sample Position of the odd message in the recording

**Returns** Longitude in degrees

**Return type** Float

`planespotting.calculator.`**`pos_local`**(*latRef*, *lonRef*, *F*, *lat_cpr*, *lon_cpr*)

Calculation of position using one frame only or 'Locally unambigous position calculation'> It uses a reference location to remove the ambiguity in the frame.

**Parameters**

- **`latRef`** (*`Float`*) – Reference latitude (ground station or previous known location)
- **`lonRef`** (*`Float`*) – Reference longitude (ground station or previous known location)
- **`F`** (*`Integer`*) – Odd/Even bit of the frame
- **`lat_cpr`** (*`Long`*) – Latitude data from the frame
- **`lon_cpr`** (*`Long`*) – Longitude data from the frame

**Returns** Latitide, longitude

**Return type** Float, Float

## 2.6 File Overlap check and search of unique signals

# CHAPTER 3

# Tutorials & Usage

Here is a presentation of an example as to how to use the current state of the project.

**Note:** *Note:* This project is currently tested in Python3.7 and is also being developed using Python3.7

Input: * The dump1090 mlat output from an raw IQ recording/file.

Arguments used in dump1090:

| | |
|---|---|
| **--mlat** | So that the output is in AVR format |
| **--ifile** | To load an IQ file |
| **--gain** | By default this value is set to (-10) |

A typical ADS-B 112 bit frame in AVR format:

```
@0000000929E28e3ff6e6990c4684000011548194;
```

AVR format is identified by '@' and the ';' at the beginning and at the end respectively.

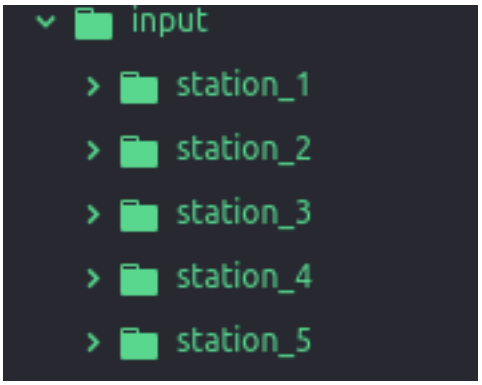
The output from dump1090 which serves as our input should be named in the following manner:

```
[date]_[station_name]_[start_time_of_recording_epoch]_frame.dat
```

Example of a typical file name:

```
20190206_station1_1566723047_frame.dat
```

Such of these files should be ordered in the following manner in a folder named **'input'** in the home directory:




Update the coordinates of each station in the planespotting/gs_data.json file

```
{
  "station_1":{
    "lat":latitude,
    "lon":longitude,
    "alt":altitude
  },
  "station_2":{
    "lat":latitude,
    "lon":longitude,
    "alt":altitude
  },
  "station_3":{
    "lat":latitude,
    "lon":longitude,
    "alt":latitude
  }
}
```

**Note:** 'latitude' and 'longitude' should be in degrees.

Now run the script by using this command in the console:

```
python3 main.py
```

While the script runs, it will save all the decoded frames and the data in the **'data'** folder in the home directory.

CHAPTER 4

# Indices and tables

- genindex
- modindex
- search

# CHAPTER 5

# Copyright and License

# Python Module Index

## p

# Index