Airport quality: holding and go-arounds

Alexander Renz-Wieland
Vrije Universiteit Amsterdam
a.r.w.renzwieland@student.vu.nl

Hugo Wallenburg
Vrije Universiteit Amsterdam
h.m.h.w.wallenburg@student.vu.nl

1. INTRODUCTION

Aircraft surveillance technology is currently undergoing a major transition. It is moving from exclusively relying on conventional air-traffic management systems such as Primary and Secondary Surveillance Radar towards also incorporating a new generation of air-traffic management tools. Drivers for this change are, among others, increased air traffic, increased precision achieved by the new systems and potential improvements for aircraft routing.

One key component of the next generation tools is Automatic Dependent Surveillance - Broadcast (ADS-B). Its core idea is that planes determine their own location using the on-board GPS equipment and transmit that information regularly. Most countries demand that airplanes are equipped with ADS-B before a certain point in the near future. In European airspace, planes have to be equipped with ADS-B by 2017 [6].

Most commercial airplanes in Europe are already equipped with ADS-B hardware and emit signals during flight. Available to us is a subset of these signals picked up by receivers of the OpenSky network, collected in September 2015.

In this project, our focus lies on holding and go-arounds. It seems odd that in a time of very advanced technology for position detection and sophisticated routing algorithms even in consumer-grade hardware, aircraft routing is still not able to manage planes in a way that eliminates holding. Every minute spent in holding leads to a considerable amount of wasted fuel, adding to the already bad carbon footprint of air traffic in general. Instead of holding, a plane could (potentially) have taken off later or flown with reduced speed for appropriate time. We want to find out at which airports holding is frequent and which airlines frequently fly holding. As an end result, we will present a ranking of airports and airlines.

We aim to do the same for go-arounds, although from an environmental perspective, they are not as bad as holding, as they are mainly done for safety purposes and therefore much harder to prevent.

2. RELATED WORK

2.1 ADS-B data

Our data set is comprised of ADS-B messages. Aircraft equipped with this technology determine their position and their movement using onboard satellite navigation systems and then use ADS-B transmitters to broadcast this information twice every second [6]. Less frequently, the hardware transmits identification, status, and urgency information.

The ADS-B protocol does not encrypt message content, which allows actors external to the professional aerospace industry to gather data for their own purposes, for instance for scientific research. OpenSky (`https://opensky-network.org`) is a participatory sensor network with the goal to collect ADS-B messages in its sensing range for further analysis [6]. OpenSky provide their data free of charge to researchers wanting to analyze it.

During the period we have data for there were fifteen active sensors in the network; apart from some sensors apparently having been deactivated since the paper by Strohmeier et al., the coverage we see in our data seems to correspond well to the network coverage they describe.

There are, however, limitations to this data that we have to take into account. The most obvious one being that there is no data outside sensor range, as can clearly be seen in fig. 1. Other limitations and how they affect our results will be discussed in section 6.

ADS-B transmits different message types. Most frequent are *airborne position* and *airborne velocities* messages. Airborne position messages transmit latitude, longitude, and altitude of the plane. Airborne velocities messages transmit velocity, vertical rate, and heading. In *aircraft identification* messages, the hardware broadcasts the current call sign the plane is flying under[1].

2.1.1 Using ADS-B data for event detection

In their 2015 paper [6], Strohmeier et al. present OpenSky Network in general, a discussion of security features (or rather, the lack thereof), detection of forged messages, as well as detection of "unusual events happening in the coverage area of the sensor network" [6, p. 12].

The section about event detection is of special interest to us, as they perform a similar kind of analysis to the one we are doing. By identifying distinct *business aircraft*, *military*

[1]More information on message types and decoding them can be found at `http://adsb-decode-guide.readthedocs.org/en/latest/introduction.html#ads-b-message-types`.

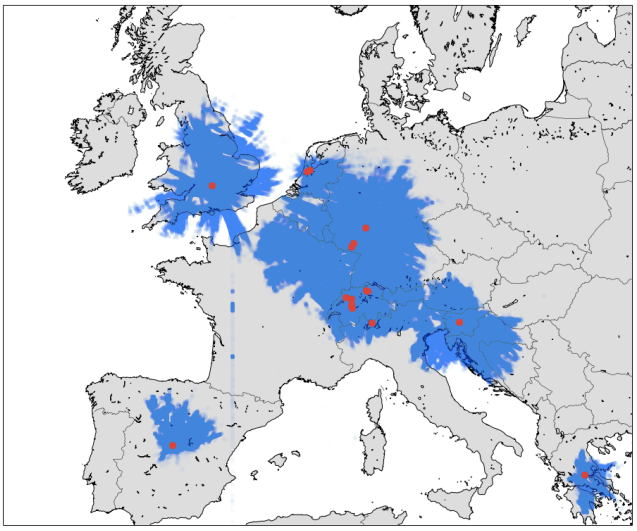

Figure 1: Map of Europe with position messages (blue) and sensor locations (red) superimposed.

aircraft, and *helicopters* they apply time series analysis to, fairly reliably, detect large events by distinguishing them as outliers in the data set.

There is also a section about how timing idiosyncrasies can facilitate distinguishing different brands of ADS-B transmitters as well as telling fake messages from real ones through anomaly detection [6]. This shows that the data can be used in a multitude of ways in several fields.

2.2 Analysis approaches

2.2.1 *Data mining approaches*

Han et al. describe methods to mine moving-object data [2]. They compare and explain different data mining approaches, both supervised and unsupervised, for application to data of moving objects. Their approaches are not ideally suited for our purposes, as we are looking to identify very specific events without labeled instances; we have no instances from which supervised methods could learn.

An imaginable approach to our problem using these methods would be to identify a number of holding patterns and go-arounds manually and then let the mining algorithms learn from these instances to find other instances in a bootstrapping fashion. Still, for our goals, the problem of initially identifying instances remains.

2.2.2 *GPS traces*

Liu et al. use GPS locations reported by taxis in Shanghai to infer Shanghai's road network [3]. For this, they describe different algorithms, using either the individual reported locations or connected consecutive data points. In their paper, they focus on working with slightly unreliable granular data, as the taxis report their locations as infrequently as once a minute and the taxis do not carry high-quality GPS equipment. On the other hand side, this data is available in large amounts.

Methods like this would be interesting for this project to identify the pathway structure in holding stacks. Using this information, it would be possible to map planes' trajectories to these holding pathways. If a plane is flying on one of the identified holding pathways, one is able to declare the plane to be in holding. But, as before, this would require some way of identifying certain pathways as holding pathways.

Instead of looking at the three-dimensional pathways, one could identify two-dimensional holding areas or, more fine-grained, the racetrack-like holding pathway. This data could then be used to identify planes flying holding patterns. Areas could presumably be identified to be holding areas more easily than holding pathways.

3. RESEARCH QUESTIONS

3.1 Questions

In this project, we want to answer the following main questions:

- How frequently do airplanes fly holding when landing and how much time is spent in holding per flight on average? How does this compare between airports and between airlines?
- How frequently do airplanes reattempt landings as a result of go-arounds and how much time is spent doing that? How does this compare between airports and airlines?

We are looking to answer these questions using our data set of ADS-B messages from Europe during September 2015. As an end result we want to quantify airport quality using these metrics. We present our findings in a concise manner in the form of a lightweight web page where the user can sort the presented data flexibly in order to compare the quality and business of airports and airlines alike.

In order to answer these questions, we need to answer some additional questions. ADS-B data does not explicitly include the destination airport of a flight. It also does not contain information about where a flight starts and where it ends. So additionally, we are looking to answer:

- How to identify flights from ADS-B data?
- How to identify the destination airport of a flight from ADS-B data?

3.2 Technology

In order to find answers to these questions in the data available to us, we need to pick appropriate tools. We identified two distinct sides to this problem: Firstly, we are challenged with developing algorithms which reliably identify the events we are looking for. The technology we choose to use for this should enable us to iterate quickly, switching back and forth between creating or altering an algorithm and evaluating it; Secondly, we need to be able to apply those algorithms to a large data set, meaning the algorithms need to be scalable and run performantly.

As the requirements for both fields are rather distinct, we chose to use two different sets of tools. We use Spark with Scala and Java for large-scale data processing, including data extraction, data decoding, and large-scale deployment of our developed algorithms. We chose Spark because it is fast, convenient, and intuitive to use, as well as being both flexible and scalable [5].

On the other side, we decided to develop algorithms locally on subsets of the data. To that end we use tools from

the Python data analysis stack, namely Jupyter Notebooks[2], Pandas[3] and Matplotlib[4]. This set of tools allows us to develop new algorithms and test them on subsets of data immediately, leading to very quick iteration. Matplotlib makes it convenient for us to visualize the data. In particular, we use a specialized library called Basemap to plot positional data on top of a map of Europe at various configurable resolutions[5].

In addition, we use the java-adsb library provided by OpenSky[6] to decode the raw message part of the ADS-B data transmissions. This library contains methods for decoding the different ADS-B messages, in addition to error handling related to corrupt or otherwise faulty message data.

4. PROJECT SETUP

4.1 Extracting data

First of all, we set up a process to read the data and decode relevant information. The data available to us consists of the raw transmitted ADS-B messages, stored in avro format[7]. As mentioned in section 2.1, there are different message types, of which *position* and *velocity* are the most relevant for our purposes. These two message types make up the vast majority of our data: in the first day of our data set, September 2nd 2015, these two message types make up around 85% of all received messages. Identification messages make up another 5%.

Each data point contains a *rawMessage*, encoded data, as well as general information about the receiving antenna. The raw message contains a message type, information specific to that message type, and the *ICAO* 24-bit address: a unique identifier for *one* ADS-B transmitter. As these transmitters are not usually transferred between aircraft, we can reliably use these addresses to uniquely identify airplanes. In order to extract data from the raw message string, we use the java-adsb[8] library provided by OpenSky. In order to use java-adsb in Spark, we needed to make some slight modifications to the code to be able to serialize the parts of the library we are using at the worker nodes.

To cut down on the size of transmitted data, no positional message contains the entire information about the current aircraft location. Messages contain either odd or even position *frames*, two of which are needed to pinpoint the exact location given in a message.

To extract the data, we do the following:

1. Extract the *ICAO* from the raw message, using *java-adsb*,
2. key and group the data by *ICAO*,
3. flatmap over the grouped data, passing an unordered list of messages, and
4. pass the messages of one *ICAO* in chronological order to extract the data we are looking for.

This way of extraction is rather scalable, as the extraction can be done in parallel for each airplane. A limitation is the number of data points that arrive at one worker in the flatmap step, as too many data points could cause memory problems at the worker. When extracting data over a long time frame, the number of these data points might increase drastically. To make our extraction process applicable for larger time periods, we group the data by both day and *ICAO*. For this, we cut between two days in the night. In our experiments, we did not miss a significant number of landings as in Central Europe, there are usually no, or very few, landings during the night.

We are aware that *groupByKey* is expensive, as a lot of data needs to be shared, but seeing as we need to pass the data in chronological order for each ICAO and do not have an associative or commutative operation, we are forced to use it.

Using this extraction process, we can extract data of interest in a flexible way. We used this scheme for different types of extractions. For example, we extracted data points of specific planes, data points from certain locations and data of planes which flew certain patterns.

4.2 Understanding data

We imported these extracted subsets into our local data analysis environments, the Jupyter notebooks. There, we were able to work with the data interactively. Some effort was required to get a first understanding of the data we were working with; our first efforts showed data that we were not expecting. This was mostly caused by the limitations to the data available to us. First, the antennas of the OpenSky network do not, for the most part, pick up signals when planes are close to the ground. Second, we have no data points when planes fly over areas where there is no antenna coverage. And third, there usually is a delay before antennas start picking up the signals from starting planes. These limitations make our tasks of discovering landings, destination airports, holding patterns and go-arounds more involved.

Figure 2 shows these limitations using the example of one plane. The figure shows the altitude reported by the plane with *ICAO address* `3c5ee8` on September 2nd. In the morning and early afternoon, the plane is flying in areas with good antenna coverage. Data is only missing when the plane is close to the ground and when the antennas need some minutes to pick up the plane's signals. In the late afternoon and the evening, the plane is flying in areas with worse data coverage. During this time, data is missing for longer intervals. It also looks like data for some landings is missing completely.

4.3 Algorithm development

4.3.1 Landings

Initially we were hoping to be able to infer from the data, or look up externally, flight routes with information on origin and destination airport, and scheduled departure and arrival times. In our investigation of the data we discovered that this was exceedingly difficult to deduce given the unforeseen sparsity we had to deal with.

As our task is centered around identifying certain events or patterns that typically take place right before landing,

[2] `http://jupyter.org/`
[3] `http://pandas.pydata.org/`
[4] `http://matplotlib.org/`
[5] `http://matplotlib.org/basemap/`
[6] `https://opensky-network.org/network/projects/20-java-adsb`
[7] `https://avro.apache.org/`
[8] `https://github.com/openskynetwork/java-adsb`

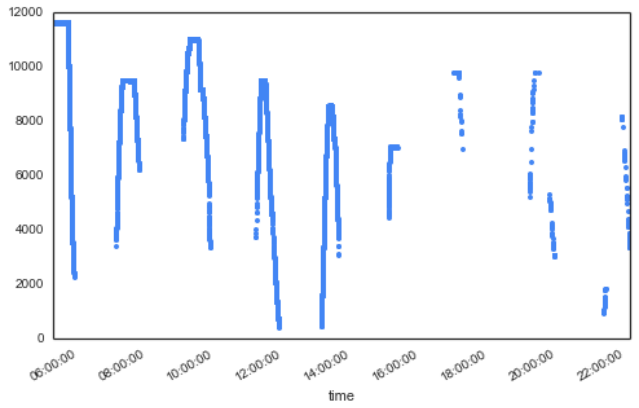

Figure 2: Altitude profile for `ICAO 3c5ee8` on September 2nd 2015

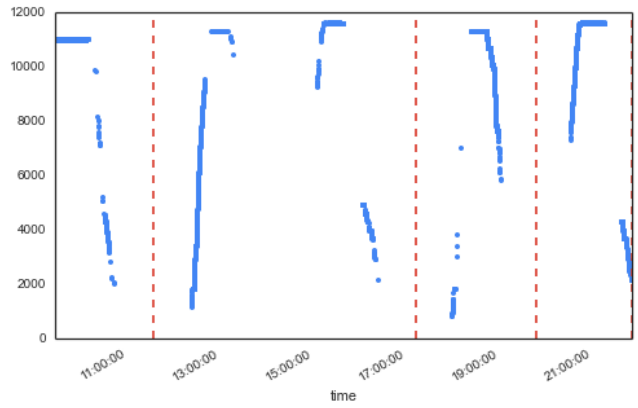

Figure 3: Altitude profile (blue) and identified landings (red) of plane `ICAO 406bbb` on September 2nd, 2015

landings are what we focused on at the beginning. It is possible to infer discrete flights by identifying landings (and the corresponding destination airport), without necessarily having to identify take-off (these are implied by the previously identified landing).

To identify a landing from a sequence of transmitted positional and velocity data points without knowing the destination airport, most of our ideas centered around looking for descents. We were also thinking about looking at the distance to the ground at the current position of the plane. There are different numbers reported by the aircraft that are potentially interesting for this. Besides reported *altitude*, one might find *vertical rate* helpful.

The main difficulty in doing this is that the sensors stop receiving messages when the airplane gets close to the ground. Nearby terrain might in some cases severely affect reception.

To detect landings even in situations with sparse data, we mainly look for significant descents, meaning descents of more than 1500 feet. We had a professional airline pilot confirm our hypothesis that a plane will land soon if it descends 1500 feet. In addition to the descent, we look for a pause in the data after the descent. For identifying descent, we use reported altitude rather than the vertical rate of a plane, as it is more robust towards missing data points. If we miss some data points of reported altitude, we can still reliably calculate the difference. Vertical rate gives an idea of the current rate of descent, but if there is missing data, it is risky to make statements about the time between the data points.

Figure 3 shows the identified landings for one plane throughout the first day of data. The graph clearly shows periods of no data being received, along with what looks like a landing our algorithm missed at around 13:40. For this specific flight we have almost no data from the descent, and the little descent we do see is not drastic enough for our algorithm to flag it as a landing.

4.3.2 *Destination airport*

The next part is to identify the destination airport of a flight, as the destination is not explicitly contained in the data. For this, we discussed a couple of approaches.

One of our first ideas was to use the call sign – which is transmitted in the ADS-B identification messages – and a database connecting call sign and routes to identify the destination airport. Unfortunately, our research showed that there is no such database publicly available. In order to achieve similar goals, FlightAware and similar services apparently manually curate databases. An option would have been to write a script to scrape the database of Flightradar24, for example. This approach has three limitations though. First, we have around 29,000 unique call signs with landings in our data. Second, the data we can scrape is current data and not historical data for September 2015. And third, apparently, there seem to be airlines that do not use the same call sign for a route every time.

So we opted to detect the destination airport from the available data. This task sounds easier than it is because, as mentioned, the data for planes flying near the ground is missing, so the last part of the landing approach is missing. The point where the antenna stops to pick up the signals depends on antenna location, the distance of the antenna to the destination airport and the obstacles between plane and antenna.

Our first approach was to take the last reported position before a landing and look for the closest airport. By closest, we mean closest in terms of euclidean distance on a 2D-map, considering only latitude and longitude. Additionally using altitude might give a slight improvement for some cases, we will touch on that later. To do this, we use external data, a list of airports with their locations from OpenFlights[9]. Unfortunately, the list contains also small, non-commercial airports. So looking for the closest airport of the last data point frequently results in those. To prevent this, we filtered the airports list. First, we tried to get rid of all airports that have no three-letter codes. But it turned out, that still some non-commercial airports in our coverage area remain. So we used a list of commercial airports by region to restrain the list to European commercial passenger airports. In our filtered list, we have 369 remaining airports, which can be seen in fig. 4.

With the filtered list, the mapping works very well if the distance between commercial airports is large enough. Usually this is the case and we are confident that our mapping is correct, but for certain areas this method is inaccurate. London has five commercial airports; Heathrow has four hold-

[9] `http://openflights.org/data.html`

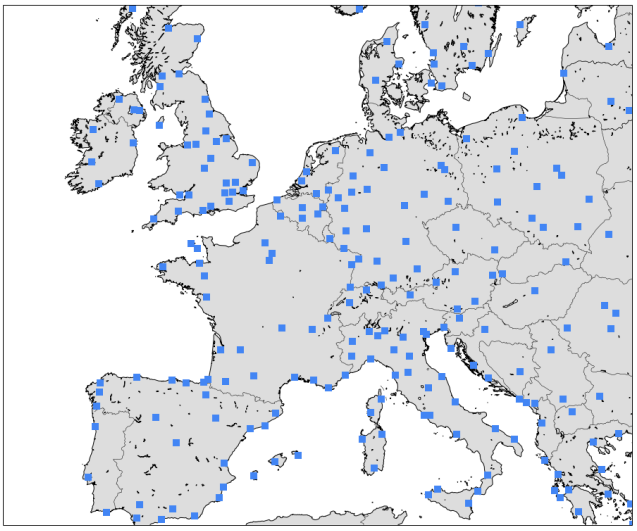

Figure 4: Commercial passenger-airports in Europe, zoomed to the part relevant for our data

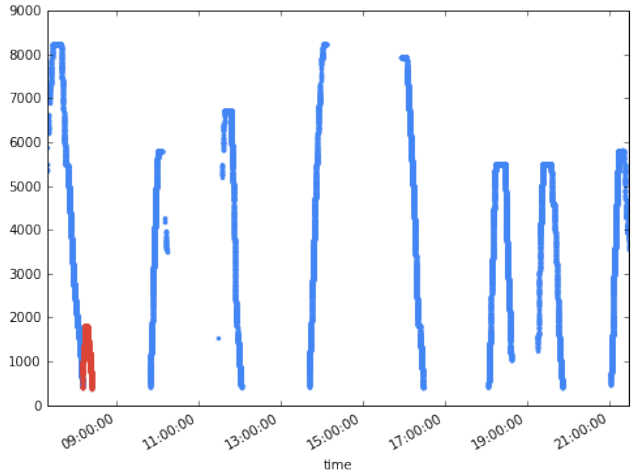

Figure 5: Altitude over time for `ICAO 4b17a3` on September 2nd 2015. Go-around in red for landing in afen Zürich at 08:24.

ing stacks, designated for each cardinal direction. If a plane approaches Heathrow from the east, it will hold in its designated stack. If we lose data right as the airplane exits holding, it is practically impossible to tell from only positional data whether the airplane was headed for London Heathrow or London City Airport.

We did have some ideas to improve this. One of them was to look at airplane trajectories and extrapolate the landing approach should it be needed – essentially, following the path of the plane and checking when it hits the ground. However, the amount of data points to use as basis for extrapolation is a difficult problem, especially as there might be missing or erroneous data points. Both considering too few and too many data points could lead the algorithm to arrive at the wrong conclusion. The fact that data is lost at different stages for every airport only compounds this problem.

Another idea was to consider the last data point before the landing and the first one after the landing, and look for the closest airport to the middle point between them. A slight caveat to this is that there is the delay until antennas pick up the signal from planes that are taking off. Otherwise, we think this might be a good approach.

All in all, we decided to go with the simple approach here, as we have no way of evaluating the identified destination airports on a larger data set, so we have no way of telling which approach works better than another. We can look up the destination of individual call signs manually (though finding accurate information about old routes might be difficult without the right contacts).

4.3.3 Holding

Our initial ideas towards detecting holding patterns involved identifying *pathways* containing holding patterns and match flight paths to those [3] and finding fixed areas designated for holding patterns and match circling aircraft patterns to those areas. The latter was inspired by reading about London Heathrow, which has four main designated holding areas, or *stacks*, for approaching aircraft; one for each cardinal direction.

First, we researched airplane holding and put together a collection of facts that we figured could be helpful for identifying or confirming holding patterns:

1. Holding patterns consist of racetrack-like circles.
2. Consecutive circles are separated by about 1000 feet vertically.
3. Completing one round takes about four to six minutes.
4. Holding patterns have designated entry points and procedures.
5. There are strict regulations for aircraft speed in holding patterns.

We evaluated various ideas such as looking for aircraft flying at certain holding speeds for periods of time, aircraft passing the same location twice with roughly 1000 feet difference, or trying to look for aircraft hitting the entry point locations, but in our early experiments, an approach based mostly on facts 1 and 3 delivered the most reliably results. Our algorithm relies on the fact that planes turn a full 360°in one direction every 4-6 minutes when they are flying holding. We had hypothesized that this would not be enough to reliably match most of the holding patterns, but it turns out this is actually the case. Therefore, we decided to stick with this approach rather than pathway identification.

4.3.4 Go-arounds

Go-arounds eluded us for quite some time. Compared to holdings they are exceedingly rare; we struggled for a while to find any go-around instances at all in our local data set (limited to September 2nd 2015).

Our first good idea that we implemented and tested was to look at which flights spent the most time flying under 6000 feet after reaching cruising altitude. This gave us an enormous list that included occurrences of what looked like training flights, helicopters, and stunt flights. After inspecting plots of the four hundred aircraft that spent the most

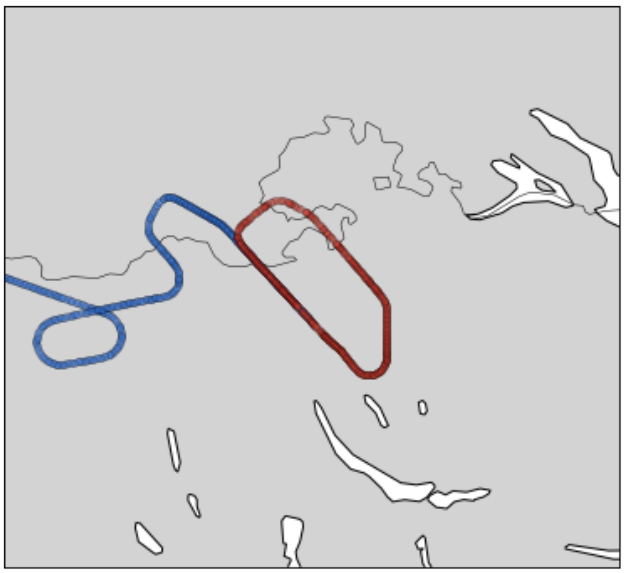

Figure 6: Positions for ICAO 4b17a3 on September 2nd 2015. Showing only positions for landing at 08:24. Positions in go-around shown in red.

time under 6000 feet we finally found our first example of a real go-around, see figs. 5 and 6 for altitude and position plots. Essentially they look like small bumps in altitude right before landing.

Based on the knowledge of what go-arounds look like in our data we devised a method for reliably identifying a go-around:

1. Attempt to rule out flights that have too little data to properly identify a valid go-around. This is done by checking that the final data point for a flight is within 1000 feet of the destination airport. If the final point is too far off we cannot reliably tell if the altitude bump we are looking for is attributable to an actual go-around or just an ascent followed by a descent.

2. Go-arounds we have spotted in the data have not ascended more than about 2000 feet after first having attempted to land. We limit the search to ascents of 1000 to 2000 feet to find a bump that is deliberate, yet small enough still to be a go-around.

3. Make sure the start of the bump is at approximately the same level as the final landing, that is making sure the bump is symmetrical with regards to altitude changes.

4. Identify the start of the bump.

This method makes some pretty drastic assumptions that possibly might lead to missing some go-arounds, but with the amount of missing data we are working with this is a reality we have to accept. Experiments were made where we slacked up on the restrictions, resulting in a significant amount of false positives.

Table 1 shows all of the go-arounds we successfully identified. In 0.4% (80 of approx. 20.000) of our flights we detect a go-around. This is around what we expected to see with these limitations, comparing it to what is being seen in practice [1].

Airport	Go-arounds
Zürich	31
Frankfurt Main	23
Madrid Barajas	10
London Luton	5
Milano Malpensa	4
Bern Belp	2
Ljubljana	1
London Heathrow	1
Ajaccio Campo Dell Oro	1
Birmingham	1

Table 1: Identified go-arounds in September 2015

The data in table 1 matches anecdotal numbers for go-arounds to a degree, but more importantly it shows how dependent upon pristine data we are for this detection to work. We have mentioned how we tend to lose data when the airplane is close to the ground, especially so with spotty sensor reception. This is disastrous for go-around detection as it relies heavily on low-altitude data by nature.

Consequently, we can only detect go-arounds at airports with good data coverage, for example Frankfurt Main or Zürich. We can not detect go-arounds for airports with bad coverage, such as Paris Charles De Gaulle or Munich, as we do not see any data points of the actual go-arounds. Additionally, the total number of go-arounds is very low (see table 1). For these two reasons, we decided not to compare go-around numbers between airports or airlines.

4.4 Large-scale implementation

After initially developing each of these algorithms individually and locally in Python, we then continually merged them into one combined Java program, which we can use in Spark. To do this, we were deciding between Java and Scala. We did not consider Python as not all Spark functionality is available in Python. We decided to use Scala for high-level code such as adding columns, keying and grouping data, as it is very convenient and expressive to do these kind of operations in Scala and because there is Spark's interactive Scala shell. For the data processing method itself, we opted for Java, as we both are much more familiar with Java and especially its data structures.

We consolidated all our individual algorithms into one single pass over the data in Java. To run this, we are using the scheme described in section 4.1. In the data processing step, we apply the consolidated algorithm. We did run different versions of this consolidated algorithm on the entire data set successfully. To ensure that the algorithm runs correctly also on larger data sets, we first tested it on smaller subsets. After running it on the entire data set, we continually conducted a couple of experiments on the data to check how well the extractions work, section 5 will elaborate on these.

The run time of applying this algorithm to the entire data set slightly differed from version to version. Usually, the job took around 14 hours to complete.

4.5 Data analysis

The consolidated algorithm extracts landings with additional information about the landing. That means, it gives us a list of all landings in the data. For each landing, it notes

the *ICAO* and the current call sign of the landing plane as well as the time and destination airport of the landing. Additionally, for each landing, it gives information about potential holding done before this landing. Specifically, it notes how long the plane flew holding and when and where it started its holding. Additionally, it stores in which direction the plane flew its holding (clockwise or counter-clockwise). Extracting this information from the raw data of all days results in a 15MB CSV file.

That size is well manageable in local data analysis environments. So we load this data into Python to aggregate the results and prepare the data for display on the website. We group the landings by destination airport and calculate for each airport:

- the number of seen landings,
- the percentage of landings preceded by holding, and
- average time spent in holding per landing.

To get insights into differences between airlines, we extract the airline of the flight from the call sign. To do this, we use data about call signs of airlines [10]. The mapping to airlines works rather well, only around 3.5% of the landings end up without a mapped airline. Using the extracted airline information, we group by airline and calculate the same figures as for the airports.

The resulting data can be explored on the website we created for this project. We did not put online data for airports for which we see less than 300 landings or airlines for which we see less than 300 landings, because we think that data gives an unrepresentative impression of the airport or the airline, respectively. The cutoff at 300 is chosen freely by our own best judgment. We provide the full data sets in the data directory handed in with this report. We did not export any data on go-arounds for the reasons laid out in section 4.3.4.

5. EXPERIMENTS

To verify the results of the holding pattern extraction, we conducted a couple of experiments on the extracted data. We want to make sure that we accurately identify the events we are looking for and that we capture them exhaustively.

First off, we wanted to know whether the number of landings, which we identified, roughly matches to what we would expect. To do that, we used external data about the yearly aircraft movements at major airports, reported by the Airports Council International [4]. An aircraft movement is either a take-off or a landing. We have data for the five largest airports in our data set. For these airports, we compare the number of extracted landings at this airport to the reported number of movements at this airport. To compare the numbers of one day of identified landings to the yearly movements, we adjust the movement figures. For each airport AP, table 2 shows the quotient of

$$quotient_{AP} = \frac{identified\_landings_{AP} \cdot 2 \cdot 365}{airport\_movements_{AP}}.$$

Note that we are adjusting by 2 to account for the fact that movements can be either a take-off or a landing and by 365 to compare yearly and daily numbers.

[10] `https://en.wikipedia.org/wiki/List_of_airline_codes`

Airport	Quotient
Frankfurt Main	1.03
Paris Charles De Gaulle	0.38
London Heathrow	0.42
Munich	0.17
Amsterdam Schiphol	0.98

Table 2: Identified landings in relation to adjusted airport movements

Slight differences as for example in the Frankfurt Main numbers are expected, as the reported movements are from the year 2013 and our data is from 2015, but the larger differences for other airports were a bit unexpected at first. Upon inspecting the distance of each airport to the closest antenna the numbers made sense. For both Amsterdam Schiphol and Frankfurt Main, there are antennas very close by, so we have excellent data coverage there. Both Paris Charles De Gaulle and Munich lie right at the edge of the antenna coverage, so antennas do not pick up all incoming flights. Even with antennas close by, we do not see data from planes below a certain altitude. The further the antenna, the worse this gets. For London Heathrow, it looks like we are indeed missing a lot of flights. We estimate that we are falsely declaring the destination airport to be London City for some of the Heathrow flights. The sensor location for England might be another reason for this: there is only one sensor, located in a field outside Oxford. After inspecting a topographic map of the vicinity we noticed there seems to be a hilly forested ridge separating Heathrow airport from the only sensor in England. This would explain why we lose data from almost every plane coming into the final landing approach for that airport, even though the antenna is relatively close.

The number differences could definitely be seen as an indication that there is room for improvement in our landing detection algorithm, but one has to consider that some airports are located slightly outside of the coverage range of the network's antennas. It is very far from the closest antenna, which means that we lose data for incoming flights very early. That is, for flights that come from a direction where we have data coverage: if an airplane arrives from outside of sensor range we will completely miss it. In the case of Munich Airportfig. 7 we can realistically can not identify airplanes approaching from the east. One can see that there is some positions reported close to Munich, but on investigation, these turn out to be planes at cruising altitude. We only identify landings in Munich when the planes approach from an area of antenna coverage and drop by more than 1500 feet before we lose data. So, considering the available data, we think our algorithm works reasonably well.

To identify false positives in holding detection, we took a random sample of the identified holding patterns. For every second day, we randomly chose 4 planes that flew holding that day. We then extract the location data for these planes on the given days from the original data and inspect the time frame where our algorithm identified a holding pattern. This manual inspection is based mainly on latitude-longitude- and plane direction plots. After a couple of iterations on our algorithm using different samples every time, in our judgment, the algorithm does almost no false positives by now.

To identify false negatives, we use the same decoded data

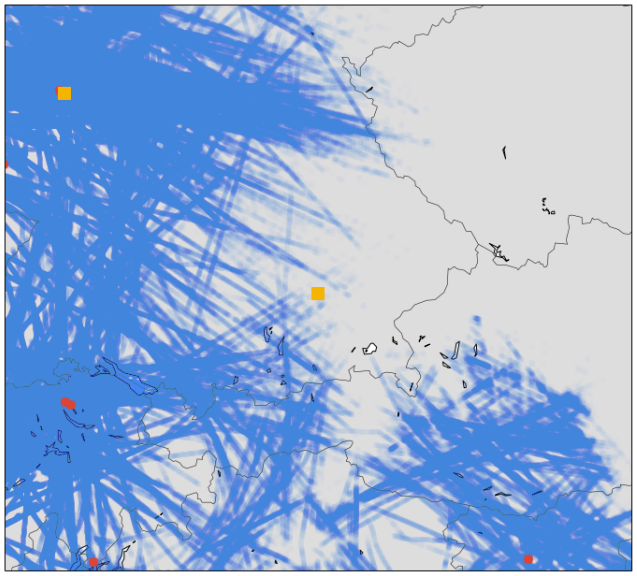

Figure 7: Positions (blue) reported close to Munich Airport (yellow, center), together with nearby antennas (red) and Frankfurt Main Airport (yellow, upper left)

and look at all the landings, for which no holding was identified. Here, we follow the same manual approach as for false positives. We do feel more confident about our algorithm by now, but there are still some cases where it is missing instances. For example, it would not detect the go-around on the left of fig. 6, which happened before the go-around. That is because the plane did turn less than 320°in the pattern. We are accounting for non-360°turns by setting the minimum turn down to 320°. Setting it lower would improve this and would probably not introduce false positives, as long as we do not set it too low. This might be an area for future improvement.

6. CONCLUSIONS

Our efforts have been effective. For areas where we have good sensor coverage our landing detection algorithm correctly identifies virtually every landing. Our holding detection algorithm reliably identifies holding patterns when there are data points from the holding pattern. For areas with excellent data coverage, our go-around detection reliably identifies go-arounds. Unfortunately, this is only true for a small number of airports as an antenna needs to be very close to the airport.

We created the rankings of airports and airlines by holding percentage and average time spent in holding. To present these results, we created a website, which allows users to sort and filter the data. We also provide the unfiltered original data as CSV. We decided not to compare airports or airlines by go-arounds as we have so few identified instances and can make reliable statements only about a very limited number of airports.

Unfortunately, there are many areas in Europe, which are poorly covered by the OpenSky network. As is showcased in fig. 2 and, on a grander scale, fig. 1 there is a lot of missing data. According to Strohmeier et al., only about 40% of European air traffic is gathered [6, p. 1]. This means that we do not have any basis to provide results for airports outside the sensor range shown in fig. 1. Additionally, we cannot provide reliable data for airports located in areas where sensor coverage is spotty, because they are at the border of the covered area. As described in section 5, this is definitely the case for Munich and Paris. For airports close to the coverage zone border, we lose data very early. So we might identify a landing, because we see a plane dropping for 1500 feet. But then we might lose data before the plane even starts holding. And it is very unlikely that we ever see a go-around for any of these airports without excellent coverage, as we do not see any data point from the plane between the actual go-around and the repeated landing attempt.

So we assume that besides landings, holdings and go-arounds in total, we also underestimate the percentage of holding done and the average time spent in holding for these airports. We further underestimate those numbers for all airports due to the holdings we miss because they happened after the point in time where we lose data for a plane.

If one adds up the duration of all holding we have identified in the 29 days of data we have, you end at 65,749 minutes spent in holding. A plane's fuel consumption strongly depends on the type and size of the airplane, its cruising speed, its vertical rate and its load. But if we assume a very rough but reasonable estimate of per-minute consumption of 55 kilograms, that amounts to around 3,600 tons of fuel wasted in holding, around 125 tons every day. And these numbers can only act as conservative lower bounds, as we do not have data for all European airports and we very likely underestimate holding for all the airports in our data set.

We hope that the adoption of ADS-B in air-traffic control will enable new opportunities in aircraft scheduling, which might in turn reduce the amount of fuel and time wasted in holding.

7. FUTURE WORK

Further work for this project would naturally improve our existing algorithms. In particular, the landing detection could benefit from some carefully applied extrapolation to figure out the exact landing position and altitude. A solution for this could perhaps use velocity messages with their heading and vertical rate to make up for missing position messages: extrapolating a Bézier curve leading to the ground, or defining an area and window of time in which a plane is likely to land are both approaches that could work well for the problem. Altitude data for the surface terrain could be taken into account as well. Should we not be able to acquire a freely available database for correlating destinations with call signs or *ICAO*s, we could also use this to help identify exact destinations.

Holding detection could be improved by using more of the facts we identified and combining these different indicators in a sensible way. Also, one could imagine using Machine Learning approaches to identify holding, as for example the Data Mining approaches described in section 2.2.1, using the the identified holding patterns as a sample of learning instances.

There are possible optimizations to the large-scale implementations of our algorithms, which would improve the performance of our program when applied to larger sets of data. This might include pre-organizing the data into partitions

of data that belongs either to one period of time or to a specific *ICAO*. If these partitions are kept at the same node, this would drastically reduce the need for reshuffling data when running our extraction. Smarter handling of the Scala/Java interaction might also improve performance. When applying the extraction to a larger geographic area, the distance checking to airports might get more expensive as there are more airports involved. This could be improved by utilizing quad trees or similar optimized data structures for distance checking. If more advanced detection techniques are to be implemented, these kinds of seemingly minor optimizations are surely needed so as to not unnecessarily bloat the run time of the analysis.

Ideas that might build on top of this project could be using our data to more accurately calculate pollution generated as a result of the extra time spent holding or reattempting landings. Further analysis could investigate ways for planes to prevent holding by slowing down, flying different routes, taking off later or other measures.

8. ACKNOWLEDGMENTS

We would like to thank: Max Haas, First Officer for Swiss International Air Lines, for advising us on aircraft procedures and for providing us with information on the peculiarities of Central European airports; and Peter Boncz and Hannes Mühleisen for teaching the 2016 LSDE course and counseling us on the specifics of the project.

9. REFERENCES

[1] J. A. Donoghue. The rare go-around. http://flightsafety.org/aerosafety-world-magazine/march-2012/the-rare-go-around. *possibly anecdotal* – Accessed: 2016-03-17.

[2] J. Han, Z. Li, and L. A. Tang. *Database Systems for Advanced Applications: 15th International Conference, DASFAA 2010, Tsukuba, Japan, April 1-4, 2010, Proceedings, Part II*, chapter Mining Moving Object, Trajectory and Traffic Data, pages 485–486. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[3] X. Liu, J. Biagioni, J. Eriksson, Y. Wang, G. Forman, and Y. Zhu. Mining large-scale, sparse gps traces for map inference: Comparison of approaches. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pages 669–677, New York, NY, USA, 2012. ACM.

[4] Airports Council International. Preliminary world airport traffic and rankings 2013.

[5] J. Shi, Y. Qiu, U. F. Minhas, L. Jiao, C. Wang, B. Reinwald, and F. Özcan. Clash of the titans: Mapreduce vs. spark for large scale data analytics. *Proc. VLDB Endow.*, 8(13):2110–2121, Sept. 2015.

[6] M. Strohmeier, I. Martinovic, M. Fuchs, M. Schäfer, and V. Lenders. Opensky: A swiss army knife for air traffic security research. In *Digital Avionics Systems Conference (DASC), 2015 IEEE/AIAA 34th*, pages 4A1–1–4A1–14, Prague, Czech Republic, Sept 2015.